Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

# Design and Simulation of a Machine-learning and Model Predictive Control Approach to Autonomous Race Driving for the F1/10 Platform

Alexandru Tătulea-Codrean,[*] Tommaso Mariani,[*] Sebastian Engell[*]

[*] *Dynamics and Operations Group, Technische Universität Dortmund, Germany (e-mail: { alexandru.tatulea, tommaso.mariani, sebastian.engell } @tu-dortmund.de).*

**Abstract:** This paper addresses the challenges of developing an embedded non-linear model predictive control (NMPC) solution for the optimal driving of miniature scale autonomous vehicles (AVs). The NMPC approach lends itself perfectly to driving applications, provided that a system for localization and tracking of the vehicle is available. An important challenge in the implementation results from the need to accurately steer the vehicle at high speeds, which requires fast actuation. In this paper we present a solution to this problem, which employs an artificial neural network (ANN) controller trained with rigorous NMPC input-output data. We discuss the development process, from modelling until the realization of the ANN controller within the operating system of the AV. The procedure is demonstrated within the virtual environment of the popular F1/10 race car, an AV platform widely used in AI and autonomous driving challenges. The results contain both NMPC and ANN-based simulations for different race tracks and for different driving strategies. The main focus of this work lies in the formulation of the optimal driving control problem and the training method of the ANN. Our approach uses a standardization of the driving problem, which enables us to abstractize optimal driving and to simplify it for the learning process. We show how driving patterns can be learned accurately on a reduced set of training data and that they can subsequently be extended to new and more challenging driving situations.

*Keywords:* NMPC, artificial neural networks, learned control, F1/10, AV.

## 1. INTRODUCTION

The F1/10 cyber-physical platform (see O'Kelly et al. (2019)) was first introduced in 2015 as an open source learning platform, drawing inspiration from events like the DARPA challenges and the Google autonomous driving challenge. With its added aspect of being also a student competition, it offers an exciting learning environment for those passionate about control, autonomous driving and artificial intelligence.
The challenge is primarily posed as an "autonomous driving" one, with control coming in as a secondary, but necessary, task. Therefore the research work published so far in the F1/10 community has predominantly been focused on solutions for positioning and localization, which constitute a crucial part of any autonomous driving platform. In the driving department, often the approach consists of a two-layer structure, where an upper-layer algorithm analyses the sensor data and computes a local path, which is then tracked by a low-level controller. Some examples of this approach can be found in O'Kelly et al. (2019). These results show that this approach is very robust and efficient, offering a sufficiently good solution for the competitive environment. However, outside the explicit scope of F1/10 racing, other scientific results have tackled the problem of autonomous driving as a compact optimization problem and tried to solve it accordingly. Paden et al. (2016), Plessen (2019) or Kabzan et al. (2019) summarize some of the latest research in autonomous driving and propose learning-based solutions for modelling the non-linear behaviour of vehicles and for solving the planning problem. Meanwhile, other authors as e.g. Zanon et al. (2014) and Kong et al. (2015) have shown how race driving can be treated as an non-linear model predictive control problem and be solved rather efficiently on low-resources embedded hardware.
In this work, we present a viable alternative to the previous results, which introduces a simple, yet effective way of posing the driving problem and its constraints by using local track data obtained from the LIDAR system. The approach starts by constructing a non-linear model of the race car and adapting its parameters to given driving behaviour and track conditions. This non-linear model is used in an NMPC structure in order to generate the optimal driving policies, which are tested and tuned in a realistic simulation environment. The second step is to use the simulation results of the NMPC in order to train an artificial neural network (ANN) to replace the predictive controller. Because the optimal controller must directly set the inputs of the car, the computation times are required to be very small, so that the controller can effectively steer the car in the real-time environment. It has been shown previously by, e.g Chen et al. (2019) that ANNs can replace linear MPC controllers. Furthermore, applications of machine learning approaches have been reported for more complex NMPC challenges, like the complex chemical reactor in Lucia and Karg (2018). The previous results inspired confidence towards solving the optimal autonomous driving as a machine learning approach for the F1/10 test case.

## 2. THE F1/10 SIMULATION PLATFORM AND THE DYNAMIC MODEL OF THE CAR

The F1/10 learning platform is built around low cost and widely available resources for AV building and control. Researchers are encouraged to test their algorithms in an open-source simulation environment which is curated by the F1/10 community and is very well documented (see O'Kelly et al. (2020)). The platform is built around ROS -Robot Operating System (Stanford Artificial Intelligence Laboratory et al. (2018)), a common framework for robotics systems applications. A simplified version of the F1/10 ROS platform is depicted in Fig. 1. It permits seamless switching between the simulation model and the real AV. Furthermore, the community manages several simulation workspaces for F1/10 development. It is important to note that the platform also simulates a realistic data gathering module, which employs the LIDAR and SLAM algorithms described by Kohlbrecher et al. (2011). In this work, the ROS simulations were done in the Gazebo simulator (Koenig and Howard (2004)), using the model developed for the Shukla et al. (2018). In order to begin designing a model predictive controller, one

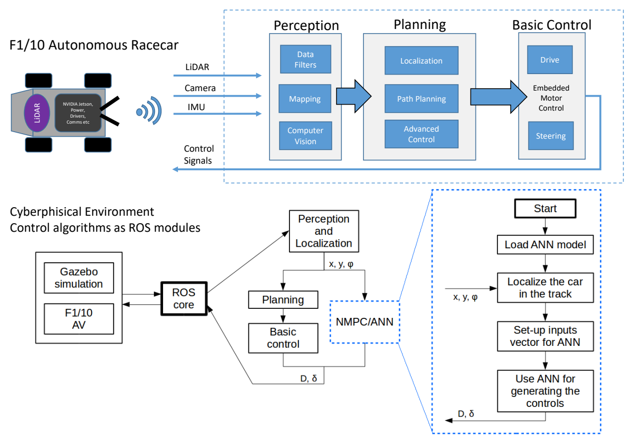

Fig. 1. Architecture of the F1/10 software platform, including information flow and a conceptual overview of the ROS configuration. The Gazebo model replaces the real AV in our simulation studies.

first needs an accurate dynamic model of the car. In this work we have followed one of the most common approaches for the modelling of 4-wheeled vehicles, which involves a simplification of the system to a 2-wheeled bicycle model. Similar approaches have been adopted by Zanon et al. (2014), Kong et al. (2015) or Kabzan et al. (2019), with different levels of detail. The physical parameters of the AV are based on the Traxxas 1/10 Scale Ford car model depicted in O'Kelly et al. (2019). For the motor and tire parameters, we have considered the Gazebo model as the benchmark and used it to fit the parameters. By acquiring position, velocity and orientation measurements, we set up a least-squares minimization optimization in order to obtain their values. This procedure is not detailed further due to limited space. The resulting values are listed in Tab. 1. The model is based on Liniger et al. (2015) and consists of 6 ordinary differential equations (ODEs), with the state vector $\underline{\psi} = [x, y, v_x, v_y, \phi, \omega]$. The steering angle and the motor duty cycle are the inputs used by the controller in order to drive the car: $\underline{u} = [\delta, D]$. The model equations are:

Table 1. Parameters of the dynamic model

| Name | Description | Value | Unit |
|---|---|---|---|
| $m_{car}$ | Total mass of AV | 4.0 | $kg$ |
| $I_{car}$ | Moment of inertia | 0.1217 | $kg \cdot m^2$ |
| $C_{r0}$ | Friction coefficients | 0.6 | $1/m$ |
| $C_{r2}$ | - | 0.1 | $m/s^2$ |
| $C_{m1}$ | Motor coefficients | 1.8073 | $m/s^2$ |
| $C_{m2}$ | - | -0.2511 | $1/s$ |
| $l_{car}$ $w_{car}$ | Length, width of AV | 0.535 , 0.281 | $m$ |
| $l_f$ $l_r$ | Axel distance to COG | 0.164 , 0.160 | $m$ |
| $B_f$ $B_r$ | Tire coefficients | 29.495 , 26.9705 | – |
| $C_f$ $C_r$ | - | 0.0867 , 0.1632 | – |
| $D_f$ $D_r$ | - | 42.5255 , 161.585 | – |

$$
\begin{aligned}
\frac{dx}{dt} &= v_x \cos\phi - v_y \sin\phi \\
\frac{dy}{dt} &= v_x \sin\phi + v_y \cos\phi \\
\frac{dv_x}{dt} &= \frac{1}{m}\left(F_{x,r} - F_{y,f}\sin\delta + m v_y \dot{\phi}\right) \\
\frac{dv_y}{dt} &= \frac{1}{m}\left(F_{y,r} - F_{y,f}\cos\delta - m v_x \dot{\phi}\right) \\
\frac{d\phi}{dt} &= \omega \\
\frac{d\omega}{dt} &= \frac{1}{I_z}\left(F_{y,f} l_f \cos\delta - F_{y,r} l_r\right),
\end{aligned}
\tag{1}
$$

where:

$$
\begin{aligned}
F_{x,r} &= (C_{m1} - C_{m2} v_x) D - C_{r2} v_x^2 - C_{r0}, \\
F_{y,f} &= D_f \sin(C_f \arctan(B_f \alpha_f)), \\
F_{y,r} &= D_r \sin(C_r \arctan(B_r \alpha_r)).
\end{aligned}
$$

The position of the car in 2D space is given by the vector $P = (x, y)$, the axial velocities are $v_x$ and $v_y$, respectively, and $\phi$ and $\omega$ represent the orientation and the angular velocity of the car. The tire model is based on the Pacejka Magic Tire Formula (Pacejka and Bakker (1992)) and Liniger et al. (2015). The parameters in the equations of $F_{x,r}$, $F_{y,f}$ and $F_{y,r}$ reflect the surface and driving parameters implemented in the Gazebo simulator.

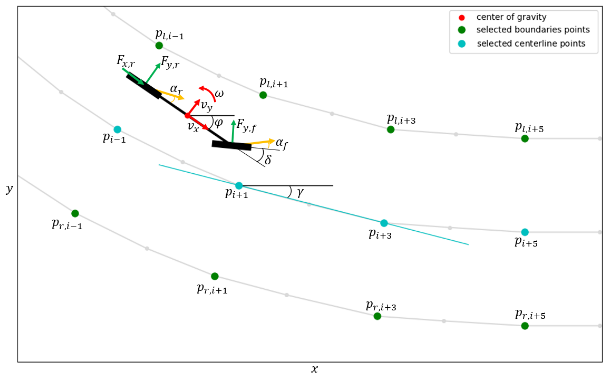

Fig. 2. A representation of the bicycle model in 2D space. The track and boundary points are also represented.

## 3. AUTONOMOUS DRIVING AS AN OPTIMAL CONTROL PROBLEM

The model presented in the previous section was implemented in Python with CasADi (Andersson et al. (2019)), a symbolic framework for simulation and optimization. The CasADi framework enabled us to use automatic differentiation and packages like SUNDIALS (Hindmarsh et al. (2005)) for simulating the model and IPOPT (Wächter and Biegler (2006)) for solving the resulting optimization problems. The work presented in this contribution also makes use of the *do-mpc* platform (Lucia et al. (2017)) to implement driving as an NMPC problem. In the following we will discuss the implementation of autonomous driving as an optimal control problem, by addressing the formulation of the driving constraints and the objective function of the controller.

One of the main challenges for autonomous driving is to steer the car through the track boundaries, defined in the F1/10 challenge by side-walls along the track. The data provided by the SLAM algorithm is used to obtain a 2D projection of the track on the floor and a center-line of the drivable track region. Furthermore, the algorithms also deliver information about the current position and orientation of the car. We do not provide further detail about the SLAM algorithm, as this is not the focus of this work and has been discussed in previous works related to the F1/10 platform. We assume that the full state of the car and the track information can be obtained from the simulation platform.

The top part of Fig. 3 illustrates an example of a track, with the car represented by a rectangle. The green points to the left and right are sampled from the track data and will be used further to define the left and right track boundaries. The center-line of the track is also marked in the figure, where the red dot represents the closest point to the car position on the center-line. Since the position and the heading of the car in the global coordinate system are known, it is possible to extract the relevant boundary points for the upcoming region of the track from the available data. The positioning data, together with the track boundaries and the car dynamics constitute the information needed in order to define the OCP problem. The main advantage of the formulation used in this work is that it requires a limited number of points around the current position of the car, which is closely related to the real-world scenario, where the AV will only "see" the portion of the track ahead of it. A mathematical expression for the two track boundaries is obtained by using the following interpolating non-linear polynomials:

$$\begin{aligned} \text{Left boundary:} g_L(x) &= c_{L3}\cdot x^3 + c_{L2}\cdot x^2 + c_{L1}\cdot x + c_{L0} \\ \text{Right boundary:} g_R(x) &= c_{R3}\cdot x^3 + c_{R2}\cdot x^2 + c_{R1}\cdot x + c_{R0} \end{aligned} \tag{2}$$

A linear transformation of the input data is performed, which results in the OCP always being posed with respect to the car's reference system. To that end, every labelled point in Fig. 2 must be rotated by an angle $\gamma$ between the orientation of the car and the x-axis and translated so that the car's position $p_{car} = [p_{i,x}, p_{i,y}]$ is located in the origin. This is a key aspect of the proposed approach and it leads to a local formulation of the OCP which is independent of the position on the track and the relative orientation of the car. It also leads to a direct translation to the training of the ANN network, as will be described in the next section. The transformed model reads:

$$\begin{aligned} x' &= \cos(\gamma)(x - p_{i,x}) + \sin(\gamma)(y - p_{i,y}) \\ y' &= -\sin(\gamma)(x - p_{i,x}) + \cos(\gamma)(y - p_{i,y}), \end{aligned} \tag{3}$$

where the variables $(x', y')$ represent the rotated and translated position for each point $(x, y) \in P$, as depicted in the lower part of Fig. 3. This transformation simplifies the NMPC problem and, at the same time, reduces the learning effort for the artificial neural network. Furthermore, the optimal control problem is formulated here as a simultaneous prediction and optimization problem of the form in (4), where the dynamic model of the car

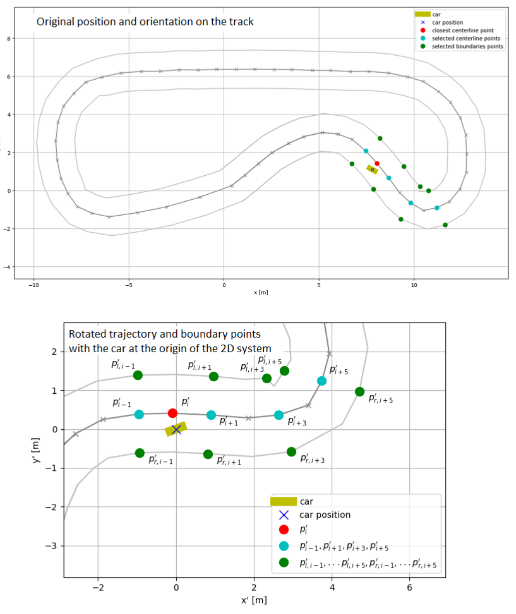

Fig. 3. Example transformation of the car and boundary coordinates that are used in the definition of the OCP.

and the track boundaries are used in order to define the feasible solution space.

$$\min_{\Psi_k, u_k} \sum_{k=1}^{N_{PH}} J_{cost}(\Psi_k, u_k), \tag{4}$$

subject to:

$$\begin{aligned} \Psi_{k+1} &= f(\Psi_k, u_k), && \forall (k+1) \in I, \\ g_L(x_k) - y_k &\le 0\,, && \forall\, k \in I, \\ g_R(x_k) - y_k &\ge 0\,, && \forall\, k \in I, \\ -\pi/6 \le \delta_k &\le \pi/6\,, && \forall\, k \in I, \\ -1 \le D_k &\le 1\,, && \forall\, k \in I, \\ \Psi_{lb} \le \Psi_k &\le \Psi_{ub}\,, && \forall\, k \in I, \end{aligned}$$

where $I$ is the set of all indices within the prediction horizon and $g_L(x)$ and $g_R(x)$ are the polynomials obtained from 2. The coefficients $c_{L,i}, c_{R_i}$ are fitted before each optimization step, taking into consideration new track data. In addition to the track boundaries, the state variables of the car and the inputs are restricted to finite sets which represent physical limitations on the variables.

The style of driving can be embedded in the OCP mainly in two ways. First the constraints in (2) can be set very close to the physical boundary detected by the SLAM algorithm, for an aggressive driving policy. Alternatively, a more conservative approach is achieved by restricting these boundaries to a narrower region around the middle line. Furthermore, the structure of the cost function plays a role in the optimal driving solution. In this work we have experimented with two types of cost functions, presented in (5). In order to implement the fastest possible driving strategy, the cost function in Eq. (5a) takes the farthest visible point in the environment which lies between the two track boundaries. This point is treated as the goal point, labelled $P_G = (x_G, y_G)$, and is updated in the cost function in each NMPC iteration. This method ensures that the optimizer continuously works towards finding the shortest and fastest path around the track. If one defines the driving as the task of tracking a predefined line on the track, then the cost function in Eq. (5b) can be seen as trying to minimize the distance between the current position of the car, $P_k = (x_k, y_k)$, and the target trajectory, which is implemented here by the polynomial $T(x_k)$. This is the same as reducing the distance between the current position of the AV and the line represented by the polynomial. The term $\Delta u_k$ represents the input change and is added to the cost function in order to penalize sharp changes in the control inputs and thus smoothen the driving behaviour.

Optimal driving :

$$J_{dmax}(x_k, u_k) = \sum_{k=0}^{N_{PH}} \left[\omega_0((x_k - x_G)^2 + (y_k - y_G)^2) + \omega_1 \Delta u_k^2\right] \tag{5a}$$

Trajectory tracking :

$$J_{lt}(x_k, u_k) = \sum_{k=0}^{N_{PH}} \left[\omega_0 \sqrt{(T(x_k) - y_k)^2} + \omega_1 \Delta u_k^2\right] \tag{5b}$$

A comparison of the two driving modes can be seen in Fig. 4. In the top section we show a 2D plot of the track, in grey, and the trajectory of the car. The black lines are the NMPC results with the minimal time approach, whereas the green lines represent the line tracking NMPC. As it was the case with all the results presented here, both algorithms were run for a fixed number of iterations and the final positions are depicted by colored starts in the figure. It can be seen that the two strategies are slightly different, especially around corners. It is expected that the first formulation will produce a faster, more "aggressive" style of driving, which is what we can see in the figure. However, the trajectory tracking strategy also works well and it would result in different driving patterns, if the trajectory was changed. From an optimization point of view, it is better to define general constraints and let the optimizer find the optimal trajectory for the car, as well as the optimal input trajectory. Any other method would, theoretically, result in sub-optimal control inputs. Therefore we propose the first driving method as a template solution for fast driving. The line tracking implementation is more conservative and it could provide a safer driving behaviour during training laps or mapping of the track.

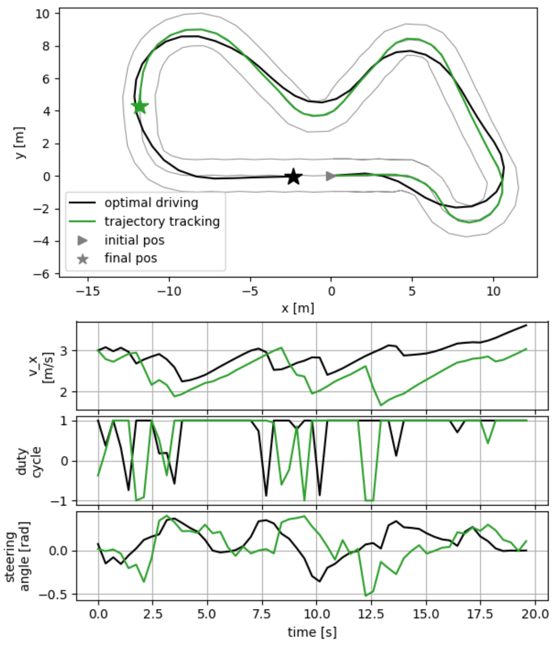

Fig. 4. Driving done by the NMPC controller: in black - optimal driving (aggressive) and in green - trajectory tracking (conservative).

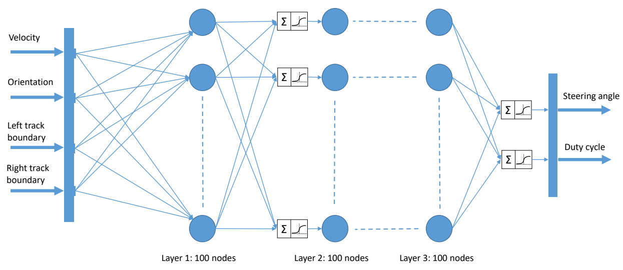

Fig. 5. An overview of the ANN structure and the input-output relationship.

## 4. THE ML APPROACH AND RESULTS

### *4.1 Training of the ANN controller*

The structure of the ANN employed in this work is given in Fig. 5. It depicts a fully-connected feed-forward network consisting of 3 linear layers connected by non-linear ReLu nodes. The outputs of the network are the two control inputs of the car. The input structure is essential for a good performance of the system and is based on the transformed coordinates depicted in Fig. 3 and discussed in the previous section. Because of the way the driving problem is posed, only the velocity of the AV together with four left and right boundary points is needed. This is the minimum data necessary and it was found that it can, indeed, be used for successful training of the ANN. The input vector of the network thus consists of 18 individual values, which are also depicted in Fig. 5: $U_{ANN} = [v_x, v_y, pl_{i-1}, ..., pl_{i+3}, pr_{i-1}, ..., pr_{i+3}]$. The output of the network consists of only two values: $Y_{ANN} = [\delta, D]$. Therefore the network effectively implements a non-linear mapping between these two sets $F(U_{ANN}) = Y_{ANN}$. This mapping must be learned by the ANN controller in the training process. In contrast to the example given by Lucia and Karg (2018), in this work the complexity of the ANN is much lower, as the driving problem is comparatively an easy one. The number of data pairs needed for training is also much lower, because the number of possible situations is restricted and we benefit from the ability to quickly re-sample the data and compute new control moves. The training data was obtained by running 1500 individual iterations of the NMPC formulation with the cost function in Eq. 5a on the track depicted in Fig. 4, which results in pairs of input-output data of the type presented above. The curvature of the road and the speed of the car determine how much the controller will have to decelerate and steer, which gives rise to certain patterns in the driving behaviour. We have depicted such an example that emerges from the training data in Fig. 6. The crosses in the image depict clouds of boundary points for all the cases where the the NMPC controller decided to steer the car left at an angle $0.25rad \le \delta \le 0.30rad$. The point clouds clearly form the shape of a mild left turn in the track, which represents one of the rules that the ANN must learn. Similar patterns will occur for other situations and it is sufficient to train the ANN with relevant left and right turns. More data could help train a better ANN in the future. However, it was found that for this application a simple track like the one in Fig. 4 is sufficient.

The ANN controller was implemented in the open-source platform PyTorch, a Python module that can be freely downloaded from Paszke et al. (2019) and is detailed by its authors in Paszke et al. (2017). The training was done using a standard approach, consisting of a mean square error (MSE) loss function. The ANN parameters were optimized using Adam (Kingma and Ba, 2014), an optimizer that is supported by default in PyTorch. The training was executed over 5000 epochs, with the overall training accuracy reaching 98.7% at the end of the training. Fig. 7 shows how the ANN performs versus the NMPC controller. Both controllers were started from the same initial position and were required to complete one lap of the test track. Overall, the ANN controller performs very well. For this track, an average error of approximately 3.75% exists between the two control results. However, this does not translate into significant trajectory differences. Because the ANN controller takes a slightly different trajectory, its outputs should not be directly compared to the ones obtained in training from the NMPC controller.

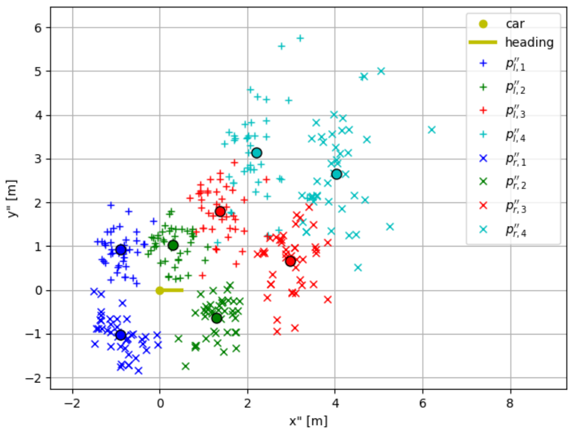

Fig. 6. Example of training data patterns. Each point represents a track constraint given to the NMPC controller. The points form clouds that can be approximated by track boundary points which depict a mild left turn.

### 4.2 *Testing of the controller on new tracks*

To further analyse the robustness of the ANN controller to changes in the driving situations, we tested the trained controller on a completely new track, which is longer and more challenging, but which poses a similar level of cornering difficulty. The results are depicted in Fig. 9, where one can see the F1/10 car driving around a scaled down version of the Nürburgring F1 circuit. Although the controller was not trained with optimal driving data for this particular track, it can successfully navigate it. The average error between the two control moves for this track did increase to 7.3%, but the ANN performed well without having a priori track data. We used this new track to test the robustness of the ANN controller and found that the limited training dataset was robust enough for small changes of the track. However, in applications with the F1/10 car, we recommend gathering data about the new tracks and re-training the controller before deploying it.

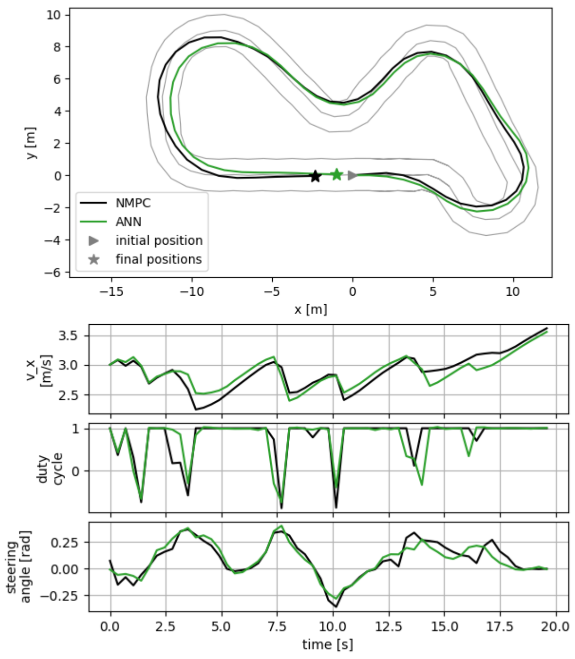

Fig. 7. Driving comparison between: black - the optimal driving NMPC controller and green - the machine-learning ANN approach. Both controllers were run for a fixed time of 20 seconds.

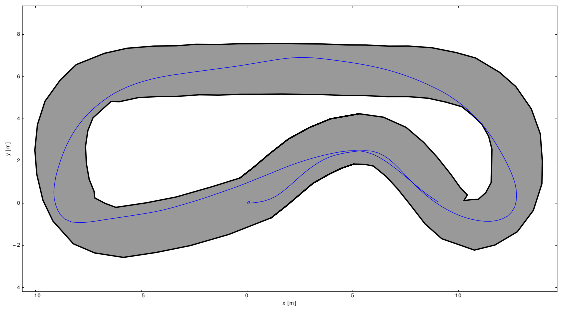

Fig. 8. Further results: simulation test with the ANN controller on the 2018 Porto F1/10 race track. Top view of the track and of the AV trajectory obtained in the real-time simulation environment Gazebo.

## 5. CONCLUSIONS

We have presented an NMPC implementation for a small-scale race car model, showing how a non-linear model can be employed for optimal driving around a known track. The driving problem was transformed into a dynamic optimization problem by transferring all positional aspects of driving around a central frame, with the car at the origin of the system. Therefore, the track can be easily represented with the help of two polynomials which are interpolated through points on the track boundaries. We showed that this strategy results in an accurate representation of the track and can be used in order to shape the optimal solution of the NMPC. However, the resulting controller is too computationally demanding for an online implementation of the controller on the on-board hardware. Therefore, our approach adopts a machine-learning based solution, where the optimal results generated by the NMPC controller are used as a training data set for a simple artificial neural network. By training the ANN to learn the optimal behaviour, one can replace the NMPC controller with a

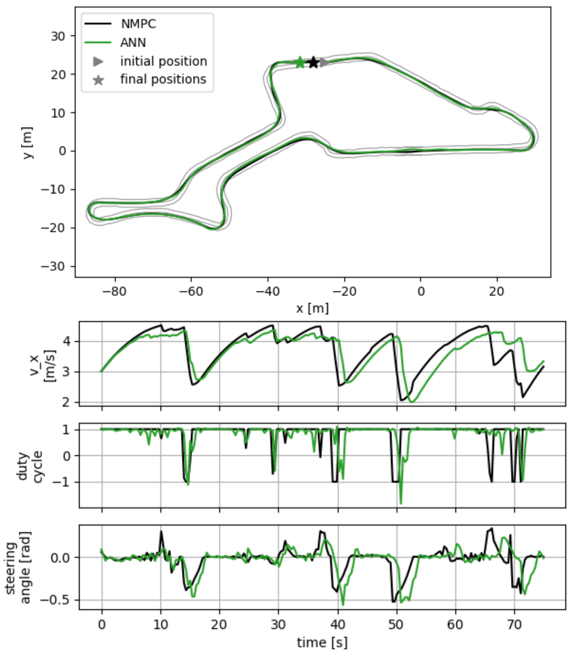

Fig. 9. Driving test on a scaled down version of the Nürburgring circuit. Although the ANN was not trained on this track, it can successfully navigate it at the first attempt.

good control performance. We showed that the ANN controller is robust to changes in the track shape and can be applied in a competitive environment, even on unknown tracks. Therefore, we are confident that the approach can be integrated with results obtained by other groups in the F1/10 community. It must be noted, however, that the proposed solution is case dependent and it assumes that the non-linear constraints can approximate well the real track boundaries. Similarly the cost function formulation does not guarantee a globally optimal path, but has shown it can generate good results on complex tracks. This two-staged approach can be used for any NMPC scenario, regardless of the underlying car model or the optimizer used in the training.

## REFERENCES

Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., and Diehl, M. (2019). Casadi- a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*.

Chen, S., Wang, T., Atanasov, N., Kumar, V., and Morari, M. (2019). Large scale model predictive control with neural networks and primal active sets. arXiv:1910.10835v1.

Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., and Woodward, C.S. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3), 363–396.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M. (2019). Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4).

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Kohlbrecher, S., Stryk, O., Meyer, J., and Klinglauf, U. (2011). A flexible and scalable SLAM system with full 3D motion estimation. In *IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan*.

Kong, J., Pfeiffer, M., Schildbach, G., and Borrelli, F. (2015). Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, 1094–1099. doi:10.1109/IVS.2015.7225830.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5), 628–647. doi:DOI: 10.1002/oca.2123.

Lucia, S. and Karg, B. (2018). A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20), 511–516.

Lucia, S., Tatulea-Codrean, A., Schoppmeyer, C., and Engell, S. (2017). Rapid development of modular and sustainable nonlinear model predictive control solutions. *Computer Engineering Practice*, 60, 51–62. URL `https://github.com/do-mpc/do-mpc`.

O'Kelly, M., Abbas, H., Harkins, J., Kao, C., Vardhan, Y., Mangharam, R., Agarwal, D., Behl, M., Burgio, P., and Bertogna, M. (2019). F1/10: An open-source autonomous cyber-physical platform.

O'Kelly, M., Behl, M., Krovi, V., et al. (2020). F1Tenth website on readthedocs.io. URL `https://f1tenth.readthedocs.io/`.

Pacejka, H.B. and Bakker, E. (1992). The magic formula tire model. *Vehicle System Dynamics*, 21, 1–18.

Paden, B., Cáp, M., Yong, S.Z., Yershov, D.S., and Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1, 33–55.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS 2017 Workshop Autodiff*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2019). Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. URL `https://github.com/pytorch/pytorch`.

Plessen, M. (2019). Automating vehicles by deep reinforcement learning using task separation with hill climbing. In *FICC2019: Advances in Information and Communication*.

Shukla, M., the groups at UPenn, and MIT (2018). Ros gazebo simulation environment for f1tenth. URL `https://github.com/mlab-upenn/`.

Stanford Artificial Intelligence Laboratory et al. (2018). Robotic Operating System. URL `https://www.ros.org`.

Wächter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25–57. doi:10.1007/s10107-004-0559-y.

Zanon, M., Frasch, J., Vukov, M., Sager, S., and Diehl, M. (2014). Model predictive control of autonomous vehicles. In H. Waschl, I. Kolmanovsky, M. Steinbuch, and L. del Re (eds.), *Optimization and Optimal Control in Autonomous Systems. Lecture Notes in Control and Information Sciences*, volume 455, 41–57. Springer, Cham.