Implementation and Analysis of Nonlinear Model Predictive Controller on Embedded Systems for Real-Time Applications

Saket Adhau, Sayli Patil, Deepak Ingole, Dayaram Sonawane

▶ **To cite this version:**

Saket Adhau, Sayli Patil, Deepak Ingole, Dayaram Sonawane. Implementation and Analysis of Nonlinear Model Predictive Controller on Embedded Systems for Real-Time Applications. ECC 2019, 18th European Control Conference, Jun 2019, Naples, Italy. pp3359-3364, 10.23919/ECC.2019.8796118 . hal-03586196

HAL Id: hal-03586196
https://hal.science/hal-03586196v1

Submitted on 23 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Implementation and Analysis of Nonlinear Model Predictive Controller on Embedded Systems for Real-Time Applications

Saket Adhau[1], Sayli Patil[1], Deepak Ingole[2], and Dayaram Sonawane[1]

***Abstract*— Today, various nonlinear programming problem (NLP) solvers and C/C++ code generation frameworks are available as open source for solving nonlinear model predictive control (NMPC). Almost all the solvers are written in C/C++ code which are more compatible for the PC-based simulation environment. These codes are not directly compatible for embedded implementation and real-time control. An attempt has been made to address this shortcoming by creating a customized framework on top of the C code generated from `ACADO Toolkit` to make it directly compatible with ARM based embedded platforms. The study also analyzes the embedded implementation aspects using C code generated for qpOASES, qpDUNES, and HPMPC solvers from `ACADO Toolkit`. We show the results of hardware-in-loop (HIL) simulations with detailed analysis and comparison of memory requirement and achievable sampling time for three benchmark dynamical systems on different embedded platforms viz ARM Cortex M3, PYNQ FPGA and Raspberry Pi. The results show that qpOASES outperforms as compared to the other two solvers when the computational time is of prime importance for small prediction horizon. Similarly, when there are limited on-chip memory resources, qpDUNES can prove beneficial.**

***Index Terms*— Nonlinear model predictive control, optimization solvers, embedded systems.**

I. INTRODUCTION

Model predictive control (MPC) has been consistently used in both, industries and academic research [1]. MPC has an inherent potential to handle multi-objective/multi-variable problems, dealing explicitly with hard constraints as well as economic and safety considerations. The prevalent use of linear model predictive control in control tasks has been well established and the nonlinear model predictive control (NMPC) looks like an intriguing solution for highly nonlinear control tasks. NMPC captures all the natural nonlinear dynamics and constraints of the model and outperforms as compared to linear MPC. This potential can only be fully utilized when all the nonlinear process dynamics and constraints are explicitly defined in the controller [2], [3], [4].

The blooming development in the optimization area has made room for many nonlinear solvers and C/C++ code generation tools. These tools are highly efficient, reliable and feasible for real-time online optimization with sampling time in milliseconds and microseconds range [5].

However, computationally challenging applications like autonomous vehicles, unmanned aerial vehicles, fast-changing dynamic systems, etc., require real-time implementations. The code generation tools such as Casadi [6], IPOPT [7], PANOC NMPC [8], and AutoGenU for Mapple in [9] are excellent for PC-based environment but not as suitable and ready to port for embedded implementation. Tools such as FORCES PRO [10] and ODYS [11] are commercially available for embedded implementation of NMPC.

On the other hand, `ACADO Toolkit` [12], is a generalized framework for various automatic control and dynamic optimization along-with built in NMPC. This is a self-contained object-oriented C/C++ code, which comes with the MATLAB user interface and is available as open source under GNU lesser general public license[1]. The toolkit generates highly efficient C/C++ code but it is generally library dependent. Several authors have validated the reliability of the `ACADO Toolkit` C/C++ code in PC-based simulations with sampling time in microsecond range [13]. The previous works [14], [15], [16] shows the implementation of NMPC with computational time results obtained from PC-based simulations. The author [17] implemented C/C++ code from `ACADO Toolkit` on RaspberryPi which is an OS-based embedded platform.

Features like high-performance architecture, flexible peripheral connectivity, analog support, and low power consumption makes ARM based processor a suitable choice in various industrial applications. ARM based controllers are restricted by the available memory and clock frequency. FPGA is one of the possible solutions for high memory and speed demanding applications. Due to features like parallelism and optimal performance, FPGA is proven to be the most probable choice for applications like image processing, telecommunication, military & aerospace, medical electronics, automotive etc.

In this paper, NMPC for real-time control is implemented on Atmel ARM Cortex M3 micro-controller which has limited resources in terms of clock and memory. `ACADO Toolkit` generates a s-function which is compatible with MATLAB simulations but not with embedded devices. For HIL co-simulations on Atmel ARM, we have modified the s-function so as it can be easily ported on embedded devices. Similarly, we have validated the results on RaspberryPi and Xilinx PYNQ FPGA for models with large memory requirements. Extending the work, we have also performed detailed analysis of memory utilization and sampling time for different solvers available in `ACADO Toolkit` for different problem size.

[1] Department of Instrumentation and Control Engineering, College of Engineering Pune, Shivajinagar 411005, India {adhauss17.instru, patilsd17.instru, dns.instru}@coep.ac.in

[2] University of Lyon, IFSTTAR, ENTPE, Lyon 69120, France deepak.ingole@ifsttar.fr

[1] http://acado.github.io/

II. Nonlinear Model Predictive Control

MPC as a constrained finite-time optimal control (CFTOC) problem for reference tracking is represented as,

$$\min_{x(.),u(.)} \int_{t_0}^{t_0+T} \frac{1}{2}\left(||x(t)-x^{\text{ref}}(t)||_Q^2 + ||u(t)-u^{\text{ref}}(t)||_R^2\right) dt + ||x(t_0+T)-x^{\text{ref}}(t_0+T)||_P^2, \tag{1a}$$

subjected to following constraints for all $t \in [t_0, t_0+T]$

$$x(t_0) = \hat{x}_0, \tag{1b}$$
$$\dot{x}(t) = f(x(t), u(t)), \tag{1c}$$
$$u_{\min} \leq u(t) \leq u_{\max}, \tag{1d}$$
$$x_{\min} \leq x(t) \leq x_{\max}. \tag{1e}$$

with initial condition $x(t_0) \in \mathbb{R}^n$. The term (1a), is the cost function and $T > 0$ is the prediction horizon. The cost function is weighted by $Q \succeq 0, P \succeq 0$, and $R \succ 0$. The function f in (1c) describes the nonlinear dynamics of the system in ordinary differential equations (ODE) form which depends on the input signal $u(t) \in \mathbb{R}^m$. The $\hat{x}_0$ is the current state estimate in (1b) and t_0 is the current time. The input is bounded by $u_{\min}$ and $u_{\max}$. The time varying states and control references are x^{ref} and u^{ref}, respectively, see [18], [19, Chapter 2].

III. ACADO for NMPC Problem Formulation and Code Generation

This section briefly describes the low level C/C++ based numerical methods employed to solve the NLP at each real-time iteration.

The `ACADO Toolkit` includes a variety of algorithms for dynamic optimization, state and parameter estimation, and automatic control. The toolkit exports highly efficient C/C++ code for NMPC. There are numerous user accessible settings for defining the NMPC problem and achieving the desired performance. The `ACADO Toolkit`-MATLAB interface is used to generate customized C code which can be readily ported onto the embedded devices. Some of the basic steps while problem formulation and code generation are discussed below:

A. Real-Time Iteration Scheme

Nonlinear model predictive control algorithm is computationally expensive task because it needs to solve CFTOC problem (1a) at each sampling instant. In recent years, many algorithms have been proposed which reduce computational complexity [13], [18], [20]. Real-time iteration (RTI) scheme is one of the most efficient approaches among the available algorithms. RTI scheme for fast NMPC implementation was first introduced [21]. For a quick feedback of the NMPC to the process, RTI executes merely one sequential quadratic programming (SQP) iteration per time step.

RTI divides the computation task into two phases, preparation and feedback phase. Preparation phase includes linearization and generation of feedback approximation function and is the most expensive step. Feedback phase rapidly solves the sub-problem to obtain an approximate solution. The time required to feed control inputs to the real process depends on how fast quadratic problem (QP) is solved.

B. Restrictions to use RTI Scheme

While using the RTI scheme in real-time applications, some factors might lead to failure of the algorithm [13]:

1) Infeasibility: The cost function in (1a), subjected to state and inequality constraints is vulnerable to two types of infeasibilities. The nonlinear optimization problem might itself be infeasible or the underlying SQP-type algorithm can become infeasible.

2) Instability: Assuming the QP is feasible for all problems, the closed-loop system might still become unstable. As we have finite prediction horizon, closed-loop system is susceptible to instability. Adding weights on the cost function and/or zero terminal constraints removes this instability.

C. Nonlinear Model and OCP Formulation

Using the MATLAB interface, nonlinear dynamics of the model were added in the form of f. All the case studies have been studied with prediction horizon, $T = 5$ and same control horizon.

D. Solvers Used

Solvers require to solve the parameterized dynamic optimization problems simultaneously for obtaining an optimal control law at each sampling time. Convex QP solvers are used in `ACADO Toolkit` for solving optimal control problems. In this study, solvers have been selected in consideration with the realization of embedded NMPC to run in a real-time scenario. The following factors were the main criteria for selection of the solvers.

- Performance and numerical stability considering the safety aspects so as to run with limited resources and the ability to use single precision arithmetic for the microcontroller,
- Structure of the C-code, minimum library dependence, and code portability,
- Ability to terminate the solver early in order to obtain a feasible but suboptimal solution for the real-time applications,
- Ability to hotstart the QP for fast receding horizon control.

The listed embedded solvers are used in this study and are also supported by `ACADO Toolkit` code generation. The solver option can be set by using the command `mpc.set('QP_SOLVER')`.

- qpOASES: This is an open source C++-based implementation of online active set strategy proposed in [22] for solving QP problems.
- qpDUNES: This is a plain C-code compiling to C90 standards making it feasible to port the code on various embedded platforms, see [23].
- HPMPC: Defined as the high-performance implementation of solvers for NMPC, currently interior-point method (IPM) and alternating direction method of multipliers (ADMM) solvers are supported in this with the

ability to select the target architecture in configuration file [24].

E. Code Export

As discussed above, `ACADO Toolkit` can generate highly efficient C-code for solving nonlinear MPC using Real-time iteration (RTI) and Gauss-Newton Hessian approximation. For code export, `mpc.exportCode('export_NMPC')` is used. The generated code is exported in `export_NMPC` folder consisting of solver, integrator code and a test file. `MAKEFILE` is also provided to facilitate the compilation process. `ACADO Toolkit` also gives liberty to set compiler flag options for target specific code generation. Also, the solvers allow to set their own compiler flags to cater the user needs. In all the cases we use single precision (32-bit), which is done using `'USE SINGLE PRECISION'`. Printing of all the NLP solver information is disabled by `'PRINTLEVEL','NONE'`.

IV. Embedded Implementation of NMPC

In the following sections we describe the NMPC working scheme, used embedded hardware, and case studies which have been implemented and the methodology to implement on embedded hardware.

A. NMPC Framework

The Fig. 1 shows the basic closed-loop control strategy for obtaining the optimal control value by using nonlinear optimization solver or NLP [25].

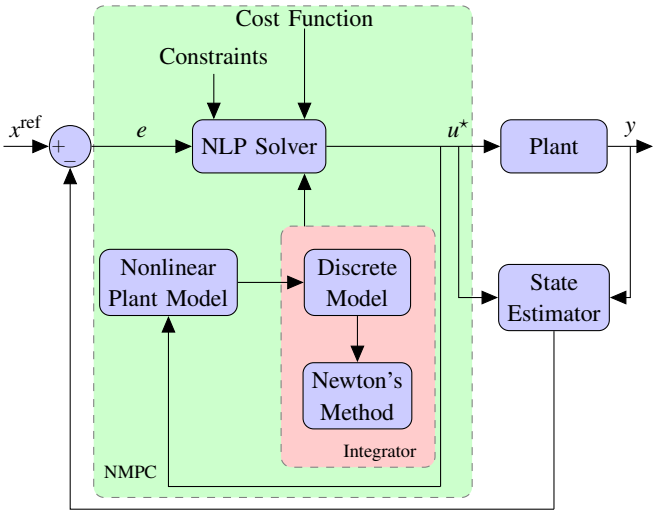

Fig. 1. Closed-loop NMPC scheme.

B. Embedded Hardware Used

We have used three different hardware for NMPC implementation and HIL simulation purpose. Each device has it's own significance for its selection and usage. We have experimented from basic 32-bit ARM microcontroller to high functioning PYNQ FPGA board. The study aims to implement NMPC in real-time on true embedded devices as otherwise shown in previous works on PC [16].

1) Atmel ARM Cortex-M3: Atmel's ARM Cortex-M3 is a true 32-bit microcontroller running at 84 MHz is a preferred microcontroller in embedded automotive applications. It has 512 kB of program memory and 96 kB of SRAM. Fig. 2 illustrates HIL implementation flow on ARM using S-Function in MATLAB and Simulink.

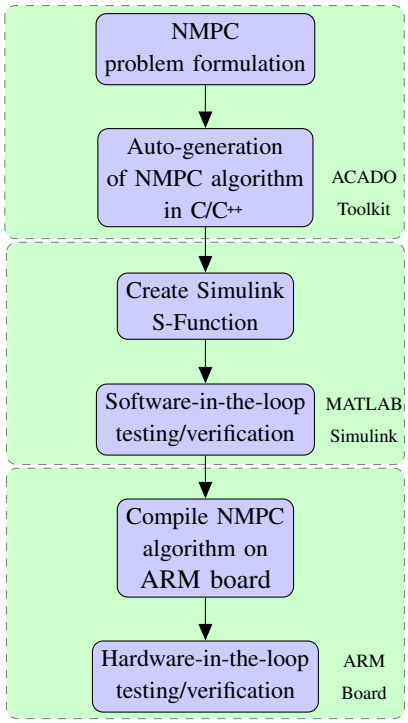

Fig. 2. HIL design flow of NMPC algorithm on ARM.

2) Xilinx PYNQ FPGA: PYNQ is abbreviated as Python productivity for ZYNQ. Talking about the hardware, the core of PYNQ is Xilinx ZYNQ system on chip (SOC) XC7Z020 – 1CLG400C with dual-core ARM Cortex-A9 processor running at 450 MHz and 630 kB. The Artix-7 family programmable logic contains 13300 logic slices, each with four 6-input lookup tables (LUTs) and 8 flip-flops (FFs). In addition, it also provides 220 DSP blocks and 630 kB of fast block RAM [26]. We are using the dual-core ARM on this device.

3) RaspberryPi: RaspberryPi is SoC featuring 64-bit quad-core ARM Cortex-A7 processor which runs at 1.2 GHz with 1 GB RAM. This is mostly preferred by hobby enthusiasts and is easy to use.

C. Case Study I: Position Control of Hovercraft

As the case studies, we have considered three nonlinear models having different number of states and control inputs. One of the models is position control of hovercraft which is a six state model having two control inputs with bounds on input. We applied NMPC to this model for state tracking and obtained closed-loop HIL simulation results on PYNQ board. The mathematical model and the values of physical parameter of this model are taken from [27]. The hovercraft

has two propellers which are fed with DC Motor. The center of mass is denoted by (x,y) and θ denotes the attitude angle of the hovercraft as shown in Fig. 3. The generated thrust by the propellers are u_1 and u_2, r is the length between the center line of the hovercraft and the thrusters, M is the mass while I is the inertia of the model.

The differential state equation $\frac{dx}{dt} = \dot{x} = f(x,u)$, is obtained from equations of motion in x direction, $\frac{dy}{dt} = \dot{y} = f(y,u)$ is the equations of motion in y direction and $\frac{d\theta}{dt} = \dot{\theta} = f(\theta,u)$ denotes as attitude angle. Neglecting the aerodynamic forces for simplicity, the continuous time equations governing the nonlinear dynamics of the model having the states $X = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]^T$ and input vectors $U = [u_1, u_2]^T$ are given as follows:

$$\frac{d\dot{x}}{dt} = \frac{cos\theta(u1+u2)}{M}, \tag{2a}$$

$$\frac{d\dot{y}}{dt} = \frac{sin\theta(u1+u2)}{M}, \tag{2b}$$

$$\frac{d\dot{\theta}}{dt} = \frac{(u1-u2)r}{I}. \tag{2c}$$

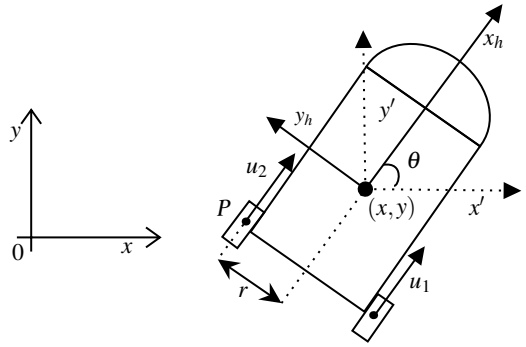

Fig. 3. A schematic of the hovercraft.

TABLE I
VALUES OF PARAMETERS USED IN HOVERCRAFT MODEL.

Parameter	Value	Unit
Mass (M)	0.974	kg
Inertia (I)	0.0125	kg m^2
Length (r)	0.0485	m

The objective function for the hovercraft model is formulated as follows:

$$\min_{x(.),u(.)} \int_{t_0}^{t_0+T} \frac{1}{2} \left(||x(t)||_Q^2 + ||u(t)||_R^2 \right) dt + ||x(t_0+T)||_P^2 \tag{3a}$$

subject to,

$$-0.121 \leq u_1(t) \leq 0.342, \tag{3b}$$

$$-0.121 \leq u_2(t) \leq 0.342, \tag{3c}$$

where $x^{\text{ref}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

D. Case Study II and III: Motor and Quadrotor Model

For further analysis, we have chosen two more models for implementation and comparison purpose.

1) Motor Model: Speed control of the DC motor has been done using NMPC. The nonlinear DC motor model has been taken from [28] [29, Chapter 2]. The DC motor model is of two states with bounds on input and state variables.

2) Quadrotor Model: Attitude and position control of the quadrotor has been done using NMPC. The nonlinear quadrotor model is taken from [30], [31]. The model is highly nonlinear consisting of nine states and four control input with bounds on inputs and state variables.

V. NMPC PERFORMANCE COMPARISON

NMPC closed-loop HIL simulation have been performed in three case studies. The results of case studies are analyzed using different solvers on various embedded platforms. It is significant to note that in the literature, the performance of controllers on considered case studies have been presented for PC based simulation environment. But, in this work we are presenting the results of NMPC implemented on embedded systems. The work intends to show effect of the change in number of states and the resource utilization of embedded optimization solvers.

NMPC closed-loop simulation performed on PYNQ as shown in Fig. 5. The figure shows the simulation results for the hovercraft model with horizon length, $T = 5$ and sampling time, $Ts = 500$ms and the initial conditions as $x_0 = \begin{bmatrix} -0.25 & 0.35 & 0 & 0 & 0 & 0 \end{bmatrix}$. Considering the above settings optimization problem was solved using qpDUNES embedded optimization solver. We observed minimum and maximum RTI execution time to be 902 µs and 940 µs, respectively with the size of executable file as 97 kB. This shows real-time implementation of NMPC scheme is feasible on embedded devices for varying problem size.

Table. II and III, shows the detailed analysis of computational time and memory requirements NMPC implementation. It can be clearly seen that, increasing the problem size results in increased memory requirements and computational time. Due to memory restrictions, large problem size could not be implemented on Atmel ARM where it throws the error of memory overflow and due to that we are not able to record the RTI time. It can also be inferred, computational time is typically low for condensing solver (qpOASES) compared to the sparse solver such as qpDUNES and HPMPC.

However when the prediction horizon increases, qpDUNES and HPMPC perform better than qpOASES as stated earlier. Also, code memory does not seem to change noticeably with increase in prediction horizon. These results motivates to select appropriate solver and embedded platform to meet the requirements. For result validation, we analyzed NMPC performance for randomly generated NLP problems and implemented them on PYNQ FPGA. Every NLP problem was built with a random model while increasing the state variables which resulted in typical sizes for the intended NMPC applications. The generated NMPC were tested on above NLP solvers. Fig. 4 illustrates the results of variation in

TABLE II

NMPC C/C++ PROGRAM MEMORY REQUIREMENT IN (kB) WITH T = 5.

Embedded Platforms	Atmel ARM			PYNQ FPGA			Raspberry Pi		
Solver	No. of States								
	2	6	9	2	6	9	2	6	9
qpOASES	352	Memory overflow		146	163	180	164	184	198
qpDUNES	298			81	97	115	100	120	140
HPMPC	389			252	277	300	300	332	284

TABLE III

NMPC RTI AVERAGE TIME IN (µs) WITH T = 5.

Embedded Platforms	Atmel ARM			PYNQ FPGA			Raspberry Pi		
Solver	No. of States								
	2	6	9	2	6	9	2	6	9
qpOASES	242	Not available		139	721	2770	150	769	3330
qpDUNES	305			233	928	3090	237	1190	3580
HPMPC	350			189	789	2860	255	832	3370

TABLE IV

VARIATION IN AVERAGE RTI TIME IN (µs) WITH CHANGE IN PREDICTION HORIZON(T) FOR HOVERCRAFT MODEL (2)

Embedded Platforms	PYNQ FPGA			Raspberry Pi		
Solver	Prediction Horizon (T)					
	3	5	10	3	5	10
qpOASES	473	721	1545	530	769	1670
qpDUNES	542	928	1152	680	1190	1229
HPMPC	559	789	1356	702	832	1310

NMPC memory requirements with an increase in the number of states.

VI. CONCLUSIONS

The work shows the real-time implementation of NMPC on embedded systems and its performance evaluation using `ACADO Toolkit` code generation. The work shows detailed investigation of various NLP solvers implemented for different problem size in terms of memory and time profiling for one real-time iteration of NMPC. We also presented the closed-loop simulation results for the hovercraft model on PYNQ FPGA board. The work aims at contributing towards feasible real-time implementation of NMPC in complex control tasks and proves to be a promising solution. Future work will attempt to write a customized wrappers on top of generated C/C++ codes of NMPC to improve its memory requirements and sampling time for real-time control applications.

ACKNOWLEDGMENT

We gratefully acknowledge the support from R & D center of the COEP. Deepak Ingole would like to thank for a financial contribution from the ERC under the European Unions Horizon 2020 research and innovation program (grant agreement no. 646592 MAGnUM project).

REFERENCES

[1] X. Yang, D. W. Griffith, and L. T. Biegler, "Nonlinear programming properties for stable and robust NMPC," *IFAC-PapersOnLine*, pp. 388–397, 2015.

[2] D. Kouzoupis, R. Quirynen, J. Frasch, and M. Diehl, "Block condensing for fast nonlinear MPC with the dual newton strategy," *IFAC-PapersOnLine*, pp. 26 – 31, 2015.

[3] F. Xu, H. Chen, W. Jin, and Y. Xu, "FPGA implementation of nonlinear model predictive control," in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, 2014, pp. 108–113.

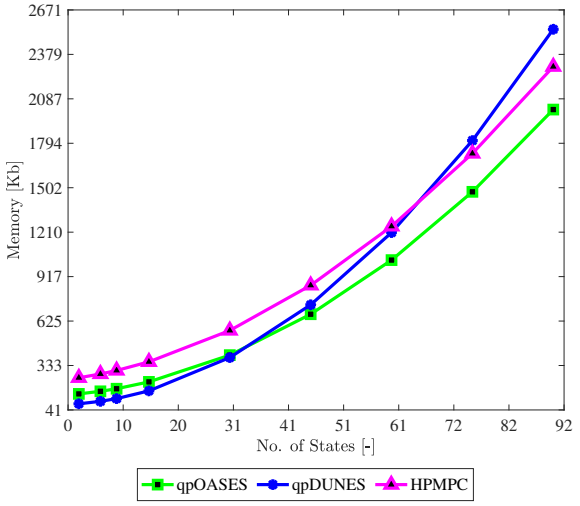

Fig. 4. Effect of change in problem size for various solvers.

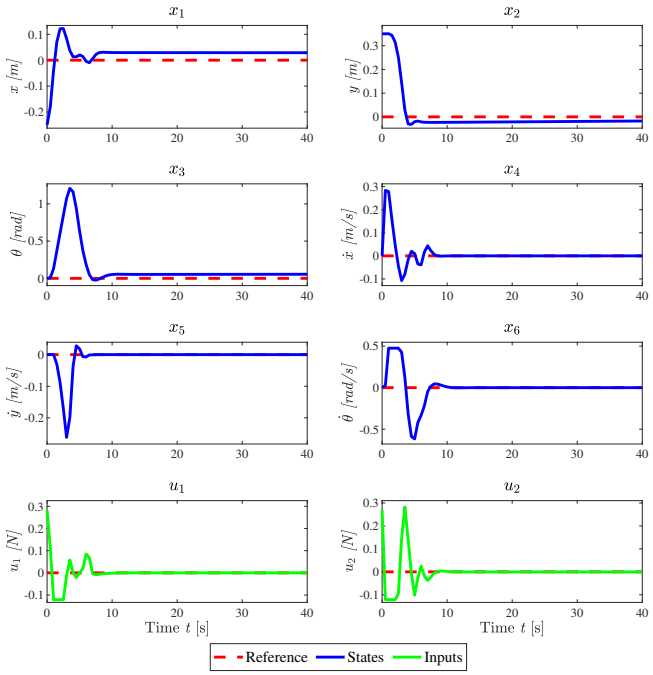

Fig. 5. HIL simulation results of NMPC implemented on PYNQ FPGA board.

[4] R. Findeisen and F. Allgöwer, "An introduction to nonlinear model predictive control," in *21st Benelux Meeting On Systems And Control, Veidhoven*, 2002, pp. 1–23.

[5] J. Mattingley and S. Boyd, *CVXGEN: A code generator for embedded convex optimization*, 1st ed. Springer, 2012.

[6] 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*, 2018.

[7] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, pp. 25–57, 2006.

[8] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A simple and efficient algorithm for nonlinear model predictive control," in *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, 2017, pp. 1939–1944.

[9] T. Ohtsuka, "A tutorial on C/GMRES and automatic code generation for nonlinear model predictive control," in *2015 European Control Conference (ECC)*, 2015, pp. 73–86.

[10] A. Domahidi and J. Perez, "FORCES PRO ACADEMIC," 2014.

[11] G. Cimini, A. Bemporad, and D. Bernardini, "ODYS QP Solver," ODYS (https://odys.it/qp), 2017.

[12] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit - An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, pp. 298–312, 2011.

[13] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, pp. 2279 – 2285, 2011.

[14] N. Duijkeren, T. Keviczky, P. Nilsson, and L. Laine, "Real-time nmpc for semi-automated highway driving of long heavy vehicle combinations," *5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015*, pp. 39 – 46, 2015.

[15] R. Verschueren, S. D. Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, "Towards time-optimal race car driving using nonlinear MPC in real-time," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 2505–2510.

[16] M. Vukov, W. V. Loock, B. Houska, H. Ferreau, J. Swevers, and M. Diehl, "Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation," in *The 2012 American Control Conference, Montreal, Canada.*, 2012.

[17] R. Quirynen, K. Berntorp, and S. Di Cairano, "Embedded optimization algorithms for steering in autonomous vehicles based on nonlinear model predictive control," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 3251–3256.

[18] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated algorithms for nonlinear model predictive control on long and on short horizons," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 5113–5118.

[19] A. Grancharova and T. A. Johansen, *Explicit nonlinear model predictive control: Theory and applications*, 1st ed., ser. Lecture Notes in Control and Information Sciences 429. Springer-Verlag Berlin Heidelberg, 2012.

[20] B. Khusainov, E. C. Kerrigan, A. Suardi, and G. A. Constantinides, "Nonlinear predictive control on a heterogeneous computing platform," in *IFAC-PapersOnLine*, 2017, pp. 11 877 – 11 882.

[21] M. Diehl, H. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," in *Journal of Process Control*, 2002, pp. 0959–1524.

[22] H. Ferreau, H. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, pp. 816–830, 2008.

[23] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Mathematical Programming Computation*, 2013.

[24] G. Frison, H. H. B. S⊘rensen, B. Dammann, and J. B. J⊘rgensen, "High-performance small-scale solvers for linear model predictive control," in *2014 European Control Conference (ECC)*, 2014, pp. 128–133.

[25] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," in *Nonlinear model predictive control*. Springer, 2009, pp. 391–417.

[26] "PYNQ," http://www.pynq.io/home.html.

[27] Y. Shimizu, T. Ohtsuka, and M. Diehl, "Nonlinear receding horizon control of an underactuated hovercraft with a multiple-shooting-based algorithm," *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pp. 603–607, 2006.

[28] V. Sankardoss and P. Geethanjali, "Parameter estimation and speed control of a PMDC motor used in wheelchair," *Energy Procedia*, vol. 117, pp. 345–352, 2017.

[29] R. Krishnan, *Electric motor drives: modeling, analysis, and control*. Prentice Hall, 2001.

[30] M. Hehn and R. D'Andrea, "A flying inverted pendulum," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 763–770.

[31] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," in *IFAC Proceedings Volumes*, 2011, pp. 1485 – 1491.

View publication stats