Long-Horizon Finite-Control-Set Model Predictive Control With Non-Recursive Sphere Decoding on an FPGA

Tinus Dorfling, *Student Member, IEEE*, Toit Mouton, *Member, IEEE*, Tobias Geyer, *Senior Member, IEEE* and Petros Karamanakos, *Senior Member, IEEE*

***Abstract*—Long-horizon finite-control-set model predictive control is implemented on a field-programmable gate array (FPGA). To solve the underlying least-squares integer program, a non-recursive sphere decoding algorithm is developed. By exploiting the problem structure, few multipliers are required, and the algorithm computes the optimal solution in a few clock cycles, thus achieving a resource-efficient implementation on the FPGA. For a prediction horizon of five steps and a three-level converter, 87 digital signal processor (DSP) blocks and an execution time of at most $13.4\,\mu s$ was required to solve the optimization problem during steady-state operation. Experimental results verify the effectiveness of the long-horizon controller.**

***Index Terms*—Model predictive control, optimal control, field-programmable gate array, integer programming, sphere decoder, three-level neutral-point clamped (NPC) converters, integer least-squares**

I. Introduction

In recent years, the use of and research in model predictive control (MPC) for power electronic applications has significantly increased [1]–[3]. More specifically, *finite-control-set* (FCS) MPC, a direct MPC method, is predominately considered for power electronic converters [4], [5]. Direct MPC methods consider the case in which the controller directly manipulates the switch positions of a converter. Due to the absence of a modulator, the output of the controller (i.e., the manipulated variable determining the switch position) is restricted to a set of integers. The adoption of long horizons, which are used to predict the trajectories of the state variables over multiple discrete time steps, provides a substantial improvement over short horizons by lowering the harmonic distortion of the current [6]. Long horizons are potentially beneficial for higher-order plants, such as converters with an LC filter [7]. Thanks to its multiple-input multiple-output approach, long-horizon FCS-MPC achieves damping of the resonance without additional damping control loops.

The optimization problem underlying FCS-MPC is an integer program usually with a quadratic cost function. Since the admissible set formed by the manipulated variables is non-convex, solving the integer program requires significant computational effort. In fact, integer programs are known to be non-deterministic polynomial-time hard. This means that, in general, when increasing the horizon, the upper bound on the computation time increases exponentially. For this reason, the implementation of long-horizon FCS-MPC on embedded control hardware remains largely an unexplored topic.

For short horizons, the optimization problem can be solved via enumeration of all the possible solutions. This is known as *exhaustive search*. Due to the exponential increase in the number of possible solutions, this approach is not suited for real-time implementation with horizons beyond $N_p > 2$, where N_p denotes the number of the prediction steps. In [8] an appropriate terminal weight was chosen, thereby approximating the infinite horizon case. This enables low harmonic distortions even with short horizons. Horizons of $N_p = 1$ and 2, with a sampling interval of $T_s = 25\,\mu s$, were implemented on a field-programmable gate array (FPGA). By exploiting parallel computations on the FPGA, the optimal solution was found in $5.8\,\mu s$ and $17.2\,\mu s$ for $N_p = 1$ and 2, respectively. Note that as the horizon was increased, the computation time significantly increased due to the use of enumeration.

For long horizons of $N_p > 2$, efficient solvers have been proposed and implemented by reformulating the integer program as an integer least-squares (ILS) problem. In [9], a Voronoi diagram is used to partition the search space into nearest-neighbour regions. A binary search tree is built offline and traversed in real-time to find the Voronoi region corresponding to the optimal solution. However, calculating the hyperplanes that define the Voronoi regions quickly becomes intractable as the horizon increases, because the number of hyperplanes defining the regions, and thus the storage requirements, increases exponentially. Algorithms such as the *iterative slicer* [10] and *Micciancio-Voulgaris* algorithm [11] have been investigated in simulations. According to these simulations, the computation time of these algorithms rapidly increases for horizons beyond $N_p > 3$. These algorithms also have significant storage requirements.

One of the most promising approaches to solve the ILS problem is *sphere decoding* [12], [13], a branch-and-bound method. As proposed in [5], the sphere decoder is suited to solve the optimization problem online with relatively little preprocessing and modest storage requirements, while quickly finding the optimal solution in a depth-first search manner, as discussed in [14]. The computational burden of the sphere

T. Dorfling and T. Mouton are with Department of Electrical and Electronic Engineer, Stellenbosch University, Stellenbosch 7599; South Africa; e-mail: mddorfling@sun.ac.za, dtmouton@sun.ac.za

T. Geyer is with ABB Corporate Research, Baden-Dättwill 5405, Switzerland; e-mail: t.geyer@ieee.org

P. Karamanakos is with the Faculty of Information Technology and Communication Sciences, Tampere University, 33101 Tampere, Finland; e-mail: p.karamanakos@ieee.org

decoder scales well when increasing the prediction horizon or when increasing the number of state variables. Simulations show that for horizons $N_p = 1$ to 10, the average number of iterations required increases almost linearly with the length of the horizon [3]. The sphere decoder has been practically implemented in [15]–[18]. In [15], a horizon of $N_p = 5$ was implemented on an FPGA for a three-level converter, and the sampling interval was set to $T_s = 25\,\mu\text{s}$. The sphere decoder required (along with the pre-processing calculations) at most $16.2\,\mu\text{s}$ to solve the optimization problem. In [16], a horizon of $N_p = 3$ was achieved on a dSpace system for a five-level converter, and the sampling interval was chosen as $T_s = 100\,\mu\text{s}$. The control algorithm required $98\,\mu\text{s}$ to find the optimal solution. In [17], a horizon of $N_p = 4$ was achieved on a dSpace system for a three-level converter. The sampling interval was set to $T_s = 125\,\mu\text{s}$ and the control algorithm required at most $112.9\,\mu\text{s}$ to find the optimal solution. In [18], a horizon of $N_p = 4$ was implemented on a dSpace system for a two-level converter. The sampling interval was set to $T_s = 25\,\mu\text{s}$ and the control algorithm required at most (roughly) $T_s = 24\,\mu\text{s}$.

Another approach to solve the optimization problem underlying direct MPC is to solve the optimization problem offline for *all* possible states by deriving the (explicit) state-feedback control law [19]. This approach is known as *explicit* MPC. Power electronic systems are known to be *hybrid systems*, which can be modelled as piecewise-affine or mixed-logical dynamic systems [20]. The optimization problem is solved offline via multi-parametric programming and the (integer) optimal switch position is stored in a lookup table [21]. During real-time operation, a binary search tree (that is constructed offline) is traversed to find the appropriate switch position for a given state vector. In the case of many state variables or long horizons, the offline computations become intractable and the memory requirements increase significantly. Therefore, this approach is only suitable for low dimensional problems.

The contribution of this paper is threefold. The first contribution is a non-recursive sphere decoding algorithm. In contrast to the recursive sphere decoding algorithm introduced in [5], and implemented on a dSpace in [16] and [17], such an approach is ill-suited to an FPGA implementation, as recursion on an FPGA is realized by synthesizing the sphere decoder multiple times to account for the recursion depth, leading to an unacceptable amount of resources. Thus, when compared to the recursive sphere decoder, the proposed non-recursive variant enables a resource-efficient implementation on an FPGA. The second contribution is the implementation of long-horizon FCS-MPC on a low-cost FPGA for a prediction horizon of $N_p = 5$ with a short sampling interval of $T_s = 25\,\mu\text{s}$. The implementation process, which thus far has not been directly addressed in the literature, is described in detail. The problem structure is exploited to allow for an efficient implementation. The final contribution is a comprehensive experimental assessment of FCS-MPC for a three-level neutral-point clamped (NPC) converter with an RL load with different horizon lengths and for a wide range of switching frequencies. The presented results verify that long horizons do offer performance benefits for a typical converter system.

This paper is organized as follows. Section II describes the dynamic model of the system and the control problem. Section III briefly revises the reformulated ILS problem and the notion of sphere decoding. Section IV presents a detailed description of the non-recursive implementation of long-horizon FCS-MPC on an FPGA. The computational burden and resource usage is investigated. Experimental results with long horizons are presented in Section VI, including a steady-state comparison between horizons $N_p = 1$, 3, and 5, as well as a demonstration of the closed-loop response to step changes in the current references when adopting the horizon $N_p = 5$. Conclusions are provided in Section VII.

II. Controller Formulation

As a case study, consider a neutral-point-clamped (NPC) inverter with an RL load as shown in Fig. 1. The described implementation of long-horizon FCS-MPC is sufficiently general so that it is applicable also to other power electronic systems, including grid-connected converters or electric machines.

A. Dynamic Model

To ensure that the model is linear, the potential of the neutral point N (see Fig. 1) is fixed to zero. The dc-link voltage is denoted by V_d. The converter can synthesise three voltage levels (with respect to the neutral point N), which are given by

$$v_x = \frac{V_d}{2} u_x, \tag{1}$$

where $x \in \{a, b, c\}$ denotes phase x and $u_x \in \{-1, 0, 1\} = \mathcal{U}$ represents the switch position of the particular phase. To simplify the controller formulation, three-phase abc variables are transformed to the $\alpha\beta$-orthogonal reference frame:

$$\boldsymbol{\xi}_{\alpha\beta} = \boldsymbol{K}\boldsymbol{\xi}_{abc}, \tag{2}$$

where

$$\boldsymbol{K} = \frac{2}{3}\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}. \tag{3}$$

By taking the orthogonal reference frame currents $\boldsymbol{i}_{\alpha\beta}(t) = \begin{bmatrix} i_\alpha(t) & i_\beta(t) \end{bmatrix}^T$ as the state variables and the three-phase switch positions $\boldsymbol{u}(t) = \begin{bmatrix} u_a(t) & u_b(t) & u_c(t) \end{bmatrix}^T \in \boldsymbol{\mathcal{U}} = \mathcal{U} \times \mathcal{U} \times \mathcal{U}$ as the input (or manipulated) variables, the continuous-time state-space representation of the system is given by

$$\frac{d\,\boldsymbol{i}_{\alpha\beta}(t)}{dt} = \boldsymbol{F}\boldsymbol{i}_{\alpha\beta}(t) + \boldsymbol{G}\boldsymbol{u}(t). \tag{4}$$

The state and input matrices are

$$\boldsymbol{F} = -\frac{R}{L}\boldsymbol{I}_2 \text{ and } \boldsymbol{G} = \frac{V_d}{2L}\boldsymbol{K}, \tag{5}$$

respectively, where $\boldsymbol{I}_2$ is the identity matrix with dimensions 2×2. By using the exact discretization [22], the discrete-time state-space representation of (4) is given by[1]

$$\boldsymbol{i}(k+1) = \boldsymbol{A}\boldsymbol{i}(k) + \boldsymbol{B}\boldsymbol{u}(k), \tag{6}$$

[1]The $\alpha\beta$-subscript has been dropped for convenience and unless stated otherwise, all currents will be in the orthogonal $\alpha\beta$-reference frame.

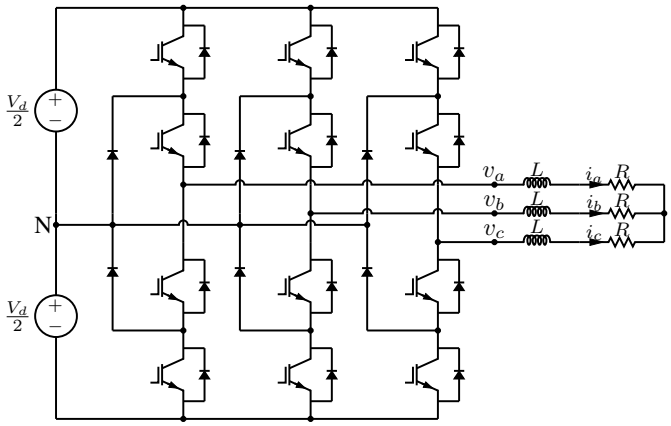


Fig. 1: Three-level NPC converter with an RL load.

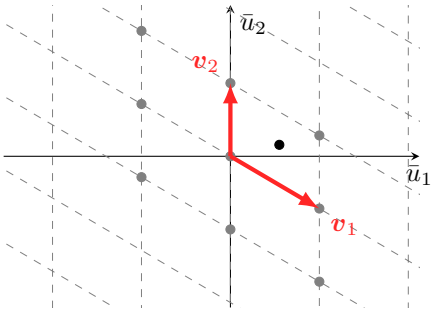


Fig. 2: Example of a lattice generated by $\boldsymbol{V}$, the columns of which form the basis for the lattice. The (transformed) unconstrained optimum is indicated by the black solid circle and the lattice points (representing feasible solutions) are depicted by the grey solid circles.

where

$$\boldsymbol{A} = e^{\boldsymbol{F}T_s} \text{ and } \boldsymbol{B} = -\boldsymbol{F}^{-1}(\boldsymbol{I}_2 - \boldsymbol{A})\boldsymbol{G}, \tag{7}$$

with $e^{\boldsymbol{F}T_s}$ being the matrix exponential of $\boldsymbol{F}T_s$ and T_s the sampling interval.

B. Cost Function and Optimization Problem

By using the state-space representation of (6), MPC predicts the evolution of the sampled state vector $\boldsymbol{i}(k)$ over the prediction horizon N_p as a function of the manipulated variable $\boldsymbol{u}(l)$, with $l = k, k+1, \ldots, k+N_p-1$. To this end, we define the *switching sequence* as the sequence of switch positions over the prediction horizon N_p as

$$\boldsymbol{U}(k) = \begin{bmatrix}\boldsymbol{u}^T(k) & \boldsymbol{u}^T(k+1) & \cdots & \boldsymbol{u}^T(k+N_p-1)\end{bmatrix}^T, \tag{8}$$

where $\boldsymbol{U}(k) \in \mathbb{U} = \boldsymbol{\mathcal{U}} \times \boldsymbol{\mathcal{U}} \times \cdots \times \boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{U}}^{N_p}$.

The control objectives are captured by a cost (or objective) function. In power electronics, the control objectives typically include reference tracking and the minimization of the switching effort, which relates to the switching losses. The cost function is chosen as:

$$J(\boldsymbol{i}(k), \boldsymbol{U}(k)) = \sum_{l=k}^{k+N_p-1} \|\boldsymbol{i}_e(l+1)\|_2^2 + \lambda_u \|\Delta\boldsymbol{u}(l)\|_2^2, \tag{9}$$

where $\boldsymbol{i}_e(l+1) = \boldsymbol{i}_{ref}(l+1) - \boldsymbol{i}(l+1)$ represents the tracking error between the reference $\boldsymbol{i}_{ref}$ and the predicted current. The switching effort is represented by $\Delta\boldsymbol{u}(l) = \boldsymbol{u}(l) - \boldsymbol{u}(l-1)$. The weighting factor $\lambda_u > 0$ adjusts the trade-off between the tracking error and the switching losses.

Minimizing (9) over the switching sequence $\boldsymbol{U}(k)$, i.e.,[2]

$$\boldsymbol{U}_{opt}(k) = \arg\min_{\boldsymbol{U}(k)} \quad J \tag{10a}$$

$$\text{subject to (6) and } \boldsymbol{U}(k) \in \mathbb{U}, \tag{10b}$$

yields the *optimal solution* $\boldsymbol{U}_{opt}(k)$ (in other words, the optimal switching sequence) that is predicted to achieve the optimal behavior (as defined by the cost function) of the system. For short horizons, $\boldsymbol{U}_{opt}(k)$ can be found by enumerating all of the possible solutions in $\mathbb{U}$. Note the exponential increase in the number of possible solutions: the set $\mathbb{U}$ has 3^{3N_p} elements.

Since the optimal control problem in (10) is solved in an open-loop manner, feedback is implemented by adopting the *receding horizon* policy. This means that once the optimal switching sequence $\boldsymbol{U}_{opt}(k)$ has been calculated, only the first switching command $\boldsymbol{u}_{opt}(k)$ is applied to the converter. The remaining entries are discarded, and the optimization problem is solved again at the next sampling instant with new state measurements and references.

III. Integer Least-Squares Program

A. Integer Least-Squares Problem

The model of (6) can be included in the cost function (9). After some mathematical manipulations [5], the cost function is rewritten as

$$J = \boldsymbol{U}^T(k)\boldsymbol{H}\boldsymbol{U}(k) + 2\boldsymbol{\Theta}^T(k)\boldsymbol{U}(k) + \theta(k) \tag{11}$$

with

$$\boldsymbol{H} = \boldsymbol{\Upsilon}^T\boldsymbol{\Upsilon} + \lambda_u\boldsymbol{S}^T\boldsymbol{S} \tag{12}$$

$$\boldsymbol{\Theta}(k) = \boldsymbol{\Upsilon}^T[\boldsymbol{\Gamma}\boldsymbol{i}(k) - \boldsymbol{Y}_{ref}(k)] - \lambda_u\boldsymbol{S}^T\boldsymbol{E}\boldsymbol{u}(k-1) \tag{13}$$

$$\theta(k) = \|\boldsymbol{\Gamma}\boldsymbol{i}(k) - \boldsymbol{Y}_{ref}(k)\|_2^2 + \lambda_u \|\boldsymbol{E}\boldsymbol{u}(k-1)\|_2^2, \tag{14}$$

where $\boldsymbol{Y}_{ref}(k)$ is introduced as the reference current over the prediction horizon

$$\boldsymbol{Y}_{ref}(k) = \begin{bmatrix}\boldsymbol{i}_{ref}^T(k+1) & \boldsymbol{i}_{ref}^T(k+2) & \cdots & \boldsymbol{i}_{ref}^T(k+N_p)\end{bmatrix}^T. \tag{15}$$

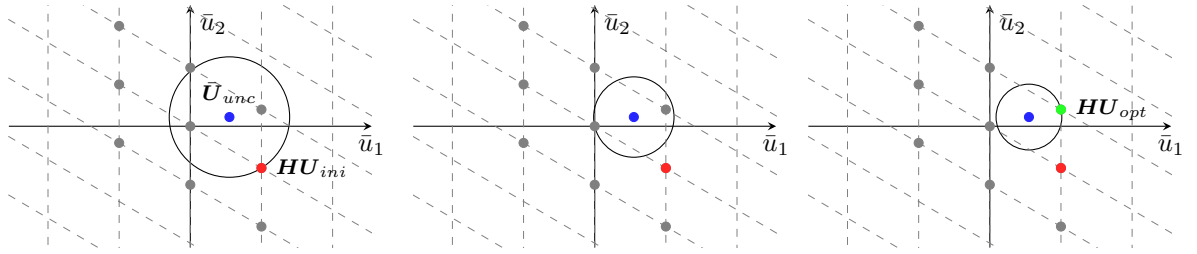
The definition of the matrices $\boldsymbol{\Gamma} \in \mathbb{R}^{2N_p \times 2}$, $\boldsymbol{\Upsilon} \in \mathbb{R}^{2N_p \times 3N_p}$, $\boldsymbol{E} \in \mathbb{R}^{3N_p \times 3}$, and $\boldsymbol{S} \in \mathbb{R}^{3N_p \times 3N_p}$ can be found in the Appendix.

The integer program in (10) can then be reformulated as an ILS problem [5]:

$$\boldsymbol{U}_{opt}(k) = \arg\min_{\boldsymbol{U}(k)} \quad \|\boldsymbol{V}\boldsymbol{U}(k) - \boldsymbol{V}\boldsymbol{U}_{unc}(k)\|_2^2 \tag{16a}$$

$$\text{subject to } \boldsymbol{U}(k) \in \mathbb{U}, \tag{16b}$$

[2]Hereafter, the arguments of J are dropped for convenience.



(a) An initial lattice point is chosen (b) The sphere is tightened (c) The sphere is tightened until only one lattice point lies within the sphere

Fig. 3: Illustration of the sphere decoding principle in two dimensions.

where

$$\boldsymbol{U}_{unc}(k) = -\boldsymbol{H}^{-1}\boldsymbol{\Theta}(k) \in \mathbb{R}^{3N_p} \tag{17}$$

is known as the unconstrained optimum. The latter is the optimal solution if the integer constraints on $\boldsymbol{U}(k)$ are relaxed. The lower-triangular generator $\boldsymbol{V}$ matrix results from a Cholesky decomposition of the positive-definite matrix $\boldsymbol{H}^{-1}$,

$$\boldsymbol{V}^{-1}\boldsymbol{V}^{-T} = \boldsymbol{H}^{-1} \tag{18}$$

and generates a truncated lattice

$$\Lambda = \{\boldsymbol{V}\boldsymbol{U}(k) | \boldsymbol{U}(k) \in \mathbb{U}\} \tag{19}$$

from all possible switching sequences. An example of a lattice is shown in Fig. 2. According to (16), solving the ILS problem amounts to finding the lattice point closest to the unconstrained optimum (hence it is also known as the *closest vector problem*).

B. Sphere Decoding

By adopting a technique from telecommunications, known as sphere decoding, the ILS problem underlying long-horizon FCS-MPC can be solved in a time-efficient manner [5]. The idea behind sphere decoding is to consider possible solutions $\boldsymbol{U}(k)$ in $\mathbb{U}$ that belong to a sphere with radius ρ, which is centred at the transformed unconstrained optimum

$$\bar{\boldsymbol{U}}_{unc}(k) = \boldsymbol{V}\boldsymbol{U}_{unc}(k), \tag{20}$$

i.e., we impose

$$\left\|\bar{\boldsymbol{U}}_{unc}(k) - \boldsymbol{V}\boldsymbol{U}(k)\right\|_2 \leq \rho. \tag{21}$$

By tightening the sphere systematically and excluding sub-optimal solutions, the optimal solution $\boldsymbol{U}_{opt}(k)$ is found once only a single lattice point lies within the sphere. An initial solution is chosen based on either the Babai estimate [23] or an educated guess [3]. The Babai estimate $\boldsymbol{U}_{bab}(k)$ is found by rounding the unconstrained optimum to the nearest (untransformed) integer solution, i.e., $\boldsymbol{U}_{bab}(k) = \lfloor \boldsymbol{U}_{unc}(k) \rceil \in \mathbb{U}$. The educated guess $\boldsymbol{U}_{ed}(k)$ is based on the assumption that the switching sequence computed at the previous time step is similar to the new (and shifted) optimal switching sequence at the current time step. Note that this makes the educated guess ill-suited to transients, since the previous and new switching sequences can look entirely different. Fig. 3 illustrates the principle of sphere decoding.

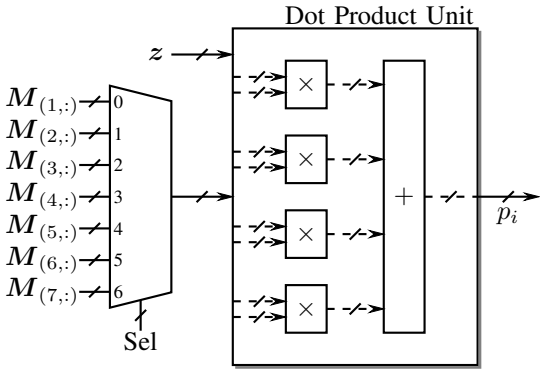


Fig. 4: Illustration of an efficient hardware implementation of the matrix-vector multiplication with four multipliers.

Because the generator matrix $\boldsymbol{V}$ of (18) is lower-triangular, the squared distance to a lattice point is given by

$$\begin{aligned} d'^2(k) = &\overbrace{(\bar{u}_{unc,1} - V_{(1,1)}u_1)^2}^{d_1^2} + \overbrace{(\bar{u}_{unc,2} - V_{(2,1)}u_1 - V_{(2,2)}u_2)^2}^{d_2^2} \\ &+ \cdots \\ &+ \overbrace{(\bar{u}_{unc,3N_p} - V_{(3N_p,1)}u_1 - \cdots - V_{(3N_p,3N_p)}u_{3N_p})^2}^{d_{3N_p}^2}, \end{aligned} \tag{22}$$

where for any matrix $\boldsymbol{\Xi}$ and vector $\boldsymbol{\xi}$, $\Xi_{i,j}$ and ξ_i indicate the ijth and ith entries, respectively. When building the switching sequence $\boldsymbol{U}(k)$ entry by entry, (22) shows that only one term needs to be added to the squared distance. More specifically, when adding the ith switch position u_i, the term d_i^2 is added to the squared distance. This allows for a computationally cheap method to determine whether the intermediate squared distance $d_i'^2$ (i.e., the squared terms up to ith switch position) meets (21) or not. This inexpensive feature is exploited by the recursive sphere decoding algorithm (SDA), which was proposed in [5] (see Algorithm 1 therein).

IV. Practical Implementation of the Controller

A. Implementing Algorithms on an FPGA

Implementing the ILS calculations and sphere decoder (which form the control algorithm) on an FPGA will require a trade-off between the number of operations per clock cycle (the execution speed) and amount of FPGA resources used.

The algorithm must be designed such that the control algorithm can execute within the chosen sampling interval, while also being resource efficient in the sense that an increase in the size of the algorithm results only in a modest increase in the amount of FPGA resources used.

To illustrate typical design considerations for an algorithm, consider a matrix-vector multiplication, $\boldsymbol{p} = \boldsymbol{M}\boldsymbol{z}$, with $\boldsymbol{M}$ being a 7×4 matrix. The given matrix-vector multiplication can be seen as 7 dot products, with each dot product containing 4 multiplications.[3] Consider the following three design options for the matrix-vector multiplication.

1) The first and computationally fastest option is to use 7×4 multipliers, which will be able to calculate the entire matrix-vector multiplication in a single clock cycle. However, increasing the dimensions of the matrix (and therefore the algorithm size) will significantly increase the resources required.
2) The second and lowest resource usage implementation would be to use only a single multiplier for all of the required multiplications. This approach would require 7×4 clock cycles to calculate the vector-matrix multiplication. Increasing the dimension of the matrix would rapidly increase the number of clock cycles required to compute the matrix-vector multiplication.
3) The final and most balanced option is to use 4 multipliers. This allows calculating a single dot product in one clock cycle, and the matrix-vector multiplication would thus require 7 clock cycles. The final implementation option is a good compromise between resource usage and computation speed; the required resources and clock cycles scale well with an increase in problem size.

Fig. 4 illustrates how option 3 can be implemented in hardware. At each clock cycle, the algorithm computes the ith entry of the vector $\boldsymbol{p}$, denoted p_i, with $i = 1, 2, \ldots, 7$.. The multiplexer selects the ith column of matrix $\boldsymbol{M}$, denoted by $\boldsymbol{M}_{(i,:)}$, that is used by the so-called *Dot Product Unit*.

B. Unconstrained Optimum

The first step towards solving the ILS problem is to calculate the transformed unconstrained optimum of (20), which is based on (13) and (17).

First consider (13). Its second term simplifies to

$$\boldsymbol{Z} := \boldsymbol{S}^T \boldsymbol{E}\boldsymbol{u}(k-1) = \begin{bmatrix} \boldsymbol{u}^T(k-1) & \boldsymbol{0}_{1\times 3(N_p-1)} \end{bmatrix}^T \in \mathbb{R}^{3N_p}. \tag{23}$$

Note that $\boldsymbol{\Theta}(k)$ is a column vector of dimension $3N_p$. Its jth entry Θ_j is given by

$$\begin{aligned} \Theta_j =& \Upsilon_{(1,j)}(\Gamma_{(1,1)} i_1 + \Gamma_{(1,2)} i_2 - Y_{ref,1}) + \cdots \\ &+ \Upsilon_{(2N_p,j)}(\Gamma_{(2N_p,1)} i_1 + \Gamma_{(2N_p,2)} i_2 - Y_{ref,2N_p}) - \lambda_u Z_j, \end{aligned} \tag{24}$$

where $j = 1, 2, \ldots, 3N_p$. To achieve a resource efficient implementation, as mentioned in Section IV-A, only one entry of $\boldsymbol{\Theta}(k)$ is calculated per clock cycle. The same logic circuit is reused repeatedly and the coefficients are changed at every

[3]In terms of resource usage, we will only consider the effect of multiplications (and not additions). Additions can be implemented using cheap logic elements, while multiplications usually require embedded multipliers on FPGAs.

Algorithm 1 Unconstrained optimum

function $[\boldsymbol{U}_{unc}, \bar{\boldsymbol{U}}_{unc}] = \textsc{UncOpt}(\boldsymbol{i})$
  for $j = 1$ to $3N_p$ **do**
    $\Theta_j = \Upsilon_{(1,j)}(\Gamma_{(1,1)} i_1 + \Gamma_{(1,2)} i_2 - Y_{ref,1}) + \cdots$ ▷ Eq. (24)
  end for
  for $q = 1$ to 2 **do**
    if $q = 1$ **then**
      $\boldsymbol{M} = -\boldsymbol{H}^{-1}$
    else if $q = 2$ **then**
      $\boldsymbol{M} = -\boldsymbol{V}\boldsymbol{H}^{-1}$
    end if
    for $j = 1$ to $3N_p$ **do**
      $u_j = M_{(j,1)}\Theta_1 + \cdots + M_{(j,3N_p)}\Theta_{3N_p}$ ▷ Eqs. (25) and (26)
    end for
    if $q = 1$ **then**
      $\boldsymbol{U}_{unc} = \boldsymbol{U}$
    else if $q = 2$ **then**
      $\bar{\boldsymbol{U}}_{unc} = \boldsymbol{U}$
    end if
  end for
end function

Algorithm 2 Initial radius

function $\rho^2_{ini} = \textsc{IniRad}(\bar{\boldsymbol{U}}_{\boldsymbol{unc}}, \bar{\boldsymbol{U}}_{\boldsymbol{bab}}, \bar{\boldsymbol{U}}_{\boldsymbol{ed}})$
  for $q = 1$ to 2 **do**
    $\rho^2 = 0$
    if $q = 1$ **then**
      $\boldsymbol{U}_{ini} = \boldsymbol{U}_{bab}$
    else if $q = 2$ **then**
      $\boldsymbol{U}_{ini} = \boldsymbol{U}_{ed}$
    end if
    for $j = 1$ to $3N_p$ **do**
      $\rho^2 = (\bar{u}_{unc,j} - V_{(j,1)} u_{ini,1} - \cdots)^2 + \rho^2$ ▷ Eq. (29)
    end for
    if $q = 1$ **then**
      $\rho^2_{bab} = \rho^2$
    else if $q = 2$ **then**
      $\rho^2_{ed} = \rho^2$
    end if
  end for
  if $\rho^2_{ed} \leq \rho^2_{bab}$ **then**
    $\rho^2_{ini} = \rho^2_{ed}$
  else
    $\rho^2_{ini} = \rho^2_{bab}$
  end if
end function

clock cycle.[4]

Similarly, the jth entry $u_{unc,j}$ of $\boldsymbol{U}_{unc}(k)$ in (17) is given by

$$u_{unc,j} = -H^{-1}_{(j,1)}\Theta_1 - H^{-1}_{(j,2)}\Theta_2 - \cdots - H^{-1}_{(j,3N_p)}\Theta_{3N_p}. \quad (25)$$

The jth entry $\bar{u}_{unc,j}$ of $\bar{\boldsymbol{U}}_{unc}(k)$ in (20) can be calculated as[5]

$$\bar{u}_{unc,j} = -\left(VH^{-1}\right)_{(j,1)}\Theta_1 - \cdots - \left(VH^{-1}\right)_{(j,3N_p)}\Theta_{3N_p}. \quad (26)$$

As (25) and (26) have the same algebraic structure and only differ regarding the coefficients, the hardware multipliers and adders used for (25) can be reused for (26). The calculation of the unconstrained optimum in the FPGA is shown in Algorithm 1. Note that the Babai estimate is calculated at the same time as the entries of the unconstrained optimum are calculated.

C. Initial Radius

Both the Babai estimate $\boldsymbol{U}_{bab}$ and the educated guess $\boldsymbol{U}_{ed}$ are considered when determining the initial radius ρ_{ini} of the sphere (see Section III-B for details on the two initial solutions). Defining the radii

$$\begin{aligned} \rho_{bab}(k) &= \left\|\bar{\boldsymbol{U}}_{unc}(k) - \boldsymbol{V}\boldsymbol{U}_{bab}(k)\right\|_2 \text{ and} \\ \rho_{ed}(k) &= \left\|\bar{\boldsymbol{U}}_{unc}(k) - \boldsymbol{V}\boldsymbol{U}_{ed}(k)\right\|_2, \end{aligned} \quad (27)$$

the initial radius is taken as the minimum of the two corresponding radii, i.e.,

$$\rho_{ini}(k) = \min\{\rho_{bab}(k), \rho_{ed}(k)\}. \quad (28)$$

During steady-state operation, the educated guess proves to be a good initial solution, whereas the Babai estimate is better during transients.

It can be seen from (22) that the (squared) radius is the sum of the squared terms from $j = 1$ to $3N_p$, i.e.,

$$\rho^2 = \sum_{j=1}^{3N_p} (\bar{u}_{unc,j} - V_{(j,1)}u_{ini,1} - \cdots - V_{(j,3N_p)}u_{ini,3N_p})^2, \quad (29)$$

where $\boldsymbol{U}_{ini} \in \{\boldsymbol{U}_{bab}, \boldsymbol{U}_{ed}\}$. By following the same principle of reusing multipliers, only one squared term is calculated per clock cycle, as stated in Algorithm 2.

D. Non-Recursive Sphere Decoding Algorithm

The SDA proposed in [5] is very efficient, albeit recursive. In high-level programming languages, such as C/C++, the SDA can be implemented with the help of a stack. Implementing recursion in VHDL, however, requires the sphere decoder to be synthesized $3N_p$-times. This would use an unacceptable amount of resources, and thus recursion should be avoided.

An alternative SDA was proposed in [15], which is shown in Algorithm 3 and explained hereafter. Lines 2–5 initialize

[4]Note that (24) is the only equation that will require modification if the system is changed (assuming the system has three phases). All the equations hereafter remain the same regardless of the application, for instance when an electric machine or a grid-connected converter is considered instead of an RL load.

[5]If the Babai estimate is not used, only the transformed unconstrained optimum $\bar{\boldsymbol{U}}_{unc}(k)$ is required.

Algorithm 3 Non-Recursive Sphere Decoder

```
1:  function U_opt = SphDec(ρ², Ū_unc)
2:      j = 1
3:      optimal_solution_found = false
4:      set each element in b to −1
5:      set each element in d² to 0
6:      while optimal_solution_found = false do
7:          u_j = b_j
8:          d'² = (ū_unc,j − V_(j,1:j) U_1:j)² + d_j²      ▷ Eq. (22)
9:          if d'² ≤ ρ² then                               ▷ Eq. (21)
10:             if j = 3N_p then
11:                 U_opt = U
12:                 ρ² = d'²
13:                 b_j ++
14:             else
15:                 j ++
16:                 d_j² = d'²
17:             end if
18:         else
19:             b_j ++
20:         end if
21:         for q = 3N_p downto 2 do
22:             if b_q > 1 then
23:                 b_q = −1
24:                 j = q − 1
25:                 b_j ++
26:             end if
27:         end for
28:         if b_1 > 1 then
29:             optimal_solution_found = true
30:         end if
31:     end while
32: end function
```

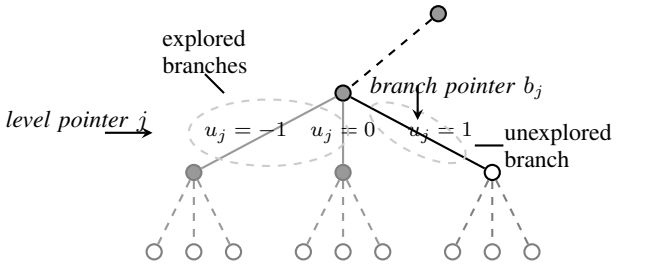
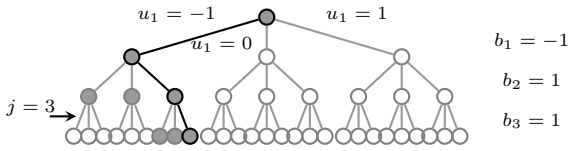
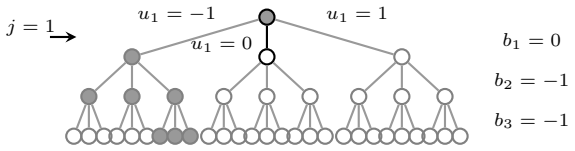


Fig. 5: Illustration of a level in a search tree being explored. Nodes that are shaded have been visited by the algorithm, whereas unshaded nodes are unvisited.

the algorithm. The SDA can be visualised traversing a search tree with depth $3N_p$. Fig. 5, which will be used to explain the algorithm, illustrates an extract of a level in a search tree. The algorithm assembles the switching sequence $\boldsymbol{U}(k)$ entry by entry, where the jth level of the search tree corresponds to the entry u_j. The admissible single-phase switch positions $\mathcal{U} = \{-1, 0, 1\}$ are considered at each level and are represented by the branches of the search tree. The algorithm uses two different pointers to keep track of the explored branches. The first pointer is the *level pointer*, denoted by $j = 1, 2, \ldots, 3N_p$,



(a) The sphere decoder is currently at a leaf node, where the switching sequence is $\boldsymbol{U}(k) = [-1\ 1\ 1]^T$. Once the evaluation has been finished, the bottom branch pointer is incremented to $b_3 = 2$ ($\notin \mathcal{U}$) and backtracking starts.



(b) The bottom branch pointer is now set back to $b_3 = -1$ and the current level is decreased to $j = 2$. Next, the branch pointer at the second level is incremented to $b_2 = 2$ ($\notin \mathcal{U}$). The sphere decoder sets $b_2 = -1$ and backtracks to the top level, where the branch pointer is incremented to $b_1 = 0$. Backtracking is finished and the new (partial) switching sequence is $\boldsymbol{U}(k) = [0]$.

Fig. 6: Illustration of backtracking for the non-recursive sphere decoder. The black lines represent the branches of the search tree that are currently explored.

which refers to the level of the search tree that is currently explored (i.e., the jth entry of $\boldsymbol{U}(k)$). The second pointer is the *branch pointer*, denoted by $b_j = -1, 0, 1$, which refers to the next branch to be explored at level j of the search tree (i.e., the switch position to be evaluated at the jth entry of $\boldsymbol{U}(k)$; see line 7). There are $3N_p$ such pointers stored in the array $\boldsymbol{b}$.

When a branch is explored, a node is visited where the (intermediate) squared radius d'^2 is calculated with (22) in line 8 (shaded nodes in Fig. 5 represent the nodes where d'^2 has been calculated). If d'^2 exceeds the radius of the sphere (the condition in line 9 is not satisfied), the sub branches attached to the particular node are, by definition, suboptimal and can be pruned. After pruning the branch from the search tree, thereby reducing the number of possible solutions that have to be considered, the branch pointer is updated to refer to the next (adjacent) branch (line 19). If there is no adjacent branch left on the particular node to be explored, the sphere decoder backtracks (i.e., moves up one level in the search tree). The backtracking procedure will be explained later. In the case that the solution that is being built is still contained within the sphere (the condition in line 9 is satisfied), we increment the level pointer and move further down the tree (line 15). The squared terms of the ILS problem in line 8 are stored in the array $\boldsymbol{d}^2$, where d_j^2 in line 16 is the accumulated squared term up to (not including) the jth level.

Once a bottom node has been reached (implying the condition in line 10 has been satisfied), known as a *leaf* node, a full switching sequence has been assembled. At the leaf node, the incumbent optimal solution is updated and the sphere is tightened (lines 11 and 12). The branch pointer is also incremented (line 13), since the next admissible switch position on the leaf node will be investigated. Note that a certificate for optimality (i.e., the incumbent optimal solution is indeed optimal) is only obtained once all the possible branches have been investigated. This happens if the top-level branch pointer exceeds $b_1 > 1$ in line 28. The sphere decoder then terminates, and a certificate for optimality has been obtained.

Backtracking is achieved by evaluating $3N_p - 1$ *if-statements* (lines 21–26). Backtracking works as follows: if a branch pointer exceeds 1, which means that the switch position does not belong to the single-phase set $\mathcal{U}$, the branch pointer is set equal to -1. The current level is decreased by one and the next branch of the node is evaluated. This procedure is done from bottom to top. Fig. 6 illustrates and explains the backtracking procedure.

All of the lines in Algorithm 3 are executed within one clock cycle. This means that for each clock cycle one node of the search tree is visited.

V. Computational Burden and FPGA Resource Usage

The controller was directly written in VHDL and implemented on an Altera-Intel Cyclone V 5CSEMA5F31C6N FPGA.[6] This low-cost FPGA has 32070 adaptive logic modules (ALMs) and 87 variable-precision digital signal processing (DSP) blocks. Following a timing analysis[7], the lower bound and upper bound on the maximum clock frequency, at which the FPGA can operate reliably, is $16.1\,\mathrm{MHz}$ and $31.1\,\mathrm{MHz}$, respectively. A conservative approach is taken and the clock frequency is set to $15\,\mathrm{MHz}$.

A. Off-Line Calculations

All matrices are calculated in advance and stored in the FPGA in arrays. These include the prediction matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{\Upsilon}$. The Cholesky decomposition of $\boldsymbol{H}$ is calculated via MATLAB in order to obtain the generator matrix $\boldsymbol{V}$. The multiplication of $\boldsymbol{H}^{-1}$ and $\boldsymbol{V}$ is also done offline. Note that the matrices $\boldsymbol{S}$ and $\boldsymbol{E}$ are not required for practical implementation (see (24)). The reference currents over one fundamental period are also calculated off-line and stored in arrays.

It is important to note that when the weighting factor λ_u is changed, $\boldsymbol{H}$ and $\boldsymbol{V}$ have to be recalculated. This poses a problem as the switching frequency, by tuning λ_u, cannot be adjusted online without recalculating the required matrices. When a reference change is given, the switching frequency could change to an undesired frequency. This can be overcome in two possible ways. For multiple λ_u values, multiple $\boldsymbol{H}$ and $\boldsymbol{V}$ matrices can be calculated offline and stored on the FPGA in lookup tables. Another approach would be to calculate the matrices online. It is important to note that it is not necessary to calculate the required matrices within one sampling interval

[6]Note that the FPGA includes an ARM Cortex processor, which was not utilized for the implementation of the controller.

[7]The timing analysis tool does an analysis that is subjected to a variety of conditions. This includes temperature, voltage, and manufacturing process conditions.

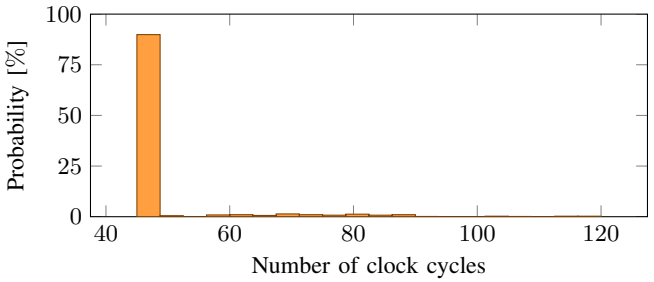


Fig. 7: Histogram of the number of clock cycles used by the non-recursive sphere decoder for a horizon $N_p = 5$.

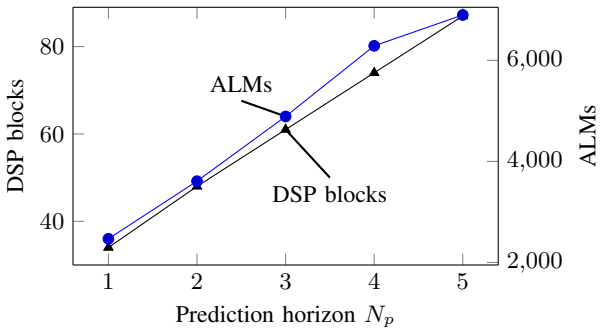


Fig. 8: Resource usage of the controller implementation. The FPGA has a total of 87 DSP blocks and 32070 ALMs.

before the control algorithm is executed; multiple sampling intervals can be used to calculate the matrices. The new matrices can be calculated while running the controller by either utilizing the ARM core (if available) or a low-resource implementation on the FPGA itself. The current matrices, from the previous weighting factor, are used by the controller until the new matrices are available.

B. On-line calculations

The number of clock cycles available for on-line calculations is given by $T_s f_{clk}$, where f_{clk} is the clock frequency of the control algorithm. To guarantee optimality of the solution, the sphere decoder must finalize its calculations within the allocated number of clock cycles.

1) Pre-processing calculations: Irrespective of the sampling interval and clock frequency, calculations for the unconstrained optimum and initial radius are limited to one term per clock cycle (see Sections IV-B and IV-C).

Five clock cycles are used to read in the sampled $\alpha\beta$-currents. This includes processing and converting the ADC value, applying the Clarke transformation, and compensating for the measurement and calculation delays (see Section 4.2.8 in [3] for more details). A total of $5 \times 3N_p$ clock cycles is required to calculate the unconstrained optimum $\boldsymbol{U}_{unc}(k)$, the transformed unconstrained solution $\bar{\boldsymbol{U}}_{unc}(k)$, and the squared radii $\rho^2_{ed}(k)$ and $\rho^2_{bab}(k)$. One clock cycle is used to determine the initial radius $\rho^2_{ini}(k)$, and another cycle is used to send $\bar{\boldsymbol{U}}_{unc}(k)$ and $\rho^2_{ini}(k)$ to the SDA. In total, $7 + 5 \times 3N_p$ clock cycles are required before the sphere decoder starts.

2) Sphere Decoding: The remaining clock cycles after the pre-processing calculations are available for sphere decoding. To evaluate the number of clock cycles required for sphere decoding, the sphere decoder that is implemented on the FPGA is given typical unconstrained optima and initial radii of the ILS problem during steady-state conditions. The sampling interval is set to $T_s = 25\,\mu\text{s}$ to demonstrate that the optimization problem can be solved within such a short time.

Fig. 7 shows the histogram of the number of clock cycles required for sphere decoding within one fundamental period during steady-state operation. The data are captured from the FPGA for the horizon $N_p = 5$. In $89.5\,\%$ of the occurrences, the sphere decoder requires only 45 clock cycles to issue a certificate of optimality. This translates to only $3\,\mu\text{s}$. The calculations during the first 82 clock cycles (for pre-processing calculations) require $5.4\,\mu\text{s}$. Thus, in $89.5\,\%$ of the occurrences, the controller requires only $8.4\,\mu\text{s}$ to find the optimal switch positions — well below $25\,\mu\text{s}$. The maximum number of clock cycles the sphere decoder required to solve the optimization problem was 120 cycles. This translates to $13.4\,\mu\text{s}$ in total (including the pre-processing calculations), which is still well within the chosen sampling interval.

C. Resource Usage

The resource usage of the FPGA for prediction horizons $N_p = 1$ to 5 is shown in Fig. 8. Increasing the horizon leads to a linear increase in resource usage. Fixed-point arithmetic is used due to its speed advantage and low resource usage compared to floating-point arithmetic. All of the variables are of signed-fixed-point type, and the bit assignment is shown in Table I (note that the bit assignment for matrices and vectors refers to each entry). The variable sizing is done according to MATLAB simulations with various weighting factors λ_u. The integer part is chosen in order to avoid truncation, which is important when transient conditions are of concern. The fractional part is chosen such that sufficient numerical accuracy is achieved. Extra bits are added to both the integer and fractional parts for additional headroom. The limiting factor for the prediction horizon was the number of available DSP blocks, as $100\,\%$ are used when $N_p = 5$, while only $21\,\%$ of the ALMs are used for the same case.

Finally, it is worthwhile to mention that the Cyclone 10 series has FPGAs that offer up to 288 DSP blocks, i.e., more than three times the DSP blocks of the Cyclone V used in this work. Since, according to Fig. 8, the DSP block usage increases linearly with the prediction horizon length, it can be deduced that a horizon of at least $N_p = 15$ could be implemented on the largest Cyclone 10 FPGA.

VI. Experimental Results

A block diagram of the experimental setup is shown in Fig. 9. The flow of the control algorithm, which is clocked at $15\,\text{MHz}$, is illustrated.

The experimental setup consists of the previously mentioned Cyclone V 5CSEMA5F31C6N FPGA, an Infineon F3L030E07 evaluation board with an F3L75R07W2E3 three-level IGBT module, an RL load, two Allegro ACS714 Hall effect current

TABLE I: Bit assignment for signed fixed-point variables

Variable	Integer/fractional bits
$\boldsymbol{x}$	5/20
$\boldsymbol{\Gamma}$, $\boldsymbol{\Upsilon}$, $\boldsymbol{H}$, $\boldsymbol{V}$, $\boldsymbol{\Theta}$, $\boldsymbol{U}_{unc}$, $\bar{\boldsymbol{U}}_{unc}$	6/17
ρ^2, $\boldsymbol{d}'^2$	11/22

TABLE II: System parameters

Parameter	Description	Value
V_d	Bus voltage	100 V
R	Load resistor	3.5 Ω
L	Load inductor	2 mH
T_s	Sampling interval	100 µs

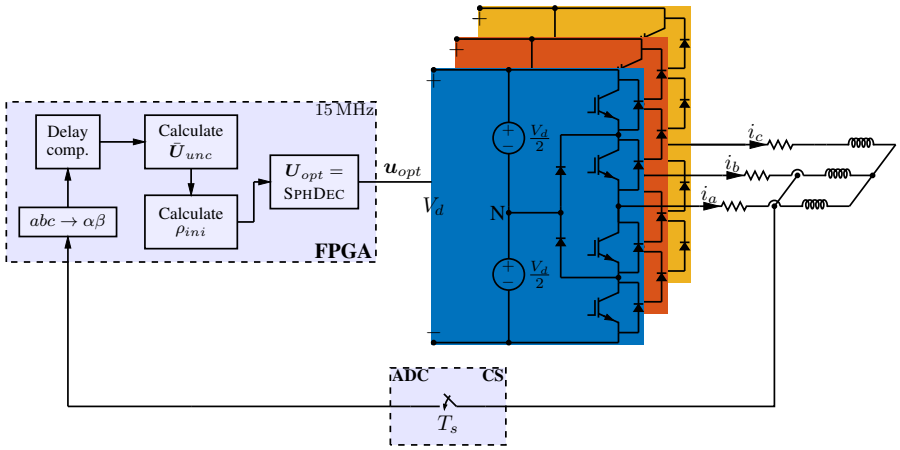


Fig. 9: Block diagram of the experimental setup.

sensors (CS) and a Texas Instruments ADS7864 ADC. The system parameters are shown in Table II. Note that the analysis of computational burden of the sphere decoder in Section V-B2 was done at a sampling interval of $T_s = 25\,\mu\text{s}$, but for the experimental results a sampling interval of $T_s = 100\,\mu\text{s}$ is used. This it due to the observation that longer sampling intervals are better suited to low switching frequencies [6]. The bus voltage is realized by two power supplies; one across each of the top and bottom capacitors. The power supplies ensure that the neutral point potential has minimum fluctuations, which were assumed to be zero in Section II-A.

A. Steady-State Operation

In Fig. 10, the current total harmonic distortion (THD) for the switching frequencies $f_{sw} = 125\,\text{Hz}$ to $450\,\text{Hz}$ (which are widely used in medium-voltage applications) and for prediction horizons $N_p = 1$, 3, and 5 are shown. Solid circles represent measured values. Appropriate trend lines are plotted. In total, more than 500 tests were performed. Unless otherwise stated, all experiments were done with a sinusoidal reference current of 8 A peak and a frequency of 50 Hz. Note that the THD calculations include all harmonics and not just those at integer multiples of the fundamental component. It can be seen that, in general, longer horizons tend to reduce the current THD. The advantage that longer horizons have over $N_p = 1$ is pronounced between $f_{sw} = 225\,\text{Hz}$ and $400\,\text{Hz}$. Concerning the long horizons, $N_p = 5$ outperforms $N_p = 3$ noticeably at some frequencies, such as between 125 Hz and 175 Hz and also between 250 Hz and 300 Hz. At a switching frequency of 250 Hz, the current THD is decreased from 8.61 % to 8.27 % with a horizon of $N_p = 3$ instead of $N_p = 1$. The current THD is further deceased to 7.94 % if a horizon of $N_p = 5$ is used.

Even though these experimental results might indicate that longer horizons do not always reduce the current distortion for a given switching frequency, the reader is reminded that only the weighting factor λ_u was adjusted to obtain the desired switching frequency. It is shown in [6] that the sampling interval also influences the system performance. This is due the fact that T_s, along with N_p, determines the length of the horizon in time, $N_p T_s$. With an appropriate selection of both T_s and λ_u, longer horizons should always offer improved performance; see Fig. 11 in [6] for Monte Carlo simulations. Moreover, the advantages of long horizons are much more pronounced for higher-order systems; see [24] for a quasi-Z-source inverter and [7] for a three-level inverter with an LC filter and an induction machine.

The phase currents, switch positions and current spectra for $N_p = 5$ at $f_{sw} = 250\,\text{Hz}$ with a 9 A peak reference are shown in Fig. 11. Due to the non-periodic switch positions, the current spectra exhibit noticeable inter-harmonics, see Fig. 11(c). However, there is some form of periodicity and the harmonic energy is mostly concentrated at integer multiples of the fundamental.

B. Transient Response

A major benefit of direct MPC is its fast response during transients. Solving the optimization problem during transients requires a greater computational effort than in steady-state operation because the initial radius of the sphere tends to be larger, i.e., it includes a higher number of nodes (candidate solutions). Therefore, practical evaluation is conducted with a horizon of $N_p = 5$, as this horizon has the highest computational burden of all the implemented horizons. If the optimal

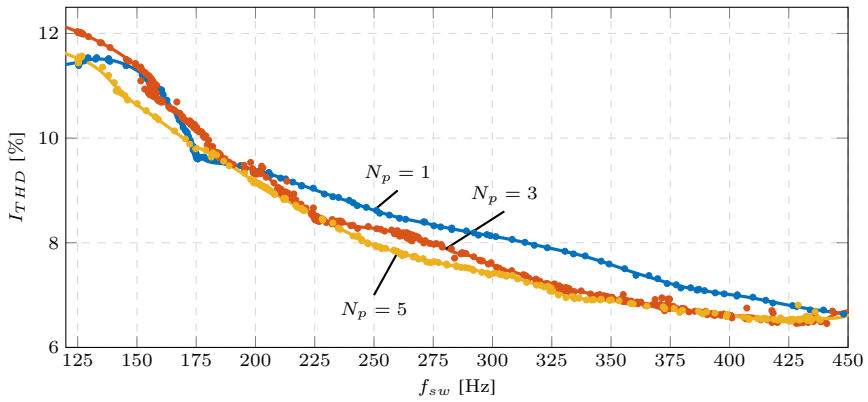
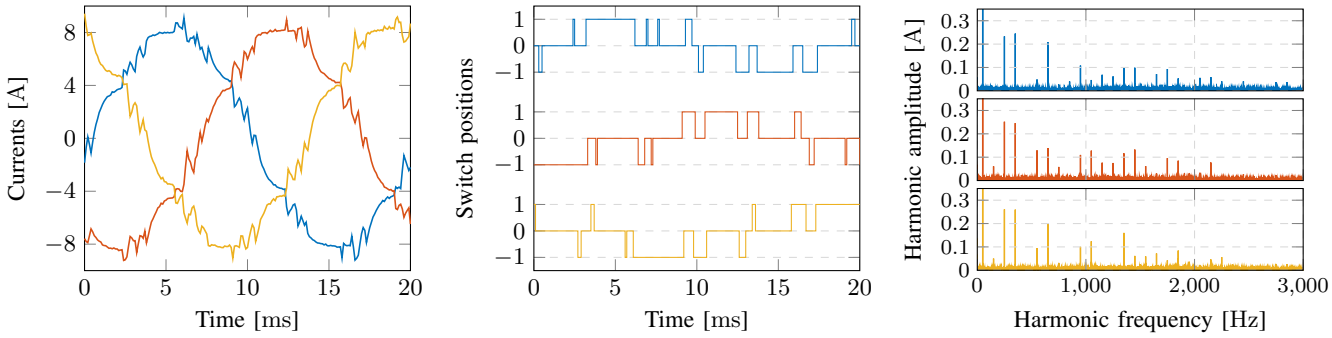
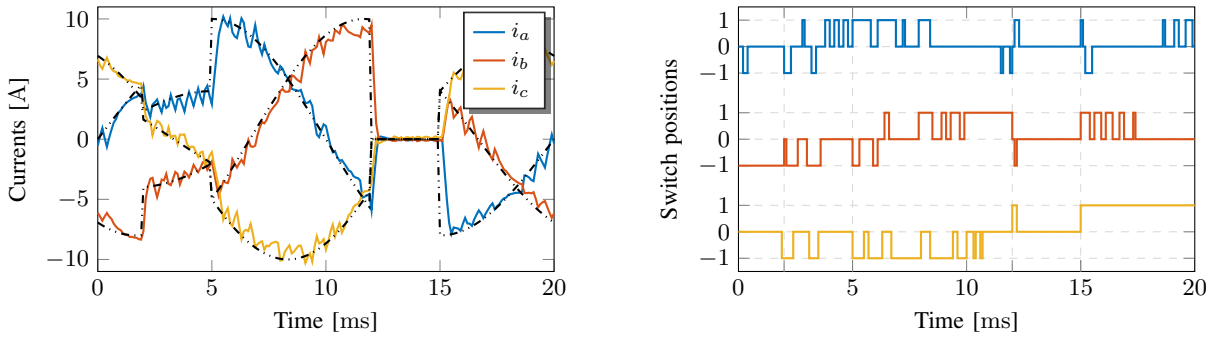


Fig. 10: Experimental results of the current THD vs the switching frequency for the horizons $N_p = 1$, 3, and 5.



(a) Phase currents (b) Switch positions (c) Current spectra

Fig. 11: Experimental results during steady-state operation with the horizon $N_p = 5$ and the switching frequency $f_{sw} = 250\,\text{Hz}$. The switch positions of phase a are at the top, and phases b and c are shown below.



(a) Current waveforms. The phase references are shown by the dash dotted lines

(b) Switch positions, with phase a at the top, and phases b and c below

Fig. 12: Experimental phase currents and switch positions.

solution is not found during a transient and a suboptimal switch position was applied, the fast response of MPC could be inhibited.

In Fig. 12, the current is stepped from $8\,\text{A}$ to $4\,\text{A}$, then to $10\,\text{A}$, then to $0\,\text{A}$ and then back to $8\,\text{A}$. The corresponding switch positions are shown in Fig. 12(b). The reference step is imposed without prior knowledge and thus cannot be anticipated by the controller. The negative reference steps have a settling time of approximately $0.2\,\text{ms}$ and $0.3\,\text{ms}$, respectively, while both the positive steps have a settling time of approximately $0.5\,\text{ms}$. The step-down cases are faster since the voltage margin across the inductor is larger due to the current through the load.

The maximum number of clock cycles required to find the optimal solution during the transients is 1030 out of the available 1500 clock cycles. This includes the calculation of the unconstrained optimum and initial radii. For more aggressive reference steps to larger currents or if shorter sampling intervals were used, there is no guarantee that the optimal solution would be found within the sampling interval. One could then resort to suboptimal solutions, as proposed in [25].

Recently, efforts have been made to alleviate the computational burden during transients [17], [26]. The idea is to project the unconstrained optimum, which is located far outside the convex hull of the lattice during transients and results in a large initial radius, onto the convex hull of the lattice. The projected point on the lattice then becomes the new unconstrained optimum. This leads to a much smaller initial radius and thus to a significantly reduced computation time required to solve the sphere decoding problem.

VII. Conclusions

In this paper, it was shown that long-horizon FCS-MPC can be implemented on a low-cost FPGA. To this end, a non-recursive variant of the sphere decoding algorithm, which avoids the recursion with the help of two pointer arrays, was developed. By reusing multipliers and exploiting the problem structure at hand, a resource-efficient implementation was achieved. More specifically, for a prediction horizon of five steps, 87 DSP blocks and a maximum execution time of $13.4\,\mu\text{s}$ was required to solve the optimization problem during steady-state operation. A sampling interval of $25\,\mu\text{s}$ could thus be easily achieved.

It was verified that sphere decoding is highly efficient at solving the complex optimization problem underlying long-horizon FCS-MPC in real time. Given the more powerful FPGA platform's that exist today, it is expected that horizons up to $N_p = 15$ can be achieved in real time with a short sampling interval of $T_s = 25\,\mu\text{s}$.

Finally, it was experimentally verified that long horizons offer a performance benefit. The practical implementation of FCS-MPC with long horizons for more complex systems remains a promising topic for future research.

Appendix

The prediction and auxiliary matrices are defined as

$$\boldsymbol{\Gamma} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^{N_p} \end{bmatrix}, \boldsymbol{\Upsilon} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{0}_{2\times 3} & \cdots & \boldsymbol{0}_{2\times 3} \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \cdots & \boldsymbol{0}_{2\times 3} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N_p-1}\boldsymbol{B} & \boldsymbol{A}^{N_p-2}\boldsymbol{B} & \cdots & \boldsymbol{B} \end{bmatrix},$$

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_{3\times 3} & \cdots & \boldsymbol{0}_{3\times 3} \\ -\boldsymbol{I}_3 & \boldsymbol{I}_3 & \cdots & \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{3\times 3} & -\boldsymbol{I}_3 & \cdots & \boldsymbol{0}_{3\times 3} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{0}_{3\times 3} & \cdots & \boldsymbol{I}_3 \end{bmatrix} \text{ and } \boldsymbol{E} = \begin{bmatrix} \boldsymbol{I}_3 \\ \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{3\times 3} \\ \vdots \\ \boldsymbol{0}_{3\times 3} \end{bmatrix},$$

where $\boldsymbol{0}_{n\times m}$ is an $n \times m$ zero matrix.

References

[1] P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez, "Predictive Control in Power Electronics and Drives," *IEEE Transactions on Industrial Electronics*, vol. 55, pp. 4312–4324, Dec. 2008.

[2] D. E. Quevedo, R. P. Aguilera, and T. Geyer, "Predictive Control in Power Electronics and Drives: Basic Concepts, Theory, and Methods," in *Advanced and Intelligent Control in Power Electronics and Drives* (T. Or\lowska-Kowalska, F. Blaabjerg, and J. Rodríguez, eds.), pp. 181–226, Cham: Springer International Publishing, 2014.

[3] T. Geyer, *Model Predictive Control of High Power Convertersand Industrial Drives*. Chichester, UK: John Wiley & Sons, Ltd, Nov. 2016.

[4] J. Rodríguez, J. Pontt, C. Silva, M. Salgado, S. Rees, U. Ammann, P. Lezana, R. Huerta, and P. Cortés, "Predictive control of three-phase inverter," *IET Electronics Letters*, vol. 40, no. 9, p. 561, 2004.

[5] T. Geyer and D. E. Quevedo, "Multistep Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics*, vol. 29, pp. 6836–6846, Dec. 2014.

[6] T. Geyer and D. E. Quevedo, "Performance of Multistep Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics*, vol. 30, pp. 1633–1644, Mar. 2015.

[7] T. Geyer, P. Karamanakos, and R. Kennel, "On the benefit of long-horizon direct model predictive control for drives with LC filters," in *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, (Pittsburgh, PA, USA), pp. 3520–3527, IEEE, Sept. 2014.

[8] B. Stellato, T. Geyer, and P. J. Goulart, "High-Speed Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics*, vol. 32, pp. 4007–4020, May 2017.

[9] J. Raath, D. T. Mouton, and T. Geyer, "Integration of inverter constraints in geometrical quantification of the optimal solution to an MPC controller," in *2016 IEEE 17th Workshop on Control and Modeling for Power Electronics (COMPEL)*, (Trondheim, Norway), pp. 1–6, IEEE, June 2016.

[10] J. Raath, T. Geyer, and H. d. T. Mouton, "Iterative slicing as solution to the model predictive control problem of a three-level inverter," *in Proceedings of the 25th South African Universities Power Engineering conference (SAUPEC)*, Feb. 2017.

[11] J. Raath, T. Mouton, and T. Geyer, "On the micciancio-voulgaris algorithm to solve the long-horizon direct MPC optimization problem," in *2017 IEEE International Symposium on Predictive Control of Electrical Drives and Power Electronics (PRECEDE)*, (Pilsen, Czech Republic), pp. 95–100, IEEE, Sept. 2017.

[12] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Transactions on Signal Processing*, vol. 53, pp. 2806–2818, Aug. 2005.

[13] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of Computation*, vol. 44, pp. 463–463, May 1985.

[14] P. Karamanakos, T. Geyer, and R. Kennel, "A Computationally Efficient Model Predictive Control Strategy for Linear Systems With Integer Inputs," *IEEE Transactions on Control Systems Technology*, vol. 24, pp. 1463–1471, July 2016.

[15] M. Dorfling, H. Mouton, P. Karamanakos, and T. Geyer, "Experimental evaluation of sphere decoding for long-horizon direct model predictive control," in *2017 19th European Conference on Power Electronics and Applications (EPE'17 ECCE Europe)*, (Warsaw), pp. P.1–P.10, IEEE, Sept. 2017.

[16] R. Baidya, R. P. Aguilera, P. Acuna, S. Vazquez, and H. d. T. Mouton, "Multistep Model Predictive Control for Cascaded H-Bridge Inverters: Formulation and Analysis," *IEEE Transactions on Power Electronics*, vol. 33, pp. 876–886, Jan. 2018.

[17] P. Acuna, C. A. Rojas, R. Baidya, R. P. Aguilera, and J. E. Fletcher, "On the Impact of Transients on Multistep Model Predictive Control for Medium-Voltage Drives," *IEEE Transactions on Power Electronics*, vol. 34, pp. 8342–8355, Sept. 2019.

[18] A. Andersson and T. Thiringer, "Assessment of an Improved Finite Control Set Model Predictive Current Controller for Automotive Propulsion Applications," *IEEE Transactions on Industrial Electronics*, vol. 67, pp. 91–100, Jan. 2020.

[19] G. Papafotiou, T. Geyer, and M. Morari, "A hybrid model predictive control approach to the direct torque control problem of induction motors," *International Journal of Robust and Nonlinear Control*, vol. 17, pp. 1572–1589, Nov. 2007.

[20] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, pp. 407–427, Mar. 1999.

[21] F. Borrelli, M. Baotić, A. Bemporad, and M. Morari, "Dynamic programming for constrained optimal control of discrete-time linear hybrid systems," *Automatica*, vol. 41, pp. 1709–1721, Oct. 2005.

[22] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Addison-Wesley, 3. ed ed., 1997.

[23] L. Babai, "On Lovász' lattice reduction and the nearest lattice point problem," *Combinatorica*, vol. 6, pp. 1–13, Mar. 1986.

[24] A. Ayad, P. Karamanakos, and R. Kennel, "Direct Model Predictive Current Control Strategy of Quasi-Z-Source Inverters," *IEEE Transactions on Power Electronics*, vol. 32, pp. 5786–5801, July 2017.

[25] P. Karamanakos, T. Geyer, and R. Kennel, "Suboptimal search strategies with bounded computational complexity to solve long-horizon direct model predictive control problems," in *2015 IEEE Energy Conversion Congress and Exposition (ECCE)*, (Montreal, QC, Canada), pp. 334–341, IEEE, Sept. 2015.

[26] P. Karamanakos, T. Geyer, and R. P. Aguilera, "Long-Horizon Direct Model Predictive Control: Modified Sphere Decoding for Transient Operation," *IEEE Transactions on Industry Applications*, vol. 54, pp. 6060–6070, Nov. 2018.