Implementation of a Long-Horizon Model Predictive Control Algorithm on an Embedded System

Eyke Liegmann*, Petros Karamanakos†, Ralph Kennel*
*Chair of Electrical Drive Systems and Power Electronics, Technical University of Munich
Arcisstr. 21, 80333 Munich, Germany
†Faculty of Information Technology and Communication Sciences, Tampere University
FI-33101 Tampere, Finland
Email: eyke.liegmann@tum.de, p.karamanakos@ieee.org, ralph.kennel@tum.de

Acknowledgments

Funded by Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Education and Research)—Project KI-Power.

Keywords

≪Model Predictive Control≫, ≪Multilevel converters≫, ≪Field Programmable Gate Array (FPGA)≫, ≪Variable speed drive≫.

Abstract

This paper deals with the implementation of a long-horizon finite control set model predictive control (FCS-MPC) algorithm on an embedded system. The targeted application is a medium voltage drive system implying a very low switching frequency. The implementation is facilitated by the use of a high level synthesis (HLS) tool, which synthesizes C++ code into VHDL, enabling a higher level of abstraction and faster prototype development. Experimental results based on a small-scale prototype, consisting of a three-level neutral point clamped (NPC) inverter and an induction machine, confirm that the algorithm can be executed in real time within the targeted control period of $25\,\mu s$. This allows for high switching granularity, and thus favorable steady-state and transient performance.

Introduction

In the past decade, model predictive control (MPC) [1] has gained considerable attention in the field of power electronics [2], [3]. As the name suggests, MPC uses a model of the plant to predict its dynamics. Based on this prediction, the optimal control sequence is chosen on the basis of a cost function that resembles the desired system behavior. Employing MPC with long prediction horizons significantly reduces the current distortion and notable improvements under steady-state operating conditions can be achieved [4]. However, practical realization of real-time MPC with long prediction horizons is not trivial. This is due to the significant computational challenges arising from the high number of candidate switching sequences since the latter grows exponentially with the number of the horizon steps [5].

To mitigate the computational cost of long-horizon finite control set MPC (FCS-MPC) a dedicated branch-and-bound method—named sphere decoder—tailored to the needs of power electronic systems was discussed in [6]. In previous publications, the sphere decoder was implemented on dSPACE systems [7]–[9], with the limitations that either the sampling interval is long ($>100\,\mu s$) [7], [8], which reduces the switching granularity [10], or a two-level inverter is considered [9], implying that the number of the candidate solutions, and thus the complexity of the control problem, are relatively small. In contrast to these works, implementations of sphere decoder on a field-programmable gate array (FPGA) were presented in [11], [12]. The tailored VHDL implementation in [11] allowed for a sampling interval of $25\,\mu s$ for a three-level neutral point clamped (NPC) inverter with an RL load, whereas an induction

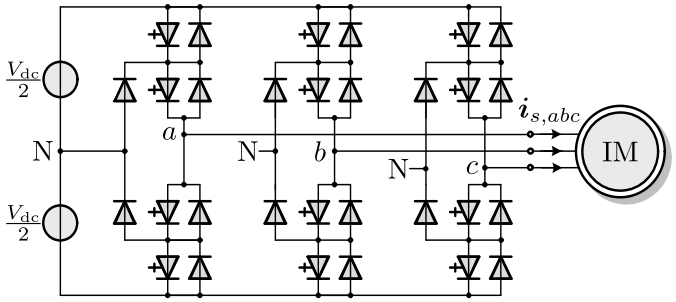


Fig. 1: Three-level three-phase NPC voltage source inverter driving an induction motor.

machine (IM) is used in an FPGA-in-the-loop simulation in [12]. The former, however, requires significant effort in the implementation stage, making its adaptation to other problems a fairly laborious task. As for the latter, it was not verified experimentally, meaning that second-order effects of a real-world setup, are neglected in the performance verification.

Motivated by the above, this paper presents an efficient real-time implementation of the sphere decoding algorithm, employed to solve the long-horizon FCS-MPC problem for a three-level NPC inverter driving an IM, as shown in Fig. 1. To this aim, optimization techniques of the HLS tool from Xilinx are utilized, which facilitate a rapid implementation on the system-on-chip (SoC), combining an FPGA and several processors in one silicon chip. By adopting this alternative strategy the targeted control frequency of $40\,\mathrm{kHz}$ is well achieved during both steady-state and transient, as verified by the presented experimental results acquired on a small-scale prototype. Hence, as demonstrated in this paper, the proposed implementation allows for a high granularity of switching, which, in turn, results in favorable system performance.

Control Model

The examined problem relates to the control of the stator current of an IM connected to a three-level NPC inverter, as depicted in Fig. 1, for which the mathematical model is derived in the sequel. Using the stator current $\boldsymbol{i}_s$ and rotor flux $\boldsymbol{\psi}_r$ in the $\alpha\beta$-plane as state, the dynamics of the IM are described by [13]

$$\frac{\mathrm{d}\boldsymbol{i}_s}{\mathrm{d}t} = -\frac{1}{\tau_s}\boldsymbol{i}_s + \left(\frac{1}{\tau_r}\boldsymbol{I} - \omega_r \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right)\frac{X_m}{D}\boldsymbol{\psi_r} + \frac{X_r}{D}\boldsymbol{v}_s \tag{1a}$$

$$\frac{\mathrm{d}\boldsymbol{\psi}_r}{\mathrm{d}t} = \frac{X_m}{\tau_r}\boldsymbol{i}_s - \frac{1}{\tau_r}\boldsymbol{\psi_r} + \omega_r \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\boldsymbol{\psi_r}\,, \tag{1b}$$

where $X_s = X_{ls} + X_m$, $X_r = X_{lr} + X_m$, and X_m are the stator, rotor and mutual reactances, respectively, with X_{ls} being the stator leakage reactance and X_{lr} that of the rotor. Moreover, $\tau_s = X_r D/(R_s X_r^2 + R_r X_m^2)$ is the transient stator time constant, while $\tau_r = X_r/R_r$ the transient time constant of the rotor winding. Finally, the constant D is defined as $D = X_s X_r - X_m^2$, and $\boldsymbol{I}$ is the identity matrix of appropriate dimensions. Note that the rotor angular speed ω_r is considered to be a time-varying parameter.

To further simplify the subsequent implementation, the dc-link voltage $V_{\rm dc}$ and the neutral point potential are assumed to be fixed. Given that the inverter output can assume three discrete values, namely, $-V_{\rm dc}/2$, 0, and $V_{\rm dc}/2$, depending on the position of its switches, the three-phase switch position is modeled as $\boldsymbol{u}_{abc} = [u_a\ u_b\ u_c]^T$, with $u_a, u_b, u_c \in \mathcal{U} = \{-1, 0, 1\}$ being the single-phase switch positions. Moreover, by choosing the stator current $\boldsymbol{i}_s$ and rotor flux $\boldsymbol{\psi}_r$ in the $\alpha\beta$ coordinate system as state variables, i.e., $\boldsymbol{x} = [i_{s\alpha}\ i_{s\beta}\ \psi_{r\alpha}\ \psi_{r\beta}]^T$, the stator current as the system output $\boldsymbol{y} = [i_{s\alpha}\ i_{s\beta}]^T$, and the three-phase switch position $\boldsymbol{u}_{abc}$ as the system input, the following continuous-time state-space model is derived

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{F}\,\boldsymbol{x}(t) + \boldsymbol{G}\,\boldsymbol{u}_{abc}(t) \tag{2a}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\,\boldsymbol{x}(t)\,, \tag{2b}$$

where the matrices $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{C}$ are given in the appendix.

By using Euler discretization, the discrete-time state-space model of the drive becomes

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}_{abc}(k) \tag{3a}$$

$$\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k) \tag{3b}$$

where $\boldsymbol{A} = \mathbf{e}^{\boldsymbol{F}T_s}$ and $\boldsymbol{B} = \int_0^{T_s} \mathbf{e}^{\boldsymbol{F}\tau}\mathrm{d}\tau\,\boldsymbol{G}$, with $\mathbf{e}$ being the matrix exponential, T_s the sampling interval, and $k \in \mathbb{N}$.

Control Problem Formulation

The control objective is to track the stator current reference $\boldsymbol{i}_s^* = \boldsymbol{y}^*$ while operating the drive at a low switching frequency so that the switching power losses are kept low. These goals are mapped into a scalar via the cost function J, defined as

$$J(k) = \sum_{\ell=k}^{k+N-1} \|\boldsymbol{y}^*(\ell+1) - \boldsymbol{y}(\ell+1)\|_2^2 + \lambda_u \|\Delta\boldsymbol{u}_{abc}(\ell)\|_2^2, \tag{4}$$

where N denotes the prediction horizon steps and the term $\Delta\boldsymbol{u}_{abc}(\ell) = \boldsymbol{u}_{abc}(\ell) - \boldsymbol{u}_{abc}(\ell-1)$ penalizes the switching effort. The weighting factor $\lambda_u > 0$ is introduced to adjust the trade-off between the tracking and control effort, thus enabling the adjustment of the switching frequency. Function (4) needs to be minimized in real time to find the sequence of switch positions

$$\boldsymbol{U}(k) = \begin{bmatrix} \boldsymbol{u}_{abc}^T(k) & \boldsymbol{u}_{abc}^T(k+1) & \dots & \boldsymbol{u}_{abc}^T(k+N-1) \end{bmatrix}^T \in \mathbb{U}, \tag{5}$$

with $\mathbb{U} \in \boldsymbol{\mathcal{U}}^N \subset \mathbb{Z}^{3N}$ and $\boldsymbol{\mathcal{U}} = \mathcal{U}^3$, that results in the desired optimal system behavior.

By denoting the output sequence over the prediction horizon as $\boldsymbol{Y}(k) = [\boldsymbol{y}^T(k+1) \ \dots \ \boldsymbol{y}^T(k+N)]^T$ and the output reference trajectory correspondingly as $\boldsymbol{Y}^*(k)$, (4) can be written in vector form as

$$J(k) = ||\boldsymbol{\Gamma}\boldsymbol{x}(k) + \boldsymbol{\Upsilon}\boldsymbol{U}(k) - \boldsymbol{Y}^*(k)||_2^2 + \lambda_u ||\boldsymbol{S}\boldsymbol{U}(k) - \boldsymbol{E}\boldsymbol{u}_{abc}(k-1)||_2^2\,, \tag{6}$$

with the matrices $\boldsymbol{\Gamma}$, $\boldsymbol{\Upsilon}$, $\boldsymbol{S}$, and $\boldsymbol{E}$ being defined in the appendix.

After some algebraic manipulations, described in detail in [6], (6) becomes

$$J(k) = \Big(\boldsymbol{U}(k) + \boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\Big)^T \boldsymbol{H} \Big(\boldsymbol{U}(k) + \boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\Big) + \mathrm{const}(k), \tag{7}$$

where the Hessian matrix $\boldsymbol{H}$ and the time-varying term $\boldsymbol{\Theta}(k)$ are defined as

$$\boldsymbol{H} = \boldsymbol{\Upsilon}^T\boldsymbol{\Upsilon} + \lambda_u \boldsymbol{S}^T\boldsymbol{S} \quad \text{and} \tag{8}$$

$$\boldsymbol{\Theta}(k) = \boldsymbol{\Upsilon}^T\big(\boldsymbol{\Gamma}\boldsymbol{x}(k) - \boldsymbol{Y}^*(k)\big) - \lambda_u \boldsymbol{S}^T\boldsymbol{E}\boldsymbol{u}_{abc}(k-1)\,. \tag{9}$$

As shown in [6], function (7) can be written such that the associated optimization problem is a truncated integer least-squares (ILS) one, i.e.,

$$\begin{aligned} &\text{minimize} \quad ||\overline{\boldsymbol{U}}_{\text{unc}}(k) - \boldsymbol{V}\boldsymbol{U}(k)||_2^2 \\ &\text{subject to} \quad \boldsymbol{U}(k) \in \mathbb{U}\,. \end{aligned} \tag{10}$$

The nonsingular, lower triangular matrix $\boldsymbol{V} \in \mathbb{R}^{3N\times 3N}$ is system dependent and is known as the lattice generator matrix that generates the discrete space (i.e., lattice) in which the solution lies. Moreover, $\overline{\boldsymbol{U}}_{\text{unc}}(k) = \boldsymbol{V}\boldsymbol{U}_{\text{unc}}(k)$, where $\boldsymbol{U}_{\text{unc}}(k)$ is given by

$$\boldsymbol{U}_{\text{unc}}(k) = -\boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\,, \tag{11}$$

and it is the unconstrained solution, i.e., the solution that minimizes (7) when relaxing the feasible set from $\mathbb{U}$ to $\mathbb{R}^{3N}$. The optimal integer solution $\boldsymbol{U}_{\text{opt}}(k)$ of (10) represents the lattice point with the shortest Euclidean distance to $\boldsymbol{U}_{\text{unc}}(k)$. The latter can be found in a computationally efficient manner by employing a branch-and-bound algorithm, called *sphere decoder*, as explained in the next section.

Sphere Decoding Algorithm

In the following the sphere decoding algorithm as discussed in [6] is presented along with refinements that relate to its real-time implementation. According to its principle, a $3N$-dimensional hypersphere of radius ρ centered at the unconstrained solution $\overline{\boldsymbol{U}}_{\text{unc}}(k)$ is computed. By doing so, only the candidate solutions inside the sphere have to be evaluated. The goal of the optimizer is to tighten the radius ρ incrementally until only the optimal solution remains inside the sphere, whereas all other candidate solutions are excluded since they constitute suboptimal options. The initial radius ρ_{ini} should be chosen such that the resulting sphere is as small as possible for the majority of lattice points to be excluded a priori, but not too small so that at least one lattice point is enclosed. To this end, two initial radii are computed based on [14]

$$\rho(k) = ||\overline{\boldsymbol{U}}_{\text{unc}}(k) - \boldsymbol{V}\boldsymbol{U}(k)||_2 \,. \tag{12}$$

First, ρ_{bab} is calculated based on the so-called Babai estimate, i.e., by rounding the unconstrained solution to its nearest integer, i.e., $\boldsymbol{U}_{\text{bab}}(k) = \lfloor \boldsymbol{U}_{\text{unc}}(k) \rceil \in \mathbb{U}$. In addition, the radius ρ_{edu} for the educated guess $\boldsymbol{U}_{\text{edu}}$ is computed, which stems from the previously calculated optimal solution $\boldsymbol{U}_{\text{opt}}(k-1)$ shifted by one time-step forward and repeating the last entry, as introduced in (40) in [6].

With the initial radius chosen as the minimum of the two options, i.e.,

$$\rho_{\text{ini}}(k) = \min\{\rho_{\text{bab}}(k), \rho_{\text{edu}}(k)\}, \tag{13}$$

the optimization process proceeds by evaluating the lattice points inside the sphere to extract the optimal solution. The pseudocode of the search process is presented in Algorithm 1. As shown, the sphere decoder traverses a search tree of $3N$ levels to find the optimal solution $\boldsymbol{U}_{\text{opt}}(k)$. To this aim, each node of the tree (i.e., a new *single-phase* switch position) is explored and the corresponding intermediate distance between the node explored, i.e., the thus far assembled sequence of switch positions, and $\boldsymbol{U}_{\text{unc}}(k)$ of appropriate dimensions is computed (line 7 in Algorithm 1).

If this distance—calculated in Algorithm 2—is smaller than the most recently updated upper bound ρ^2—initially being $\rho^2_{\text{ini}}(k)$—then the algorithm proceeds by examining the node at the next level of the tree, otherwise the remaining branch is pruned. This procedure is repeated until the bottom level (i.e., leaf node) of the tree is reached; at this point a tentative solution $\boldsymbol{U}_{\text{opt}}(k)$ is found and the sphere is tightened (lines 9–13 in Algorithm 1). Lines 21 to 27 in Algorithm 1 enable the sphere decoder to backtrack and visit unexplored nodes of the search tree. Finally, to guarantee optimality, all other possible paths from the higher levels of the tree towards the bottom one are being traversed until no better solution than the tentative one exists. Once all nodes have been explored or pruned, a certificate of optimality is provided (line 29 in Algorithm 1), meaning that the optimal solution has been found.

Lastly, after the sphere decoder terminates, the calculated sequence of switch positions is stored for reuse in the next control cycle to calculate the switching effort $\Delta\boldsymbol{u}_{abc}$ and the educated guess $\boldsymbol{U}_{\text{edu}}$ based on (12). According to the receding horizon principle, the computed three-phase switch position that corresponds to step k, i.e., $\boldsymbol{u}_{\text{opt}}(k)$, is sent to the inverter. For more details on the functionality of the sphere decoder the interested reader is referred to [6] and [15].

Looking into the provided pseudocodes into more detail, Algorithm 1 is identical to the one proposed in [11] with two proposed modifications to ensure hard real-time compliance and reduction of the computational complexity. For one, due to the non-deterministic number of nodes the sphere decoder has to visit, a maximum number of iterations of the for loop is imposed in line 5 in Algorithm 1. In this work, an upper limit of 130 nodes ensures that the execution time of the sphere decoder does not violate the hard real-time requirement of the control algorithm operating at $40\,\text{kHz}$. Thus, if the algorithm does not terminate before reaching this upper bound, the tentative solution with the lowest associated cost is used. This can be either the initial guess or a better solution found in line 10 of Algorithm 1, implying that a possibly suboptimal choice is applied to the inverter.

Algorithm 1 Sphere decoding algorithm

1: **function** SPHDEC($\rho^2, \overline{\boldsymbol{U}}_{\text{unc}}$)
2: $j = 1$ ▷ search level
3: set each element in $\mathbf{sp}[3N] = -1$
4: set each element in $\boldsymbol{d}[3N] = 0$
5: **for** maximum number of iterations **do**
6: $u_j = \text{sp}_j$
7: $d'^2 = \text{DISTCALC}(\boldsymbol{U}, \text{sp}_j, d_j^2)$
8: **if** $d'^2 \leq \rho^2$ **then**
9: **if** $j = 3N$ **then** ▷ leaf node
10: BetterSolutionFound = true
11: $\boldsymbol{U}_{\text{opt}} = \boldsymbol{U}$
12: $\rho^2 = d'^2$
13: $\text{sp}_j{+}{+}$
14: **else** ▷ not a leaf node
15: $j{+}{+}$ ▷ increase level
16: $d_j^2 = d'^2$
17: **end if**
18: **else** ▷ prune/explore sibling node
19: $\text{sp}_j{+}{+}$
20: **end if**
21: **for** $q = 3N$ downto 2 **do**
22: **if** $\text{sp}_q > 1$ **then** ▷ backtracking
23: $\text{sp}_q = -1$
24: $j = q - 1$ ▷ decrease level
25: $\text{sp}_j{+}{+}$
26: **end if**
27: **end for**
28: **if** $\text{sp}_1 > 1$ **then** ▷ search completed
29: OptimalSolutionFound = true
30: **break**
31: **end if**
32: **end for**
33: **return** $\boldsymbol{U}_{\text{opt}}$
34: **end function**

Algorithm 2 Distance calculation

1: **function** DISTCALC($\boldsymbol{U}, j, d_j^2$)
2: $\boldsymbol{\Delta} \in \mathbb{R}^{3N \times 2}$ ▷ static to store between calls
3: **if** $u_j = -1$ **then**
4: $\delta_{\text{const}} = \overline{u}_{\text{unc},j} - \boldsymbol{V}_{(j,1:(j-1))}\boldsymbol{U}_{1:(j-1)}$
5: $\delta_j^2|_{u_j=-1} = (\delta_{\text{const}} + v_{(j,j)})^2 + d_j^2$
6: $\delta_j^2|_{u_j=0} = \delta_{\text{const}}^2 + d_j^2$
7: $\delta_j^2|_{u_j=+1} = (\delta_{\text{const}} - v_{(j,j)})^2 + d_j^2$
8: **return** $\delta_j^2|_{u_j=-1}$
9: **else if** $u_j = 0$ **then**
10: **return** $\delta_j^2|_{u_j=0}$
11: **else** ▷ $u_j = +1$
12: **return** $\delta_j^2|_{u_j=+1}$
13: **end if**
14: **end function**

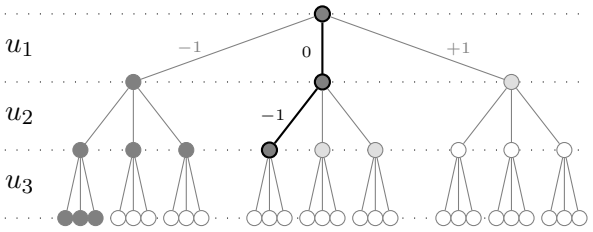


Fig. 2: Snapshot of a search tree being explored. Nodes that are shaded have been visited by the algorithm, whereas unshaded nodes (i.e., the empty circles) have not been visited thus far. The light gray nodes have not been visited, but their intermediate radii have been calculated and stored in the look-up table $\boldsymbol{\Delta}$.

The second proposed modification relates to the distance calculation of each node, see Algorithm 2. One important observation of the search process is that once a node of a particular level has been visited, all its sibling nodes will have to be evaluated as well. However, due to the depth-first search order, in most cases, the sibling nodes are visited at a later point in the search process, e.g., after backtracking from a lower level. Fig. 2 illustrates such a scenario, where the shaded nodes have been evaluated by the algorithm and the thick path indicates the currently explored node.

In line 4 of Algorithm 2, the term δ_{const} is introduced, due to the fact that the preceding switch positions $\boldsymbol{U}_{1:(j-1)}$ are identical for all three sibling nodes, e.g., $u_1 = 0$ in the illustration of Fig. 2. For this reason, δ_{const} is calculated only once when a level is visited for the first time, i.e., when $u_j = -1$. Lines 3 and 23 in Algorithm 1 ensure that the first visited node on each of the $j = 1, 2, \ldots, 3N$ levels is the node corresponding to $u_j = -1$.

Exploiting the idiosyncrasy of FPGAs to parallelize calculations allows the simultaneous computation of the intermediate (squared) radii in lines 5-7 of Algorithm 2 without increasing the execution time.

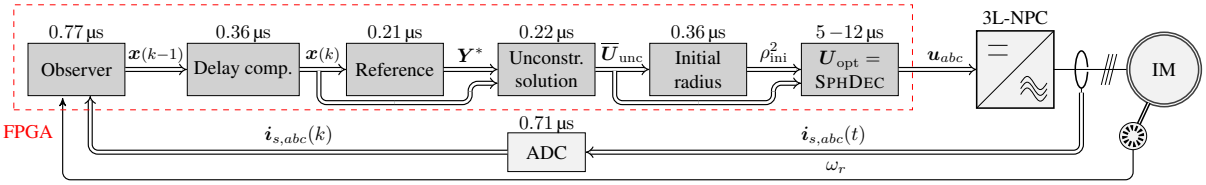


Fig. 3: Block diagram of the experimental setup including the execution time of each block.

Note that the result for $u_j = -1$ is immediately returned, while the other two radii are stored in the static matrix $\boldsymbol{\Delta} \in \mathbb{R}^{3N\times 2}$, with $3N$ resembling the number of levels in the search tree. This reduces every call of `DistCalc` for $u_j \neq -1$ to a simple look-up table, thus reducing its total execution time by two-thirds. Due to the fact that the radius calculation is the step with the highest computational complexity in Algorithm 1, this modification also significantly decreases the total execution time of the sphere decoder. The pre-calculated radii for not (yet) visited nodes are depicted in light gray in Fig. 2.

FPGA Implementation

The sphere decoder is implemented on the open-source control platform UltraZohm [16] which utilizes a Xilinx Zynq UltraScale+ 9EG as processing unit. The UltraScale+ combines a fairly large FPGA and multiple ARM processors, also referred to as processing system (PS), on the same silicon chip allowing an optimized distribution of tasks on the heterogeneous platform. To achieve the sampling interval $T_s = 25\,\mu\text{s}$ and horizon length of $N = 3$ steps, the current control loop is implemented exclusively on the FPGA, while the ARM cores are utilized for initialization, supervision, and data logging. A block diagram of the FPGA implementation of the current control loop is shown in Fig. 3.

The FPGA blocks are initialized by the PS via the advanced extensible interface (AXI). Since the system parameters and relevant matrices, e.g., $\boldsymbol{\Gamma}$ and $\boldsymbol{\Upsilon}$ (defined in the appendix), are assumed to be time invariant they are computed offline. At each time step the stator currents $\boldsymbol{i}_{s,abc}(t)$ are sampled and the rotor speed ω_r is calculated from the feedback of the incremental encoder. Based on these measurements, the state $\boldsymbol{x}(k-1)$ is observed and the time delay introduced by the digital nature of the controller is compensated for using (3), so that $\boldsymbol{x}(k)$ is acquired. With the current state $\boldsymbol{x}(k)$, the reference trajectory $\boldsymbol{Y}^*(k)$, i.e., the values of the reference current within the prediction horizon, is computed. With this information, the unconstrained solution $\overline{\boldsymbol{U}}_{\text{unc}}(k)$ is calculated with (9) and (11) on the FPGA, and, subsequently, the squared initial radius $\rho^2_{\text{ini}}(k)$ for the sphere decoder is determined based on (12), and by subsequently evaluating (13).

Table I: FPGA resource utilization of look-up tables (LUTs), flip-flops (FFs), block memory (BRAM), and digital signal processing (DSP) slices for a Xilinx Zynq UltraScale+ 9EG device [16].

IP Core	DSP	LUT	FF	BRAM	Timing	Arithmetic
Flux Observer	50	5,892	5,079	0	0.77 µs	floating point
Delay Compensation	20	3,275	5,505	3	0.36 µs	floating point
Reference Calculation	31	3,354	2,976	1	0.21 µs	floating point
Predictive Controller	192	36,571	19,377	3	4.0–11.2 µs	fixed point
Utilization in total	293	49,092	32,937	7	5.4–12.5 µs	
Utilization in %	11.6%	17.9%	6.0%	0.8%	22–50 %	

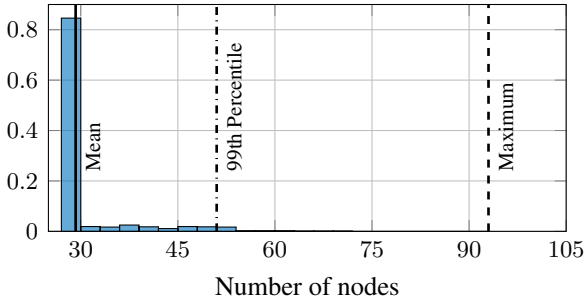


Fig. 4: Histogram of visited nodes.

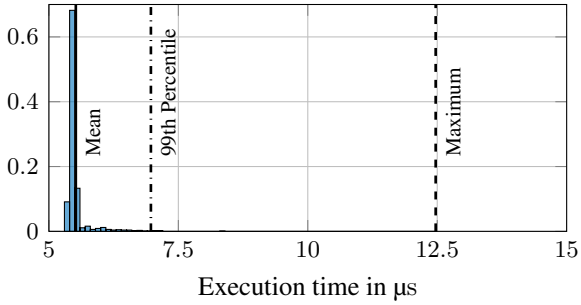


Fig. 5: Histogram of execution time of pre-calculations and predictive control algorithm.

Table I summarizes the resource utilization, timing, and arithmetic logic for each block of the FPGA implementation depicted in Fig. 3. Since the calculation blocks of the unconstrained solution, initial radius, and sphere decoding share the system-dependent matrices, they are implemented as one intellectual property (IP) core and are therefore lumped together in Table I as "Predictive Controller". In addition, the total resources required to implement the design, as well as the utilization of the total resources available on the chip are stated, suggesting that it would also be possible to run the implementation on a smaller FPGA with less resources. The timing utilization in % (see the last row of Table I) refers to the execution time of the total control period of 25 µs, indicating that either the control period, the prediction horizon, or both could be further increased. Note that the observer, delay compensation, and reference calculation blocks are implemented in *floating-point* arithmetic, while the remaining part of the predictive controller is realized in *fixed-point* logic. This choice is based on the trade-off between improved timing, and resources usage for fixed point and ease of implementation for floating-point logic.

Fig. 4 shows the histogram for the number of visited nodes required for the sphere decoder to find the optimal solution during steady-state operation. In 85% of the occurrences, the sphere decoder requires only the minimum number of possible nodes, i.e., 27, to guarantee the optimal solution. For further analysis, the mean, 99th percentile and maximum number of nodes are highlighted in Fig. 4. It is important to note that the maximum number of nodes (i.e. 93) is less than the maximum number of allowed search iterations, i.e., 130, introduced to ensure the hard real-time compliance. This implies, that the optimal solution is always found in steady-state operation and the proposed modifications do not introduce any suboptimialty.

Fig. 5 displays the histogram of the execution time for the entire control algorithm, including all blocks that are listed in Table I during steady-state operation. As can be observed, the execution time varies between 5.4 and 12.5 µs due to the heuristic (i.e., non-deterministic) nature of the sphere decoder tree search. Regardless of this, the mean execution time is 5.52 µs, while the 99th percentile is 6.97 µs, i.e., significantly less than the chosen T_s of 25 µs.

Table II: Rated values of the induction machine

Parameter	**SI**
rated voltage	380 V
rated current	5 A
rated torque	9 N m
rated speed	2870 rpm

Table III: Parameters of the drive system

Parameter	**SI**	**p.u.**
dc-link voltage V_{dc}	560 V	1.8
stator resistance R_s	2.1 Ω	0.049
rotor resistance R_r	2.2 Ω	0.052
mutual inductance L_m	340 mH	2.44
stator leakage inductance L_{ls}	10.1 mH	0.072
rotor leakage inductance L_{lr}	10.1 mH	0.072

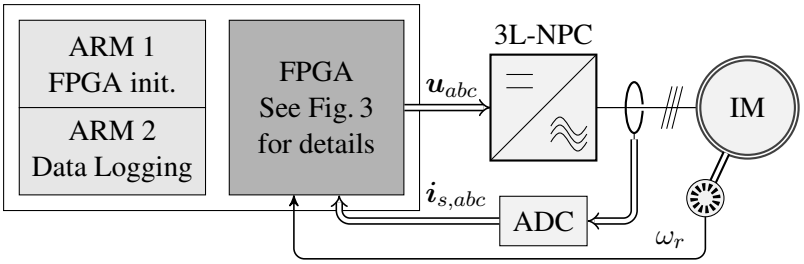
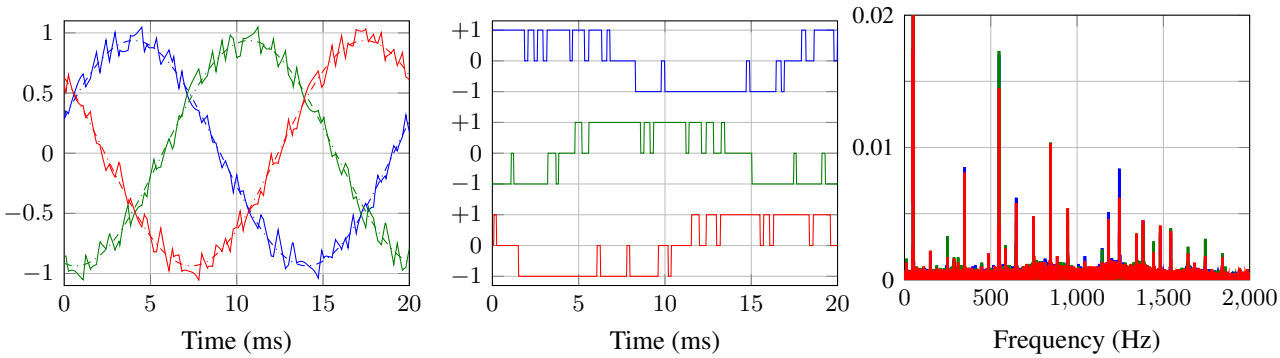


Fig. 6: Block diagram of experimental setup.



(a) Stator currents $\boldsymbol{i}_{s,abc}$ (solid) and their references (dash-dotted).

(b) Switch positions $\boldsymbol{u}_{abc}$.

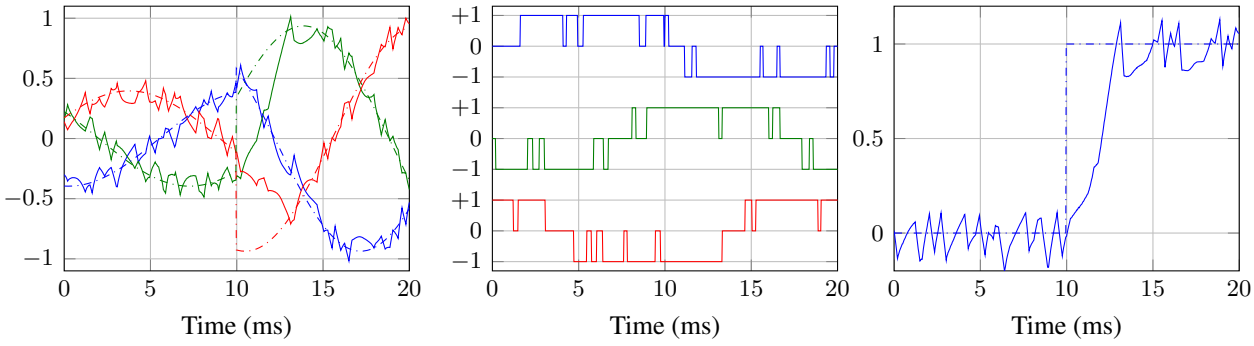
(c) Stator current spectrum.

Fig. 7: Experimental results during steady-state operation with horizon $N = 3$ and $f_{\text{sw}} = 300\,\text{Hz}$.

Experimental Results

For the presented results, a low-voltage (LV) drive shown in Fig. 1 is considered and both steady-state and transient performance at rated speed are investigated. A block diagram of the experimental setup is provided in Fig. 6, while the rated parameters as well as those of the drive system are listed in Tables II and III, respectively. Note that the per unit (p.u.) value of the total leakage inductance of the IM is 0.142. The dc-link voltage is realized by two independent power supplies; one across each of the top and bottom capacitors ensuring a fixed neutral point potential with minimum fluctuations. The load machine operates in constant-speed mode, while both inverters (i.e., the one driving the controlled machine and the one the load machine) share the same dc-link voltage, so that only the losses are supplied by the power supplies. Regarding the controller parameters, as mentioned before, the sampling interval is $T_s = 25\,\mu\text{s}$ and a three-step prediction horizon ($N = 3$) is realized. Moreover, λ_u is tuned such that an average device switching $f_{\text{sw}} = 300\,\text{Hz}$ results in all cases examined. Finally, all results are shown in the p.u. system.

The steady-state performance of the controller is shown in Fig. 7. For the depicted scenario, the torque reference is kept constant and equal to 1 p.u. As can be seen in Fig. 7a, the implemented MPC algorithm achieves a good current reference tracking, resulting in a stator current total harmonic distortion (THD) of 8.8%. The corresponding current spectrum is depicted in Fig. 7c. Considering the relatively low value of the total leakage reactance and the low switching frequency, such a THD value is reasonably good. This performance is achieved thanks to the three-step horizon that enables the algorithm to make better educated decisions. Moreover, the adopted low sampling interval results in a high granularity of switching, since the ratio of sampling-to-switching frequency is greater than 100. As discussed in [10], such a high ratio leads to favorable operation of the power electronic system when controlled by FCS-MPC, as also verified in this work. Finally, it is noteworthy that, as mentioned in the previous section, the sphere decoder always found the optimal solution despite the upper bound on the nodes allowed to be explored. Hence, the depicted behavior is indeed the optimal.

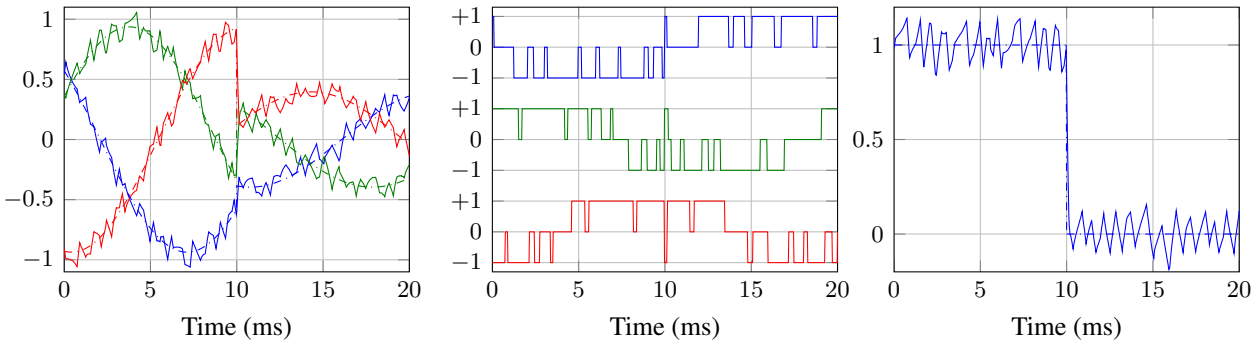


(a) Stator currents $\boldsymbol{i}_{s,abc}$ (solid) and their references (dash-dotted).

(b) Switch positions $\boldsymbol{u}_{abc}$.

(c) Electromagnetic torque T_e (solid) and its reference (dash-dotted).

Fig. 8: Experimental results for a torque reference step-up change from 0 to 1 p.u.



(a) Stator currents $\boldsymbol{i}_{s,abc}$ (solid) and their references (dash-dotted).

(b) Switch positions $\boldsymbol{u}_{abc}$.

(c) Electromagnetic torque T_e (solid) and its reference (dash-dotted).

Fig. 9: Experimental results for a torque reference step-down change from 1 to 0 p.u.

With regards to the the dynamic behavior of the drive, this is shown in Figs. 8 and 9. Specifically, Fig. 8 shows the drive behavior in a step-up change in the torque reference (translated into current reference change), while Fig. 9 depicts the dynamic performance when the torque reference is changed in a step-down manner. As can be seen, thanks to the inherent fast dynamics of direct control, MPC exhibits excellent behavior during transients with as short a settling time as possible, during both examined scenarios. Nonetheless, it is worth mentioning that in some cases the maximum number of allowed nodes was reached, meaning that suboptimal solutions were occasionally applied during the transient phenomena. However, this did not detract from the controller since it still managed to exhibit very fast responses.

Conclusion

This paper presented a computationally efficient real-time implementation of long-horizon FCS-MPC on an FPGA. Modifications in the sphere decoder that significantly reduce the computation time were presented. Experimental results showed the effectiveness of the algorithm based on a test case of a three-level NPC inverter driving an IM. As shown, thanks to the proposed refinements, and for the considered system, a prediction horizon of three steps with a maximum execution time of 15 µs were achieved while guaranteeing optimality during steady-state operation. Hence, an as low control frequency as 40 kHz could be easily achieved, ensuring high granularity of switching, and, thus, favorable system performance.

Appendix

$$\boldsymbol{C}=\begin{bmatrix}1&0\\0&1\\0&0\\0&0\end{bmatrix}^T,\ \boldsymbol{F}=\begin{bmatrix}-\frac{1}{\tau_s}&0&\frac{X_m}{\tau_r D}&\omega_r\frac{X_m}{D}\\0&-\frac{1}{\tau_s}&-\omega_r\frac{X_m}{D}&\frac{X_m}{\tau_r D}\\\frac{X_m}{\tau_r}&0&-\frac{1}{\tau_r}&-\omega_r\\0&\frac{X_m}{\tau_r}&\omega_r&-\frac{1}{\tau_r}\end{bmatrix},\ \boldsymbol{G}=\frac{X_r}{D}\frac{V_{dc}}{2}\begin{bmatrix}1&0\\0&1\\0&0\\0&0\end{bmatrix}\boldsymbol{K},\ \boldsymbol{K}=\frac{2}{3}\begin{bmatrix}1&-\frac{1}{2}&-\frac{1}{2}\\0&\frac{\sqrt{3}}{2}&-\frac{\sqrt{3}}{2}\end{bmatrix}$$

$$\boldsymbol{\Gamma}=\begin{bmatrix}\boldsymbol{CA}\\\boldsymbol{CA}^2\\\vdots\\\boldsymbol{CA}^N\end{bmatrix},\ \boldsymbol{\Upsilon}=\begin{bmatrix}\boldsymbol{CB}&\boldsymbol{0}&\dots&\boldsymbol{0}\\\boldsymbol{CAB}&\boldsymbol{CB}&\dots&\boldsymbol{0}\\\vdots&\vdots&&\vdots\\\boldsymbol{CA}^{N-1}\boldsymbol{B}&\boldsymbol{CA}^{N-2}\boldsymbol{B}&\dots&\boldsymbol{CB}\end{bmatrix},\ \boldsymbol{E}=\begin{bmatrix}\boldsymbol{I}_3\\\boldsymbol{0}_{3\text{x}3}\\\vdots\\\boldsymbol{0}_{3\text{x}3}\end{bmatrix},\ \boldsymbol{S}=\begin{bmatrix}\boldsymbol{I}_3&\boldsymbol{0}_{3\text{x}3}&\dots&\boldsymbol{0}_{3\text{x}3}\\-\boldsymbol{I}_3&\boldsymbol{I}_3&\dots&\boldsymbol{0}_{3\text{x}3}\\\boldsymbol{0}_{3\text{x}3}&-\boldsymbol{I}_3&\dots&\boldsymbol{0}_{3\text{x}3}\\\vdots&\vdots&&\vdots\\\boldsymbol{0}_{3\text{x}3}&\boldsymbol{0}_{3\text{x}3}&\dots&\boldsymbol{I}_3\end{bmatrix}$$

References

[1] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Madison, WI: Nob Hill, 2009.

[2] P. Cortés, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodríguez, "Predictive control in power electronics and drives," *IEEE Trans. Ind. Electron.*, vol. 55, no. 12, pp. 4312–4324, Dec. 2008.

[3] P. Karamanakos, E. Liegmann, T. Geyer, and R. Kennel, "Model predictive control of power electronic systems: Methods, results, and challenges," *IEEE Open J. Ind. Appl.*, vol. 1, pp. 95–114, 2020.

[4] T. Geyer, P. Karamanakos, and R. Kennel, "On the benefit of long-horizon direct model predictive control for drives with LC filters," in *Proc. IEEE Energy Convers. Congr. Expo.*, Pittsburgh, PA, USA, Sep. 2014, pp. 3520–3527.

[5] P. Karamanakos, T. Geyer, N. Oikonomou, F. D. Kieferndorf, and S. Manias, "Direct model predictive control: A review of strategies that achieve long prediction intervals for power electronics," *IEEE Ind. Electron. Mag.*, vol. 8, no. 1, pp. 32–43, Mar. 2014.

[6] T. Geyer and D. E. Quevedo, "Multistep finite control set model predictive control for power electronics," *IEEE Trans. Power Electron.*, vol. 29, no. 12, pp. 6836–6846, Dec. 2014.

[7] R. Baidya, R. P. Aguilera, P. Acuña, S. Vasquez, and H. d. T. Mouton, "Multistep model predictive control for cascaded H-bridge inverters—Formulation and analysis," *IEEE Trans. Power Electron.*, vol. 33, no. 1, pp. 876–886, Jan. 2018.

[8] P. Acuña, C. Rojas, R. Baidya, R. P. Aguilera, and J. Fletcher, "On the impact of transients on multistep model predictive control for medium-voltage drives," *IEEE Trans. Power Electron.*, vol. 34, no. 9, pp. 8342–8355, Sep. 2019.

[9] A. Andersson and T. Thiringer, "Assessment of an improved finite control set model predictive current controller for automotive propulsion applications," *IEEE Trans. Ind. Electron.*, vol. 67, no. 1, pp. 91–100, Jan. 2020.

[10] P. Karamanakos and T. Geyer, "Guidelines for the design of finite control set model predictive controllers," *IEEE Trans. Power Electron.*, vol. 35, no. 7, pp. 7434–7450, Jul. 2020.

[11] M. Dorfling, H. Mouton, T. Geyer, and P. Karamanakos, "Long-horizon finite-control-set model predictive control with non-recursive sphere decoding on an FPGA," *IEEE Trans. Power Electron.*, vol. 35, no. 7, pp. 7520–7531, Jul. 2020.

[12] S. A. Bin Khalid, E. Liegmann, P. Karamanakos, and R. Kennel, "High-level synthesis of a long horizon model predictive control algorithm for an FPGA," in *Proc. Int. Exhib. and Conf. for Power Electron., Intell. Motion, Renew. Energy and Energy Manag.*, Nuremberg, Germany, Jul. 2020, pp. 1544–1551.

[13] J. Holtz, "The representation of ac machine dynamics by complex signal flow graphs," *IEEE Trans. Ind. Electron.*, vol. 42, no. 3, pp. 263–271, Jun. 1995.

[14] P. Karamanakos, T. Geyer, and R. Kennel, "Suboptimal search strategies with bounded computational complexity to solve long-horizon direct model predictive control problems," in *Proc. IEEE Energy Convers. Congr. Expo.*, Montreal, QC, Canada, Sep. 2015, pp. 334–341.

[15] T. Geyer, *Model predictive control of high power converters and industrial drives*. Hoboken, NJ: Wiley, 2016.

[16] S. Wendel, A. Geiger, E. Liegmann, D. Arancibia, E. Durán, T. Kreppel, F. Rojas, F. Popp-Nowak, M. Diaz, A. Dietz, R. Kennel, and B. Wagner, "UltraZohm—A powerful real-time computation platform for MPC and multi-level inverters," in *Proc. Workshop on Pred. Control of Elect. Drives and Power Electron.*, Quanzhou, China, May 2019, pp. 1–6.