Real-time Nonlinear MPC Strategy with Full Vehicle Validation for Autonomous Driving

Jean Pierre Allamaa [1], Petr Listov [2], Herman Van der Auweraer [1], Colin Jones [2], Tong Duy Son [1]

Abstract— **In this paper, we present the development and deployment of an embedded optimal control strategy for autonomous driving applications on a Ford Focus road vehicle. Non-linear model predictive control (NMPC) is designed and deployed on a system with hard real-time constraints. We show the properties of sequential quadratic programming (SQP) optimization solvers that are suitable for driving tasks. Importantly, the designed algorithms are validated based on a standard automotive development cycle: model-in-the-loop (MiL) with high fidelity vehicle dynamics, hardware-in-the-loop (HiL) with vehicle actuation and embedded platform, and vehicle-hardware-in-the-loop (VeHiL) testing using a full vehicle. The autonomous driving environment contains both virtual simulation and physical proving ground tracks. Throughout the process, NMPC algorithms and optimal control problem (OCP) formulation are fine-tuned using a deployable C code via code generation compatible with the target embedded toolchains. Finally, the developed systems are applied to autonomous collision avoidance, trajectory tracking and lane change at high speed on city/highway and low speed at a parking environment.**

I. Introduction

Advanced vehicle control algorithms are crucial for the development of safe and reliable autonomous driving applications to reduce road accidents and causalities [1]. Conventional control designs such as PID and linear state feedback are often served in low-level feedback loops of industrial automotive applications. Albeit having low computational complexity, the performance of such controllers is limited in safety-critical traffic scenarios such as emergency collision avoidance. Recently, nonlinear model predictive control (NMPC) for autonomous driving applications has been studied and shown promising results, mainly from academic research [2], [3]. On the other side, due to the computational complexity and limited resources required by numerical optimization, NMPC has not been commonly considered in industrial autonomous driving control platforms [4].

The main contribution of this paper is an efficient development framework that can be used in the automotive industry to design and deploy NMPC on road vehicles. The intended application in this project is trajectory tracking in the presence of obstacles, hence, two test scenarios are created:

- Autonomous valet parking: to deal with safety around suddenly appearing pedestrians and vehicles. The driving takes place at a low speed, around 10kph. The controller is tested in a private parking area.
- Lane keeping: to avoid collision with suddenly appearing obstacles in city or highway scenarios. NMPC is tested on proving ground in Aldenhoven, Germany at high speeds around 60kph.

We present a framework for embedded NMPC in autonomous driving applications, providing a number of benefits: First, it is built upon a detailed nonlinear predictive model, capturing the system delays, actuator saturation and look-ahead capabilities to generate feasible trajectories that avoid dangerous driving situations. Second, constraints and objectives can be set for specific performance and driver comfort. NMPC is deployed in the loop with the vehicle, without any resampling, as a low-level controller calculating a control policy every iteration respecting hard real-time constraints. Real-time operation implies a dependency on the logical explanation of the solution and most importantly on the numerical optimization scheme execution time [5].

The development framework is motivated by the R&D ADAS team of Siemens, with focus on testing real-time NMPC strategies in different standard scenarios throughout XiL stages [6]. XiL verification is well-established as an effective means to develop safe and secure industrial automotive systems, as illustrated in Figure 1. In this systems engineering process, a model of the control system is first tested in simulation (MiL). MiL validation is conducted with high-fidelity multi-physics simulators using Simcenter Amesim with disturbances and parameter mismatches to test the algorithm's robustness. Once validated, C/C++ code is auto-generated and tested (SiL). Real-time performance is validated in a realistic virtual traffic environment and physics-based sensors in Simcenter Prescan. Then the generated code is integrated into ECU hardware (HiL) and eventually deployed in physical vehicle running on road (VeHiL). The plug-and-play framework requires limited workforce for the user from design to deployment on the vehicle. Real and virtual environment combined in XiL is attractive, as it helps reducing testing costs as performance is assessed without test sites and costly sensors/obstacles. Tuning campaigns are facilitated, development and implementation cycle time are decreased. Although tested with a specific toolbox and solver, the framework allows for an ease of reproducibility and expansion to other C/C++ capable toolboxes, solvers, OCP formulation, car models and scenarios.

Hardware setup comprises a Ford Focus vehicle, percep-

[1] Siemens Digital Industries Software, Leuven, Belgium
Email: {jean.pierre.allamaa, herman.vanderauweraer, son.tong}@siemens.com
[2] Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland
Email: {peter.listov, colin.jones}@epfl.ch

This work is part of FOCETA project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123.
Abstract video at: https://youtu.be/tCrn_7331xY

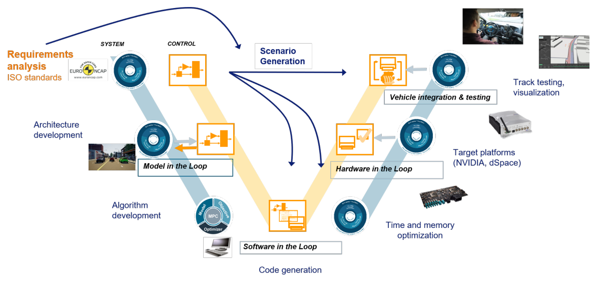

Fig. 1: XiL development framework

tion and localization sensors (GPS, Radars, Camera, LiDAR), driving robot (Anthony Best Dynamics) for actuating steering (SR), throttle (AR) and brake systems (BR), and an embedded platform dSPACE MicroAutobox III as the NMPC computation module. They are represented in Figure 7.

The paper is organized as follows. Section II discusses some background on NMPC for autonomous driving. Section III presents the hardware deployment of optimization solvers for collision avoidance application. Validation results with both virtual and real obstacles are given in Section IV.

II. Background

This section presents the vehicle model and the developed NMPC formulation for trajectory following. We then provide background on SQP solvers and explain the collision avoidance planning algorithm.

A. Bicycle Model

Car dynamics are represented with a real-time feasible model such as the 6 DoF bicycle model. Controller validation with a 15 DoF model in Amesim is later performed in a MiL framework. From experimental validation, the planar bicycle model defined below is satisfactory for lane keeping and emergency scenarios, assuming no effect of roll and pitch on lateral dynamics.

$$\begin{aligned}
\dot{v_x} &= \frac{1}{M}(F_{xf}cos\delta + F_{xr} - F_{yf}sin\delta - F_{res} + M\dot{\psi}v_y),\\
\dot{v_y} &= \frac{1}{M}(F_{xf}sin\delta + F_{yr} + F_{yf}cos\delta - M\dot{\psi}v_x),\\
\dot{r} &= \frac{1}{I_z}(L_f(F_{yf}cos\delta + F_{xf}sin\delta) - L_rF_{yr}),\\
\dot{X} &= v_xcos\psi - v_ysin\psi,\\
\dot{Y} &= v_xsin\psi + v_ycos\psi,\\
\dot{\psi} &= r,
\end{aligned} \tag{1}$$

where X, Y and ψ are the vehicle's CoG location and heading in the Cartesian global frame. v_x, v_y, and r are the longitudinal, lateral velocities and yaw rate given in the local body reference frame. Using two frames helps to locally limit rates, velocities and control actions and globally track a spatial reference. Longitudinal dynamics take into consideration air drag on the road vehicle:

$$F_{res} = F_{drag} = C_{r0} + C_{r2}v_x^2. \tag{2}$$

The car is controlled by longitudinal acceleration applied by the driving robot, assuming the front and rear axles' longitudinal forces are identical and proportional to the engine torque. With small slip angles assumption, linear tire model is used with the lateral forces being proportional to the slip angles via the cornering stiffness K_f and K_r:

$$F_{xf} = F_{xr} = 0.5\frac{t_rT_{max}}{R}, F_{yf} = K_f\alpha_f, F_{yr} = K_r\alpha_r. \tag{3}$$

The front and rear slip angles can be defined as follows:

$$\begin{aligned}
\alpha_f &= -\tan^{-1}\left(\frac{\dot{\psi}L_f + v_y}{v_x}\right) + \delta,\\
\alpha_r &= \tan^{-1}\left(\frac{\dot{\psi}L_f - v_y}{v_x}\right).
\end{aligned} \tag{4}$$

Control actions are the normalized throttle t_r with respect to the maximum engine force, and the body reference frame steering angle δ.

B. Nonlinear Model Predictive Control

NMPC directly controls throttle and steering in this application. A receding horizon scheme of N steps is used. Continuous dynamics model in Equation (1) are first discretized using a Runge-Kutta 4^{th} order method, then the NLP optimizes over the discrete time OCP. The OCP for trajectory following is:

$$\begin{aligned}
&\min_{x(0,\dots,N),u(0,\dots,N-1)} \sum_{k=0}^{N-1} l_k(x_k,u_k) + V_f(x_N)\\
&\text{subject to: } x_0 = x(0)\\
&x(k+1) = f(x(k),u(k)) \qquad k = 0,..,N\\
&v_{x,min} \leq v_x(k) \leq v_{x,max} \qquad k = 0,..,N-1\\
&v_{y,min} \leq v_y(k) \leq v_{y,max} \qquad k = 0,..,N-1\\
&\dot{\psi}_{min} \leq \dot{\psi}(k) \leq \dot{\psi}_{max} \qquad k = 0,..,N-1\\
&\delta_{min} \leq u_1 \leq \delta_{max} \qquad k = 0,..,N-1\\
&-1 \leq u_2 \leq 1 \qquad k = 0,..,N-1\\
&x_N \in \chi_f.
\end{aligned} \tag{5}$$

The stage cost $l_k(x_k,u_k)$ is defined as:

$$\begin{aligned}
l_k(x_k,u_k) = &(x(k) - x_{ref}(k))^TQ(x(k) - x_{ref}(k))\\
&+ u(k)^TRu(k) + \Delta u(k)^TS\Delta u(k),
\end{aligned} \tag{6}$$

with $Q \in \mathbb{R}^{6\times6} \succeq 0, R \in \mathbb{R}^{2\times2} \succ 0, S \in \mathbb{R}^{2\times2} \succeq 0$, and $x_{ref}(k)$ is the reference path and velocity profile from the planner. The lateral velocity and yaw rates have a zero reference. In this application constraints on the inputs rate are softened and added in the cost function.

C. SQP and QP solvers

Sequential quadratic programming is a popular optimization method thanks to its ability to handle highly nonlinear problems and to efficiently warm-start [7]. In this application, an SQP with QRQP quadratic solver from CasADi is used [8]. QRQP is based on the sparsity exploiting active-set method and the derivatives are produced by the CasADi toolbox [9], an open source symbolic software framework for nonlinear programming. This paper does not provide a benchmark/comparison of different solvers for automotive

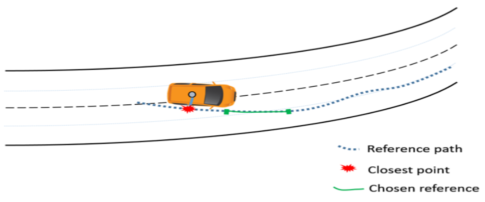

Fig. 2: Planner: closest point localizer

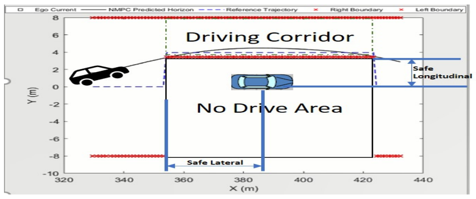

Fig. 3: Planner: safe corridor

applications, but rather implements an out-of-the box solver to demonstrate the framework. The choice of toolbox and solver is just a placeholder for any other library that is either written in C/C++ or has C code generation capabilities ([10], [11]).

D. Collision avoidance planner

The local planner takes as input a collision-free reference trajectory in X, Y, Yaw and the longitudinal velocity as discrete chunks or continuous splines. It localizes the car with respect to the closest point on the path and outputs the current position along the centerline with a reference to be tracked in the time-horizon ahead, as shown in Figure 2. The planner for collision avoidance dynamically shifts the reference and boundaries of the previous iteration to a safe lateral distance, creating a driving corridor off all obstacles as seen in Figure 3. The lateral and longitudinal safe distances create a no-go zone and can be changed online via tunable parameters to mimic different reaction times or distances. For scenarios presented in this paper, 1.2 to 1.5 seconds of safe longitudinal duration have proven to be sufficient for the car to react to a suddenly appearing obstacle.

III. Embedded NMPC Implementation

This section deals with the deployment steps on dSPACE MicroAutobox III (MABX-III) hardware for an embedded control application and validates MiL/HiL in the virtual environment of Prescan. First, the controller is prepared for a real-time environment on a platform with a C/C++ compiler: MABX-III has a ARM Cortex A-15 processor, operates on a 2GB DDR4 RAM with 64MB flash memory for real-time application and runs RTOS. Deployment to MABX-III is done via Simulink interface. This chapter explains how the tailored NMPC is C-code generated, tested in Simulink and deployed as a standalone library in runtime. Turning the optimization function into C-code enhances performance as there will be no callbacks into Matlab environment, and static memory allocation of the block's internal states is done on compile time. Second, the generated code is validated in MiL simulation with Amesim as in Figure 4.

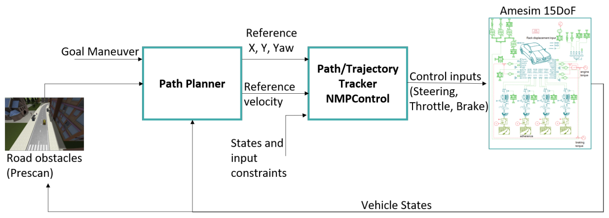

Fig. 4: MiL: closed-loop control structure

A. Code generation for standalone NMPC

NMPC for trajectory following is written in Matlab using CasADi. The challenge resides in code generating the complete Simulink model to be deployed on MABX-III and the proposed procedure is depicted in Figure 5. Particularly for this application, the C source codes are compiled using MEX (with MSVC: Microsoft Visual Studio or MinGW64 compilers), and other compilers such as GCC could be used depending on the target platform. For an OCP written in C/C++, the code can be called directly using *S-function builder*. The MEX executable is compiled from the NMPC and wrapper source codes, linked with the toolbox's implementation code, and is called using an *S-Function block*. Therefore, the MEX binary is a self contained library. The following comments are to be stated for this project implementation:

1) CasADi code generation toolbox is used
2) The wrapper output step is a functor from current states and references, to the open-loop primal solution and optimal policy (first control action)

B. MiL in a virtual environment

The first test evaluates the closed loop performance, with high fidelity vehicle dynamics (15DoF) from Amesim, with output noise and parameter mismatch, before being deployed on the car. Simulations are carried out to test the NMPC in trajectory following and emergency collision avoidance applications in an ISO 3888-1 standard double lane change scenario at 80kph. Results are shown in Figure 6, showing high performance tracking response. NMPC is solved to convergence and satisfies the real-time constraints.

The system profiler (Simulink or MSVC profiler), indicates that the code generated NMPC execution time averages at 2.4ms. NMPC profiling with and without code generation, is interpolated and presented in Table I. Since MABX-III has limited computational power compared to the host PC (with

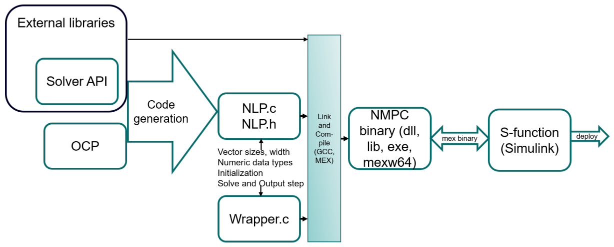

Fig. 5: HiL: NMPC deployment steps

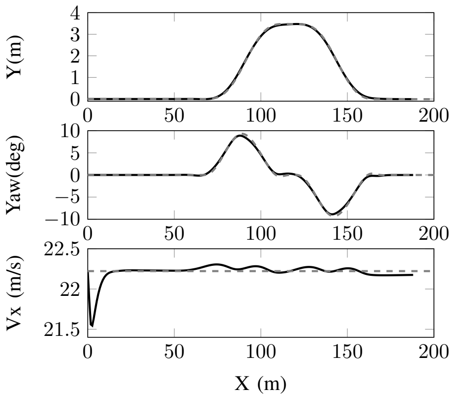

Fig. 6: MiL: vehicle states (solid) and reference (dashed)

32GB RAM), it is necessary to understand how execution time scales on the platform. Violating real-time performance on host PC necessarily indicates violation on MABX-III. After running several HiL tests, it is found that for the considered scenario, execution time is increased by a factor between 7 and 10 times on MABX-III. Therefore, the first real-time performance check before deployment is for the NMPC to run on the host PC with an execution time below 4.5ms for a sample time of 40ms. Otherwise the NMPC is real-time incapable and need to be redesigned.

C. HiL in a virtual environment

HiL validation takes the same virtual scenario in MiL, however controls are applied to a vehicle physically lifted off the ground. Controller real-time capabilities and control signal smoothness are the main targets. Two important aspects are required before deployment: a. all Simulink blocks are code generatable, b. Simulink model and NMPC formulation are code optimized for quickest execution time per iteration. Code generating the optimization function speeds up the evaluation time from 4 to 10 times as compared to the evaluation in Matlab [9]. The following properties are used in the NMPC compilation:

- Solver: SQP method with QRQP (Active-Set method)
- Maximum number of SQP iterations: 50
- Maximum number of QP iterations: 100
- Integration type: Runge Kutta 4 (RK4)
- Hessian approximation: Exact
- OCP method: Direct (discretize then optimize)

Type:	No code generation	Code generation (per evaluation)
Total	22.0ms	2.4ms (2.4ms)
QP	2.00ms	0.218ms (0.109ms)
Line search	2.00ms	0.218ms (0.109ms)
Cost and constraints evaluation	2.00ms	0.218ms (0.109ms)
Gradient evaluation	1.00ms	0.109ms (0.109ms)
Hessian evaluation	8.00ms	0.87ms (0.436ms)
Jacobian evaluation	5.00ms	0.545ms (0.27ms)

TABLE I: Profiling optimization functions with and without code generation

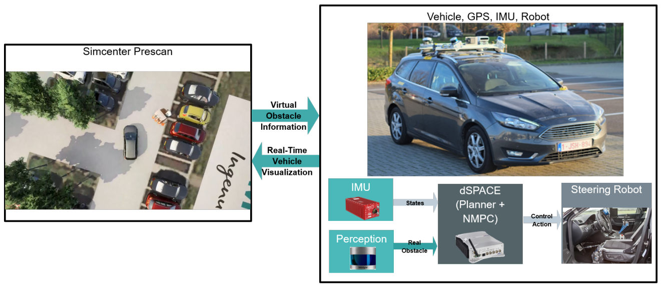

Fig. 7: VeHiL: communication and vehicle framework

- Shooting method: Multiple shooting
- Sample time: 40ms (25Hz controller)
- Prediction horizon: N = 30
- Primal and dual infeasibility threshold: $1e^{-06}$ and $1e^{-04}$

Possible improvements to reduce computation time are:

1) Warm start as SQP is heavily affected by initialization
2) Reformulate the OCP to relax non critical active constraints, add slack variables or add them to the cost
3) Scale the problem in order to improve conditioning
4) Reformulate the OCP in a matrix form and minimize nested for-loops to facilitate derivative calculations

The second test towards full vehicle deployment, is a validation of the NMPC on MABX-III for highway and cut-in scenarios. The car is visualized in Prescan through a communication via ROS [12] as in Figure 7. Control action is applied on the driving robot and the test is validated as NMPC block operates in real-time with an execution time around 22ms for a 25Hz low-level controller.

IV. Vehicle Hardware in the Loop (VeHiL)

Real-time optimal control implementation for HiL and VeHiL requires communication among several hardware and software, as will be presented in this chapter and summarized in Figure 7. Physical testing results in a private parking area and on proving ground are presented. For HiL testing, the vehicle in Figure 7 is replaced by a simulator. Therefore, the communication architecture allows the user to go from offline simulation for NMPC design, to online MiL/HiL and finally to VeHiL with the exact same code.

A. VeHiL: Parking validation

In this testing campaign, the vehicle is integrated in the loop with the embedded controller for parking scenarios, in presence of obstacles. The collision free reference trajectory is generated by manually driving around the parking area at 10kph. MiL with Amesim then HiL with the virtual environment are first validated for this scenario. In the first VeHiL application, it was found that autonomous driving is uncomfortable and performance is undesirable with jerky throttle and aggressive steering. This is mainly due to the stalling engine torque at low speeds and the layout of the parking area that included an upward slope which was not accounted for in the NMPC dynamics. Figure 7 shows real-time visualization of the physical Ego car, generated by colleagues at the ADAS group in Siemens, representing the

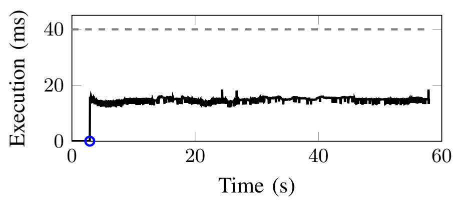

Fig. 8: Parking VeHiL: NMPC execution time (solid) and sample time (dashed)

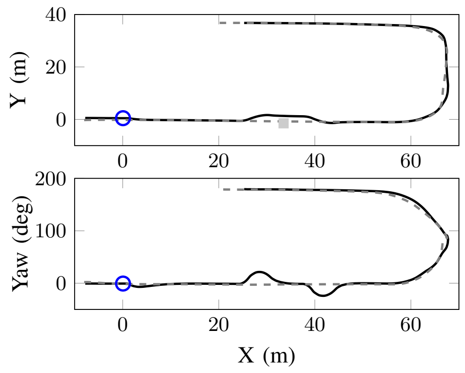

Fig. 9: Parking VeHiL: X-Y and X-Yaw states (vehicle states: solid, reference trajectory: dashed, Obstacle: box)

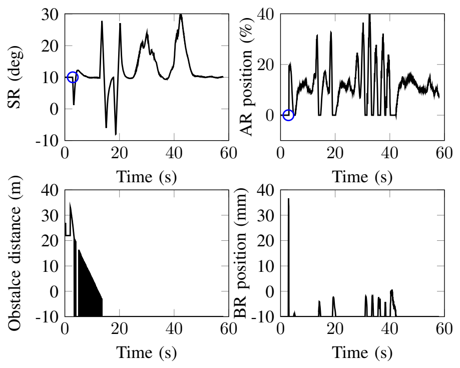

Fig. 10: Parking VeHiL: obstacle and NMPC commands

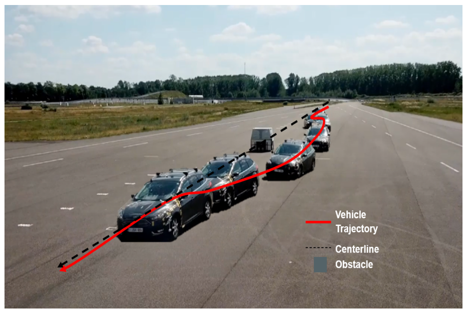

Fig. 11: Proving ground VeHiL: obstacle avoidance

real testing parking area. It happens in real-time as dSPACE MABX-III receives the vehicle coordinates from IMU and GPS and communicates them to Prescan. Ego vehicle can be seen in grey avoiding a virtual construction person.

Figure 9 shows the autonomously driven path after the controller tuning campaign for this AD scenario. NMPC commands the robot as the $X = 0m$ line is crossed (blue circle). Those plots demonstrate the NMPC capabilities in both accurate tracking and collision avoidance. The planner shifts the reference laterally from the original one for a lane change at 1.5m. NMPC reacts quickly to an obstacle only detected within 10m of distance. Finally, Figure 10 shows the obstacle detected range and the NMPC commands in steering robot angle (SR), throttle pedal position (AR) and brake (BR). The obstacle information is not fed to the NMPC before a distance to collision of 10 meters, to simulate emergency obstacle avoidance. The unmsooth throttle behavior between 30 and 40s is due to the upward slope causing a vehicle deceleration. This can be tackled by tuning, more accurate model around the operating speed, or by online parameter adaptation. Nevertheless, the task was still accomplished and with real-time performance as the execution time of one NMPC solve step averages at 14ms for a horizon of 1.2seconds with a sample time of 40ms as shown in Figure 8.

B. *VeHiL: Proving ground validation*

The second part of the testing campaign took place in Germany on secure testing site as shown in the shots of Figure 11 and consisted of an ACC for lane keeping at 60kph over 500 meters with collision avoidance using a dummy vehicle. The longitudinal safe duration parameter, presented in Figure 3 was changed online from 1.8 to 1.2 seconds of reaction time (20 meters of safe distance before and after the obstacle at 60kph). This was done in order to test the NMPC at its boundaries of sudden obstacle emergence and to show the benefits of deploying such a controller in automotive applications as the duration is insufficient for the human driver to take control of the vehicle and avoid an accident.

Results of this testing phase at 60kph, shown in Figures 12 through 14, prove the NMPC quickly reacted, while satisfying all dynamic and actuator constraints. The controller smoothly corrects the initial positioning error off the centerline and performs a cruise control until obstacle detection. The car immediately recovers the original lane after avoiding the obstacle given the small longitudinal distance parameter. NMPC's execution time averaged at 25ms, for a sampling time of 40ms, hence running in real-time on the embedded platform as in Figure 12. The velocity tracking error is similar to the first phase, at almost 0.5m/s as in Figure 13 and is mainly due to longitudinal model mismatches. As from Figures 10 and 14, the jerky throttle control could be a result of the robot delay and the step change in the spatial reference causing some constraints to become active.

Numerical convergence was typically achieved within 1 or 2 SQP iterations and between 1 and 3 QP iterations, for a total of 22ms of execution time per iteration on average for the 60kph scenario and at 15ms for the 10kph. Warm starting the primal variables, taking into consideration the driving corridor, helped in significantly cutting down execution time. The SQP solver used in this project sce-

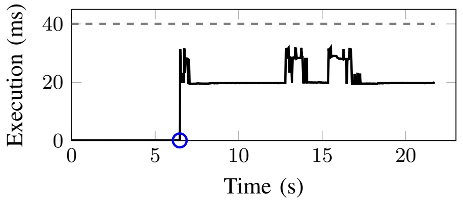

Fig. 12: Proving ground VeHiL: NMPC execution time (solid) and sample time (dashed)

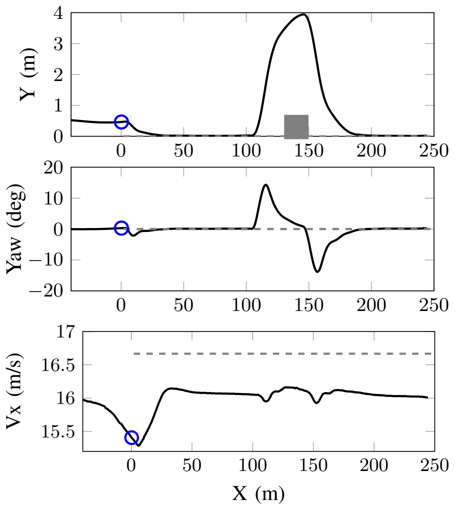

Fig. 13: Proving ground VeHiL: vehicle X-Y, X-Yaw, longitudinal speed (vehicle states: solid, reference trajectory: dashed, NMPC activation: blue, Obstacle: box)

narios proved to be efficient and satisfactory in real-time optimal control, handling non-linearities and converging to the optimal solution. The scenarios were carried out in a MiL framework and resulted in similar control policies as in VeHiL, creating a potential real to simulation flow for VeHiL scenario recreation with additional virtual sensors/obstacles. This shows the developed framework importance as most of the tuning, OCP reformulation and real-time capabilities were tested with the high fidelity model, with little costs, zero incidents and later safe vehicle integration.

C. Problems and possible improvements

- The framework allows for online manual parameter tuning, however, it would be beneficial to include an auto-tuner to facilitate performance matching
- Code generation is beneficial for the implementation of rapid prototyping such as in this project, nevertheless, it often results in very large source codes that are hard to debug rendering the detailed function profiling more complex and one could just avoid code generation

V. Conclusion

This paper presents a development framework for designing, validating and implementing a real-time optimal controller for autonomous driving applications. The validation process satisfies the automotive industry requirements as it goes from ISO-standards, scenario definition, to the different XiL applications, with high-fidelity models. The framework allows testing in real and/or virtual environments using Simcenter software. NMPC is used as low-level controller in the applications running in real-time at 25Hz, showing the potential capabilities of this controller type for collision and accident avoidance, ACC and trajectory following. Embedded numerical optimization is deployed on the platform and can be extended to more complex OCP or other optimization based formulations such as optimal planning. The framework was tested and validated on a Ford Focus in parking area and on proving grounds, in presence of obstacle and with a strict requirement on real-time calculations.

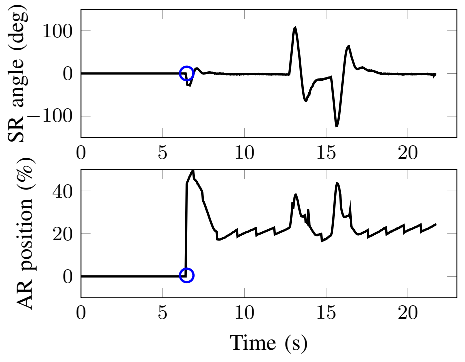

Fig. 14: Proving ground VeHiL: NMPC control actions

References

[1] A.-L. Do and F. Fauvel, “LPV approach for collision avoidance: Controller design and experiments,” *Control Engineering Practice*, vol. 113, p. 104856, 2021.

[2] T. D. Son and Q. Nguyen, “Safety-critical control for non-affine nonlinear systems with application on autonomous vehicle,” in *58th IEEE Conference on Decision and Control, Nice*, 2019.

[3] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, “Learning-based model predictive control for autonomous racing,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[4] H. Ferreau, S. Almer, R. Verschueren, M. Diehl, D. Frick, A. Domahidi, J. Jerez, G. Stathopoulos, and C. Jones, “Embedded optimization methods for industrial automatic control,” in *20th IFAC World Congress*, vol. 50, pp. 13194–13209, 2017.

[5] A. Gambier, “Real-time control systems: A tutorial,” *IEEE Explore*, 2004.

[6] T. D. Son, J. Hubrechts, L. Awatsu, A. Bhave, and H. V. der Auweraer, “A simulation-based testing and validation framework for ADAS development,” in *Transport Research Arena*, 2017.

[7] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, second ed., 2006.

[8] J. Gillis, “Nonlinear programming and code-generation in casadi,” 2019.

[9] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi – A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[10] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, “acados: a modular open-source framework for fast embedded optimal control,” 2020.

[11] P. Listov and C. Jones, “PolyMPC: An efficient and extensible tool for real-time nonlinear model predictive tracking and path following for fast mechatronic systems,” 2020.

[12] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.