Alternative Sphere Decoding Algorithm for Long-horizon Model Predictive Control of Multi-level Inverters

Johan Raath
Department of Electrical, Electronic and Computer Engineering
Central University of Technology
South Africa
Email: jraath@cut.ac.za

Toit Mouton
Department of Electrical and Electronic Engineering
University of Stellenbosch
South Africa
Email: dtmouton@sun.ac.za

Tobias Geyer
ABB Corporate Research
ABB Switzerland Ltd,
Power Electronic Systems
Switzerland
Email: t.geyer@ieee.org

***Abstract*—The computational burden associated with long-horizon finite control set Model Predictive Control remains a challenging problem in power electronic systems with short sampling intervals. In an attempt to address this problem and enable real-time applications, we introduce an alternative Sphere Decoding Algorithm which utilizes matrix block processing and offline preprocessing to support the online optimization process. The expected performance of the algorithm is presented here in the form of MATLAB© simulations together with real-time simulations, run on an OPAL5600© real-time simulator.**

I. Introduction

Long-horizon model predictive control (MPC) evaluates the cost function over a prediction horizon of multiple sampling periods. In power electronic systems, an extended prediction horizon is favoured for its enhanced closed-loop performance during steady-state operation. Under these conditions [1], showed that long prediction horizons lead to significant reductions in current distortion and converter switching frequency. The computational burden associated with long-horizon MPC is troublesome in real-time applications where computations are restricted by the length of the sampling interval. This, in particular, is the case for integer-valued finite control set MPC (FCS-MPC) in which the computational burden increases exponentially with the length of the prediction horizon N. For this reason, FCS-MPC implementations are typically limited to short prediction horizons. Some of the most recent developments with *real-time* validations include [3] and [4] which achieved prediction horizons in the range of $N = 3$ to 5. In three-phase applications, the underlying FCS-MPC problem translates to an integer least squares (ILS) problem of dimension $d = 3N$. Solving such a high dimensional problem with a large number of admissible solutions demands an efficient optimization process or solver. In power electronic systems, the well-known Sphere Decoding Algorithm (SDA) adapted in [5] and its evolution to date, has proven to be very effective in practical implementations. Some major contributions that assist the SDA include the sphere-radius selection strategy of [6] which involves the the "so-called" *Babai estimate*, and the concept of target preconditioning through orthogonal projection of the target onto the search space convex hull [7] and [8]. Target preconditioning, in particular, assists during transient events that occur in the power electronic system.

In an attempt to enhance the computability of the SDA, this paper presents an alternative approach, which proposes the use of block matrix computations. According to [9] is block matrix computation a simple but effective tool for improving algorithm performance in modern processors as it facilitates efficient data handling between main memory and the processing unit. Employing block matrices enables the inclusion of offline preprocessing, which assists in reducing the online computational burden of the decoding algorithm. The proposed algorithm is incorporated into the FCS-MPC of a complex, time-varying system. The power electronic system to be considered is similar to the one presented in [2], where an induction machine is driven by a three-phase, three-level NPC inverter via an intermediate LC-filter. For such a system of higher-order, it was concluded by [10] that long-horizon MPC is particularly beneficial, especially when switching is expensive. Hence the selection, as control of such a system substantiates the use of long-horizon FCS-MPC and the subsequent need for an efficient solver to the optimisation problem.

The paper is organized as follows: Section II introduces a brief overview of the system model. In section III, the altered Sphere Decoding Algorithm is presented, followed by the pseudo-code in section IV. The inherent nature of the algorithm is explored through MATHCAD© simulations in section V. Section VI subjects the algorithm to real-time testing on an OPAL-RT© 5600 platform and section VII concludes the paper.

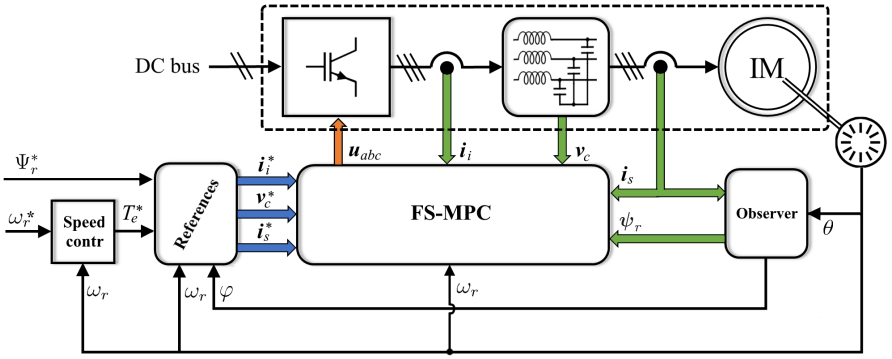

Fig. 1. FCS-MPC of a three-phase three-level NPC inverter driving an induction machine via an intermediate LC filter.

II. System and Modelling

The selected electrical drive system can be defined as a multiple-input multiple-output (MIMO) system managed by an FCS-MPC designed, MIMO controller. Figure 1 depicts the layout. The general cascaded approach of a predictive current control system is followed where the electromagnetic subsystem of the IM is managed by a predictive controller and the mechanical subsystem by a proportional-integral (PI) speed controller. Noteworthy, the following assumptions are made: Firstly, the dc supply V_{DC} is considered stiff with the neutral point of the NPC inverter regulated to the constant value $V_{DC}/2$. Secondly, the rotor speed ω_r is assumed constant for the duration of the MPC prediction horizon.

Modelling of the proposed system and formulation of the MPC problem is done in accordance with [10, p.234]. Each inverter leg $x \in \{a, b, c\}$ of the three-phase NPC inverter can deliver three voltage levels which are represented by the integer values

$$u_x \in \mathcal{U} = \{-1, 0, 1\}. \tag{1}$$

Combined, these states define the inverter control vector

$$\boldsymbol{u}_{abc} = [u_a \ u_b \ u_c]^T \in \boldsymbol{\mathcal{U}} = \mathcal{U}^3. \tag{2}$$

Note that the set $\mathcal{U}$ to the Cartesian power three results in the inverter control vector set $\boldsymbol{\mathcal{U}}$ with a cardinality $\#(\boldsymbol{\mathcal{U}}) = 27$.

The objective of the controller at every sampling instant is to manipulate the state of the inverter legs $\boldsymbol{u}_{abc}$, such that the system outputs, i.e. inverter current $\boldsymbol{i}_i$, capacitor voltage $\boldsymbol{v}_c$ and stator current $\boldsymbol{i}_s$, closely track their respective references (*), while limiting the average inverter switching frequency. System states are obtained at every sampling instant through direct measurement and an observer. The system output references which define the operating point of the induction machine are computed from the rotor flux Ψ_r^* setting and the torque requirement T_e^*. For optimal machine magnetisation, Ψ_r^* is typically held constant at its maximum value but can be decreased if field-weakening is required. To adhere to the control objective stated above, the well-known cost function

$$J = \sum_{l=k}^{k+N-1} \|\boldsymbol{y}^*(l+1) - \boldsymbol{y}(l+1)\|_{\boldsymbol{\Lambda}}^2 + \lambda_u \|\boldsymbol{u}(l) - \boldsymbol{u}(l-1)\|_2^2, \tag{3}$$

featuring reference tracking and switching cost is adopted. The first term in (3) quantifies the tracking error between the system output vector $\boldsymbol{y}$ and the reference in $\boldsymbol{y}^*$ vector. Weighing matrix $\boldsymbol{\Lambda}$, assign weighting factors to the tracking of the output variables. The second term in (3) quantifies the inverter switching cost, with λ_u denoting the switching penalty. The ratio between the weighing matrix $\boldsymbol{\Lambda}$ and switching penalty λ_u establishes a compromise in overall tracking accuracy to the switching cost. Because the cost function (3) is a function of the switching sequence

$$\tilde{\boldsymbol{u}}(k) = [\boldsymbol{u}^T(k) \ \boldsymbol{u}^T(k+1) \ ... \boldsymbol{u}^T(k+N-1)]^T \in \mathbb{Z}^{3N}, \tag{4}$$

it has been shown in [5] that the cost function can be written in terms of the unconstrained minimum

$$\tilde{\boldsymbol{u}}_{unc}(k) = [\boldsymbol{u}_{unc}^T(k) \ \boldsymbol{u}_{unc}^T(k+1) \ ... \boldsymbol{u}_{unc}^T(k+N-1)]^T \in \mathbb{R}^{3N}, \tag{5}$$

to define the optimization problem as an *integer-least-squares* (ILS) problem, i.e.

$$\hat{\boldsymbol{u}}(k) = \underset{\tilde{\boldsymbol{u}}(k) \in \mathbb{U}}{\arg\min} \|\boldsymbol{H}\tilde{\boldsymbol{u}}(k) - \boldsymbol{H}\tilde{\boldsymbol{u}}_{unc}(k)\|_2^2. \tag{6}$$

Matrix $\boldsymbol{H}$ is the Cholesky decomposition[1] of the Hessian matrix, and $\mathbb{U} = \boldsymbol{\mathcal{U}}^N$, the set of admissible switching sequences. Solving the ILS problem (6), geometrically translates to finding from the set of $\boldsymbol{H}\tilde{\boldsymbol{u}}$ vectors the vector with minimum Euclidean distance to the unconstrained solution or *target*

$$\tilde{\boldsymbol{x}} = \boldsymbol{H}\tilde{\boldsymbol{u}}_{unc} \ \in \mathbb{R}^d, \tag{7}$$

in the transformed $\boldsymbol{H}$-coordinate space. This relates to the traditional *nearest neighbour problem* which is also known as the *closest vector problem* (CVP) [11] in lattice theory.

[1]Every real-valued symmetric positive-definite matrix $\boldsymbol{A}$ has a unique *Cholesky decomposition* that results in a unique lower triangular matrix $\boldsymbol{B}$ with strictly real and positive diagonal entries such that $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^T$.

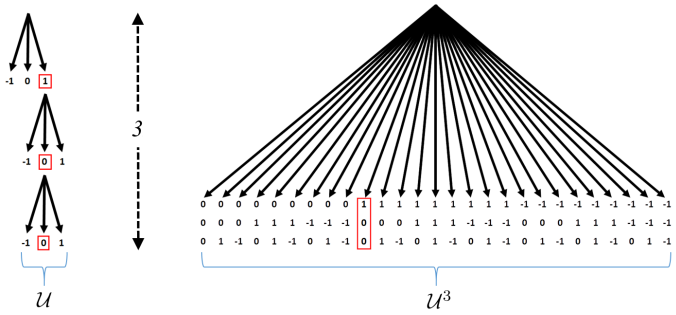

Fig. 2. Search tree with one-dimensional versus three-dimensional branching for a single-step in the prediction horizon.

III. Proposed algorithm

The basic premise in sphere decoding is to limit the number of $\boldsymbol{H}\tilde{\boldsymbol{u}}$ vectors, i.e. lattice points to search over by selecting an initial bound in the form of a sphere centered around the target [12]. Considering the lattice points inside the bounding sphere can be described as a process of pruning a search tree. The conventional approach constitutes a search tree of depth $d = 3N$, with one-dimensional branches corresponding to the elements of the control set $\mathcal{U}$. Pruning of a branch occurs if the cost J of traversal to a higher dimensional node exceeds the square of the bounding sphere radius δ^2. If this is the case, the algorithm steps back and attempts to find another path. The process continues until a path or sequence of d-branching steps is found with minimal cost. The computational complexity of the SDA corresponds to the number of floating-point operations (flops) per node visit, multiplied by the number of nodes visited (algorithm iterations) throughout the search process [12].

A. Algorithm concept

In essence, the proposed strategy is similar to the general sphere decoding approach except that an alternative search tree is considered. The new tree poses three-dimensional branches that correspond to the inverter control vector set $\boldsymbol{\mathcal{U}}$. This in effect reduces the tree depth from $d = 3N$, to a depth equal to the prediction horizon N. Fig. 2 compares the new tree structure with the conventional. The proposed strategy deviates from the conventional approach where at every parent node, a single one-dimensional branching step is evaluated to a process where 27 three-dimensional branches are evaluated simultaneously. The branches that effect a partial cost less than the sphere radius identifies the sibling nodes which are to be considered as parents in the next tree level. In a similar fashion as the conventional SDA, the process follows a depth-first approach and prunes the branches with an excessive cost. The search ends when only one path through the search tree remains, i.e. the optimal solution (6) as a sequence of N, three-dimensional branching steps. To easy further explanation of the algorithm concept, we will drop the time dependency (k), and state the optimal solution (6) as N, three-dimensional sub-vectors

$$\hat{\boldsymbol{u}} = [\hat{\boldsymbol{u}}_1^T \ \hat{\boldsymbol{u}}_2^T ... \hat{\boldsymbol{u}}_N^T]^T \ \in \mathbb{U} \subset \mathbb{Z}^d, \tag{8}$$

which correspond to the search-tree levels, i.e. intermediate horizons

$$n = 1, 2, \ldots, N. \tag{9}$$

B. Cost calculation

As a first observation, the proposed approach of computing at every parent node, the cost of 27 branchings

$$\tilde{\boldsymbol{J}} = [\tilde{J}_1 \ \tilde{J}_2 ... \tilde{J}_{27}] \ \in \mathbb{R}^{27}, \tag{10}$$

seems demanding. However, it can be alleviated by introducing some offline precomputation. This necessitates the representation of the inverter control set $\boldsymbol{\mathcal{U}}$ as the matrix

$$\boldsymbol{U} = [\boldsymbol{u}_1 \ \boldsymbol{u}_2 ... \boldsymbol{u}_{27}] \ \in \mathbb{Z}^{3\times 27}. \tag{11}$$

with the control vectors indexed as the columns of $\boldsymbol{U}$. Transformation of these three-dimensional control vectors to the $\boldsymbol{H}$-coordinate space is accomplished with three-by-three transformation matrices. By definition, the transformation matrix $\boldsymbol{H} \in \mathbb{R}^{d\times d}$ is square and lower triangular. Hence, partitioning it into three-by-three, *non-zero* sub-matrices results in

$$y = \frac{N(N+1)}{2}, \tag{12}$$

sub-matrices, i.e.

$$\boldsymbol{H}_z, \ z = 1, 2..y, \quad \boldsymbol{H}_z \subset \boldsymbol{H}. \tag{13}$$

The numbering of the non-zero sub-matrices in this work is done in a top-to-bottom, and left-to-right manner, as demonstrated for the horizon $N = 3$ case,

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 & 0 & 0 \\ \boldsymbol{H}_2 & \boldsymbol{H}_3 & 0 \\ \boldsymbol{H}_4 & \boldsymbol{H}_5 & \boldsymbol{H}_6 \end{bmatrix}.$$

Note that the rows of sub-matrices in $\boldsymbol{H}$ are congruent to the search-tree levels, i.e. intermediate horizons (9). Also, the sub-matrices that constitute the diagonal of $\boldsymbol{H}$ are identified with

$$b = \frac{n(n+1)}{2}. \tag{14}$$

Each of the sub-matrices $\boldsymbol{H}_z$ transforms the set of control vectors $\boldsymbol{U}$ to corresponding three-dimensional column-vectors of matrices

$$\boldsymbol{G}_z = \boldsymbol{H}_z \boldsymbol{U}, \ \boldsymbol{G}_z \subset \boldsymbol{G}. \tag{15}$$

This allows for the precomputation and storage of the transformed sub-vectors in sub-matrices of the matrix $\boldsymbol{G} \in \mathbb{R}^{3\times 27\times y}$. To express the cost vector (10) in terms of the intermediate horizon n, the target vector is also presented as a sequence of three-dimensional vectors

$$\tilde{\boldsymbol{x}} = [\tilde{\boldsymbol{x}}_1^T \ \tilde{\boldsymbol{x}}_2^T ... \tilde{\boldsymbol{x}}_N^T]^T \ \in \mathbb{R}^d. \tag{16}$$

The 27 costs for traversing the 27 branches from the search-tree *root node* to the *first* tree level ($n = 1$), is obtained from augmenting (6) to give

$$\tilde{\boldsymbol{J}} = \|\boldsymbol{H}_1\boldsymbol{U} - \tilde{\boldsymbol{x}}_n\boldsymbol{o}\|^2 \tag{17a}$$

$$= \|\boldsymbol{G}_1 - \tilde{\boldsymbol{x}}_n\boldsymbol{o}\|^2 . \tag{17b}$$

Note that the vector $\boldsymbol{o} = [1_1, 1_2, \ldots, 1_{27}]^T$ replicates the sub-vector $\tilde{\boldsymbol{x}}_n$ to constitute a matrix for the element-wise subtraction from $\boldsymbol{G}_1$. Traversing to higher tree levels necessitates the introduction of an index vector

$$\boldsymbol{p} = [p_1\ p_2 ... p_N]^T\ \in \mathbb{Z}^N, \tag{18}$$

which records the branching steps per tree level. Each element of $\boldsymbol{p}$ refers to the column index of a selected control vector in $\boldsymbol{U}$. This is to keep track of the path followed and the intermediate cost incurred to a specific node in the search tree. The 27 intermediate branching costs from a node in tree-level *one* to tree-level *two* will therefore resemble

$$\tilde{\boldsymbol{J}} = \left\|(\boldsymbol{H}_3\boldsymbol{U} + \boldsymbol{H}_2\boldsymbol{u}_{p_{n-1}}\boldsymbol{o}) - \tilde{\boldsymbol{x}}_n\boldsymbol{o}\right\|^2 + \tilde{J}_{p_{n-1}} \tag{19a}$$

$$= \left\|\boldsymbol{H}_3\boldsymbol{U} - (\tilde{\boldsymbol{x}}_n - \boldsymbol{H}_2\boldsymbol{u}_{p_{n-1}})\boldsymbol{o}\right\|^2 + \tilde{J}_{p_{n-1}} \tag{19b}$$

$$= \left\|\boldsymbol{G}_3 - (\tilde{\boldsymbol{x}}_n - \boldsymbol{G}_{2,p_{n-1}})\boldsymbol{o}\right\|^2 + \tilde{J}_{p_{n-1}}. \tag{19c}$$

At tree-level ($n = 2$), the term (p_{n-1}) identifies the partial cost $\tilde{J}_{p_{n-1}}$ uncured for traversal to level one via the branching effected by sub-vector $\boldsymbol{u}_{p_{n-1}}$. Congruently, denotes $\boldsymbol{G}_{2,p_{n-1}}$ the p_1^{th} column vector in the matrix $\boldsymbol{G}_2$. From (19c), and with the inclusion of variable b (14), the evolution of the partial cost for traversing to tree levels $n > 1$, can be envisioned as

$$\tilde{\boldsymbol{J}} = \left\|\boldsymbol{G}_b - (\tilde{\boldsymbol{x}}_n - (\boldsymbol{G}_{b-1,p_{n-1}} + \boldsymbol{G}_{b-2,p_{n-2}} + \ldots + \boldsymbol{G}_{b-(n-1),p_1}))\boldsymbol{o}\right.^2 + J_{n-1,p_{n-1}}. \tag{20}$$

The availability of matrix $\boldsymbol{G}$ eliminates the need for online transformation computations, and as a result, reduces the computational effort per node visited. Henceforward, we shall refer to the proposed approach as the Sphere Block Decoding Algorithm (SBDA).

IV. Algorithm pseudo-code

The Sphere Block Decoding Algorithm outputs the optimal control sequence $\hat{\boldsymbol{u}}$ upon input of a target $\tilde{\boldsymbol{x}}$ and the estimated initial sphere radius δ. Also, the SBDA requires the inverter control set $\boldsymbol{U}$ (11) and the precalculated matrix $\boldsymbol{G}$ (15). Algorithm 1 lists the main routine SphBlkDec and Algorithm 2 the subroutine Decode. The main routine initiates the subroutine (line:7), and upon receiving an index vector (18), constructs the optimal control sequence $\hat{\boldsymbol{u}}$ from the set $\boldsymbol{U}$.

The subroutine Decode initiates with the first tree level ($n = 1$), the index vector $\boldsymbol{p} \leftarrow \text{ones}(1, N)$ and a queue-vector $\boldsymbol{q}$, loaded with a single element referencing the root node. Starting at the root necessitates the zeroing of the branching costs $\tilde{\boldsymbol{J}}$. As a first step, the variable b (14) is computed for the current tree level n, followed by the for-loop, which repeats for the number of admissible parent nodes listed in the queue-vector $\boldsymbol{q}$. The n^{th} level sub-vector is extracted from the target vector (line:4), followed by the gathering of the preceding branching cost. This is zero for tree-level ($n = 1$), and updating of the target sub-vector $\boldsymbol{x}$ is also not required (lines:8 to 13). All 27 branching costs are computed in (line:14) and compared with the squared sphere radius (line:15). The resulting queue-vector reflects the index values of admissible branchings which are to be considered as parent nodes for branching to the next tree level. If admissible branchings exist, traversal to the next tree level is considered by re-calling the Decode subroutine (line:21).

Algorithm 1 Sphere Block Decoding Algorithm - main routine

1: **function** SphBlkDec($\tilde{\boldsymbol{x}}, \delta^2, \boldsymbol{G}, \boldsymbol{U}$)
2: $n \leftarrow 1$
3: $\boldsymbol{p} \leftarrow \text{ones}(1, N)$
4: $\boldsymbol{q} \leftarrow [1]$
5: $\tilde{\boldsymbol{J}} \leftarrow \text{zeros}(1, 27)$
6: $\boldsymbol{o} \leftarrow \text{ones}(1, 27)$
7: $[\hat{\boldsymbol{p}}, \delta^2] = \text{Decode}(\tilde{\boldsymbol{x}}, \delta^2, \boldsymbol{G}, n, \boldsymbol{p}, \boldsymbol{q}, \tilde{\boldsymbol{J}}, \boldsymbol{o})$
8: **for** $n = 1 : N$ **do**
9: $\hat{\boldsymbol{u}}_n \leftarrow \boldsymbol{U}_{1:3,\hat{p}_n}$
10: **end for**
11: **return** $\hat{\boldsymbol{u}}$
12: **end function**

Algorithm 2 Sphere Block Decoding Algorithm - subroutine

1: **function** Decode($\tilde{\boldsymbol{x}}, \delta^2, \boldsymbol{G}, n, \boldsymbol{p}, \boldsymbol{q}, \tilde{\boldsymbol{J}}, \boldsymbol{o}$)
2: $b \leftarrow \text{sum}(1 : n)$
3: **for** $m = 1 : \text{nummel}(\boldsymbol{q})$ **do**
4: $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}_n$
5: **if** $n = 1$ **then**
6: $J \leftarrow \tilde{J}_{p_n}$
7: **else**
8: $p_{(n-1)} \leftarrow q_m$
9: $J \leftarrow \tilde{J}_{p_{n-1}}$
10: **for** $y = 1 : (n-1)$ **do**
11: $\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{G}_{(b-y),p_{(n-y)}}$
12: **end for**
13: **end if**
14: $\tilde{\boldsymbol{J}} = \|\boldsymbol{G}_b - \boldsymbol{x}\boldsymbol{o}\|^2 + J$
15: $\tilde{\boldsymbol{q}} = \text{find}(\tilde{\boldsymbol{J}} \leq \delta^2)$
16: **if** $\text{nummel}(\tilde{\boldsymbol{q}}) \geq 1$ **then**
17: **if** $n = N$ **then**
18: $[\delta^2, p_n] = \min(\tilde{\boldsymbol{J}})$
19: $\hat{\boldsymbol{p}} \leftarrow \boldsymbol{p}$
20: **else**
21: $[\hat{\boldsymbol{p}}, \delta^2] = \text{Decode}(\tilde{\boldsymbol{x}}, \delta^2, \boldsymbol{G}, (n+1), \boldsymbol{p}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{J}}, \boldsymbol{o})$
22: **end if**
23: **end if**
24: **end for**
25: **return** $[\hat{\boldsymbol{p}}, \delta^2]$
26: **end function**

The subroutine DECODE is now initiated with the new tree-level $(n+1)$, the branching path $\boldsymbol{p}$, the admissible parent nodes $\tilde{\boldsymbol{q}}$ and the cost of all 27 traversals $\tilde{\boldsymbol{J}}$. Note that $\boldsymbol{p}$, $\tilde{\boldsymbol{q}}$ and $\tilde{\boldsymbol{J}}$ convey the relative conditions at tree-level n. The same sequence of events is followed whereby b is computed, the for-loop identifies the first parent node in the queue, and the target sub-vector is extracted (lines:1 to 4). Tree-levels ($n > 1$), require updating of the branching path with the selected parent node and also the branching cost incurred by it (lines:8 and 9). Updating of the target sub-vector with the effects of previous branchings (lines:10 to 12) again leads to the cost computation and identification of admissible branchings. If no viable branchings exist $\tilde{\boldsymbol{q}} = [\ ]$, the branch is pruned and the next parent in the queue $\boldsymbol{q}$ is considered. Branching continues until the final tree level ($n = N$) is reached, whereupon the branching with minimal cost is identified. The last element of the index vector p_n is updated, and the branching cost incurred for the recorded path $\boldsymbol{p}$ is taken as the new squared sphere radius. With the tightened sphere radius the algorithm continues to prune non-admissible branches while searching for more cost-effective paths, until a single optimal path $\hat{\boldsymbol{p}}$ remains. This path is returned to the *main routine*, for construction of $\hat{\boldsymbol{u}}$.

Although not listed in Algorithm 2, the queue of admissible parent nodes at each tree level can be pruned further by applying the *switching constraint* and/or other selection strategies like *best-first* sorting.

A. Complexity

The proposed tree structure of the SBDA is more complex than the conventional approach, and an increase in the computational burden per algorithm *iteration* is to be expected. Computation of the 27 branching costs (line:14) contributes to the bulk of the burden imposed by Algorithm 2. It amounts to a (3×27)-matrix which undergoes subtraction, element squaring and column summations. Element wise $(\otimes, \oplus, \ominus)$, the total number of operations for (line:14) can be conveyed as

$$(3 \times 27)(2 \otimes + \oplus + \ominus). \tag{21}$$

Updating of the target sub-vector (line 11) requires

$$(3n - 3)\ominus \tag{22}$$

operations. Cumulatively, (21) and (22) effects a total of

$$3n + 321 \quad \text{flops} \tag{23}$$

per parent node visited. The number of node visits, i.e. algorithm iterations for the sphere decoding approach, is dependent on the selected sphere-radius. An optimal sphere radius selection that includes only one lattice point (the optimal solution), will result in a minimum number of algorithm iterations, which equals the depth of the relevant search tree. For the SBDA with tree depth $n = N$, the lower computational bound is therefore set to

$$\frac{3}{2}(N^2 + N + 214) \quad \text{flops}. \tag{24}$$

Storage of the matrix $\boldsymbol{G}$ requires memory space, which can be computed as

$$C = 3 \times 27 \times y \times 8 \text{ bytes}, \tag{25}$$

where y denotes the number of non-zero matrices (12), and 8 bytes, an assumed word length. Even for a relatively long prediction horizon of $N = 10$, the storage demand will be relatively small, i.e. memory capacity of 36 kB.

A theoretical analysis of the SDA's and the SBDA's respective pseudo-codes in terms of their *lower computational bounds* reveals the nature of the proposed approach. Figures 4(a) and 4(b) shows the relationship between algorithm iterations and flop count of the two respective decoders. Relative to the lower bound of the SDA (SDAlo), the lower bound of the SBDA (SBDAlo), exhibits a reduced number of algorithm iterations but poses an elevated flop count for short prediction horizons. However, the difference in flop count diminishes as the prediction horizon increases.

V. SIMULATED PERFORMANCE

In an attempt to find the *upper computational bounds* of the respective decoding algorithms, the drive system presented above was simulated in MATLAB©. A sampling frequency of $F_S = 12$kHz was chosen with the system parameters as listed in Table I. Base quantities were used to establish a per-unit system. For steady-state conditions a constant torque load $T_l = 1$pu at the rated speed $\omega_r^* = 0.9911$pu and optimal

TABLE I
PARAMETERS OF THE DRIVE SYSTEM.

Parameter	SI value	pu value
Nominal stator voltage	3300 V	0.5774
Nominal stator current	356 A	1.0000
Nominal power	1.587 MW	0.7799
Apparent power	2.035 MVA	1.0000
Rated stator frequency	50 Hz	1.0000
Number of pole pairs	5	
Rotational speed	595 rpm	0.9913
Air-gap torque	26.2 kNm	1.0000
Stator resistance	57.61 mΩ	0.0108
Rotor resistance	48.89 mΩ	0.0091
Stator leakage inductance	2.544 mH	0.1493
Rotor leakage inductance	1.881 mH	0.1104
Main inductance	40.01 mH	2.3486
Filter inductance	2 mH	0.1174
Filter Capacitance	200 μF	0.3363
Filter inductor resistance	2.0 mΩ	0.0004
Filter capacitor resistance	2.0 mΩ	0.0004
DC link voltage	5200 V	1.9299

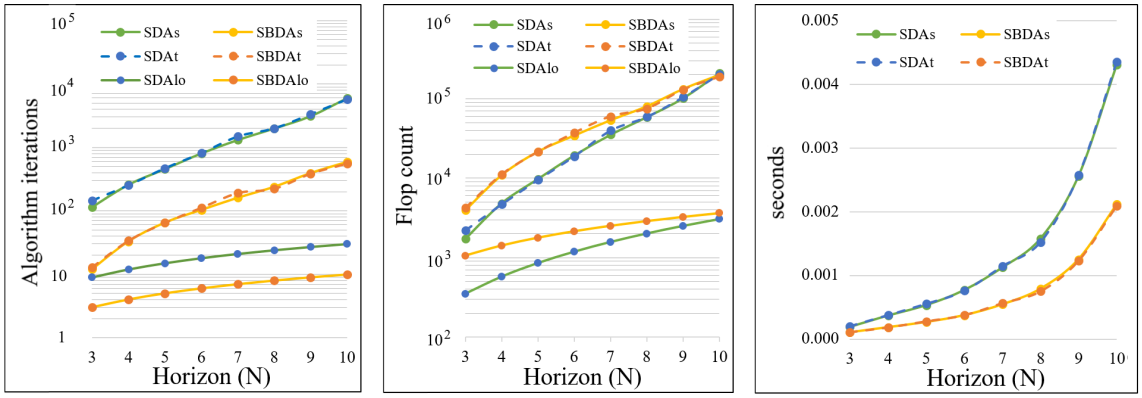

Fig. 3. Performance indicators of the SBDA and benchmark SDA simulated in MATLAB©. Figure (a) shows the algorithm iterations, Figure (b) the flop count and Figure (c), the average algorithm termination times. The respective results refer to values incurred per sample. Theoretical lower bounds are identified by the post-fix "lo" with the simulation results for steady-state operation and transients respectively identified by "s" and "t".

machine magnetisation $\Psi_r^* = 1$pu were selected. The response of the drive to transients was evaluated by invoking a rapid speed-step from $\omega_r^* = 0$pu to $\omega_r^* = 0.9911$pu while burdened with a variable torque load. In both cases, the main diagonal of the weighing matrix $\mathbf{\Lambda}$ was populated to strongly favour tracking of the stator current. Also, the switching weight λ_u was adjusted to achieve an average switching frequency per semiconductor device of $f_{sw} = 300$Hz.

Both, the SDA and SBDA were expected to solve the optimization problem (2), aided with intelligent initialization in the form of target preconditioning [7], and sphere-radius estimation [6]. Simulation of the FCS-MPC process followed a general order as illustrated by the state-machine in Figure 4. At every sample period (k), the output variables are obtained

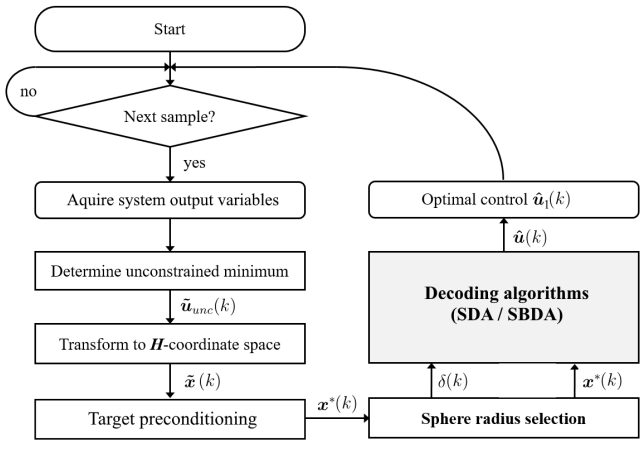

Fig. 4. State machine representation of the FCS-MPC scheme.

from which the unconstrained minimum $\tilde{\boldsymbol{u}}_{unc}$ is computed. Transformation to $\boldsymbol{H}$-coordinate space sets $\tilde{\boldsymbol{x}} = H\tilde{\boldsymbol{u}}_{\boldsymbol{unc}}$ as the target. If necessary, the target is preconditioned to give the updated target $\boldsymbol{x}^*$. The next step computes the initial sphere radius δ. After decoding, the solution to the optimisation problem (6) is produced, from which the first control vector $\hat{\boldsymbol{u}}_1$ (8) is extracted and applied as the optimal control $\boldsymbol{u}_{abc}$ action to the inverter.

Figures 4(a) to 4(c) shows the respective performance parameters when the simulations are conducted for 10 cycles of the fundamental stator current frequency. The recorded algorithm iterations and flop counts are the *maximum values* obtained during the simulation. The distinct nature of the SBDA reverberates in its upper computational bound where fewer algorithm iterations (Fig. 4(a)) are required but a heightened flop count (Fig. 4(b)) is recorded. Again, as the horizon extends, the flop count of the SBDA converges towards the SDA's bound. Figure 4(c) shows the average algorithm termination times recorded by the MATLAB© timing function. Upon investigation, it is evident that the respective algorithm complexities (Fig. 4(b)) are not captured by their corresponding termination times an opposite trend is observed. This contradiction is supported by the statement that flop count is a crude approach to the measurement of the efficiency of an algorithm in practice [13].

VI. Real-time simulation

To investigate this notion further, the same drive system was emulated on an OPAL-RT© 5600 real-time simulator. The realisation of the drive system in the RT-LAB system is illustrated in Figure 5. Note that the controller and plant are implemented on separate CPU cores and interfaced via the FPGA controlled analogue and digital IOs with a hardwired external loop. The external loop enforces hardware synchronisation, and in the process solidifies the performance results of the *controller* in real time. Reference settings such as the IM speed ω_r^* and flux Ψ_r^*, together with the mechanical load torque T_l, are managed from the host computer. Measurements of the plant during test procedures are recorded in the RT simulator and transferred to the host upon completion. Real-time implementation issues included the extrapolation of the system prediction model to compensate for an acquisition and

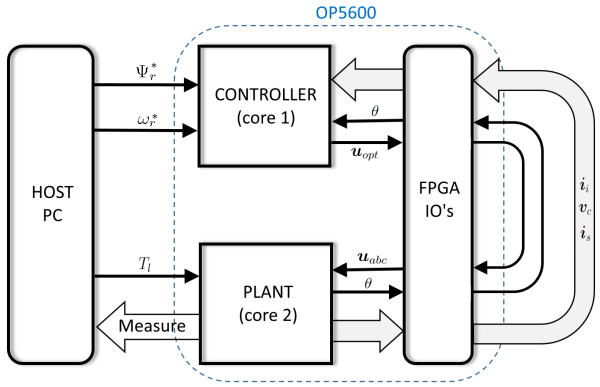

Fig. 5. Setup of the drive system on an OPAL-RT© 5600 platform.

communication delay of two sampling periods. Also, due to the dynamic nature of the system, i.e. varying rotor speed, the matrix $\boldsymbol{H}$ was precomputed for a range of incremental steps (1%) of the pu speed range.

A. Steady-state

Real-time simulation of the IM drive in steady-state conditions was done as specified in section VI. For the prediction horizon $N = 8$ case, the resulting waveforms are shown in Figure 6. Measurements of the inverter current $\boldsymbol{i}_i$ and stator current $\boldsymbol{i}_s$ were employed to compute their respective total demand distortions, i.e. $i_{i,TDD}$ and $i_{s,TDD}$. The torque total demand distortion $T_{e,TDD}$ was obtained by calculating T_e from the measured stator current $\boldsymbol{i}_s$ and estimated rotor flux $\boldsymbol{\psi}_r$. Table II list the results for prediction horizons $N = 3, 5$ and 8. At first glance, the dramatic effect

TABLE II
PERFORMANCE METRICS OF THE DRIVE SYSTEM DURING STEADY-STATE OPERATION.

N	f_{swd} (Hz)	$\boldsymbol{i}_{i,TDD}$ (%)	$\boldsymbol{i}_{s,TDD}$ (%)	$T_{e,TDD}$ (%)
3	314.3	13.422	3.592	2.737
5	301.3	13.144	2.802	2.139
8	307.8	12.495	1.849	1.490

of the LC filter on the current TDD can be noticed. The input current to the filter $\boldsymbol{i}_i$, with a fairly high TDD, is transformed to result in the stator current $\boldsymbol{i}_s$ with a low single-digit value. As a drive performance indicator, this improves with extension of the prediction horizon. Compared to the theoretical MATLAB© simulations, the real-time results showed a general decrease from the theoretical values. The deviation is contributed to the inaccuracies instilled by the analogue-to-digital and digital-to-analogue conversion processes.

Performance of the two controller algorithms was quantified in terms of their respective computation times t_{SDA} and t_{SBDA}. Reported by a function of the OPAL-RT© software, the termination times are plotted in the last graph of Figure

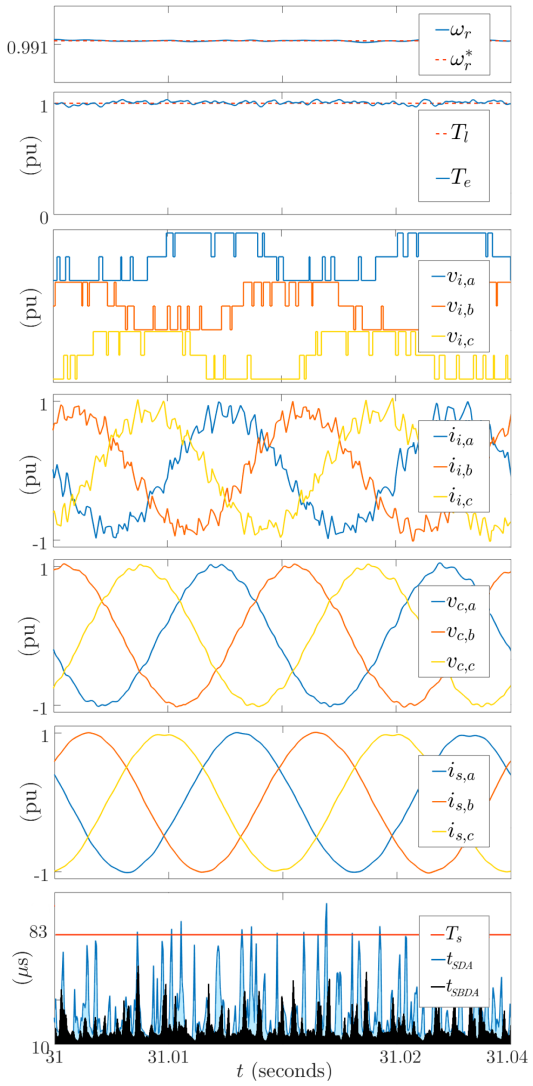

Fig. 6. Real-time simulation results of the IM drive during steady-state conditions. The inverter output voltages, stator currents and respective algorithm termination times are denoted by $\boldsymbol{v}_{i,abc}$, $\boldsymbol{i}_{s,abc}$ and t_{SDA}, t_{SBDA}.

6. Note that due to the sampling frequency of 12kHz, the computation time is bounded to the duration of a sampling period $T_S = 83\mu$s, which is indicated with the red line. The SBDA (dark silhouette), in general, terminates faster than the SDA and well within the sampling period, whereas the SDA terminates closer to the 83μs bound with frequent overruns[2] occurring. Prediction horizons 9 and 10 demanded too big a computational effort for successful termination of either algorithm.

B. Transients

The response of the drive to transients was tested for a speed step as specified in section VI. The prediction horizon

[2]An "overrun" occurs if the operations of the real-time simulator are not achieved within a period of the selected *fixed time-step*, i.e. sampling period. In such a case the real-time simulation is considered erroneous.

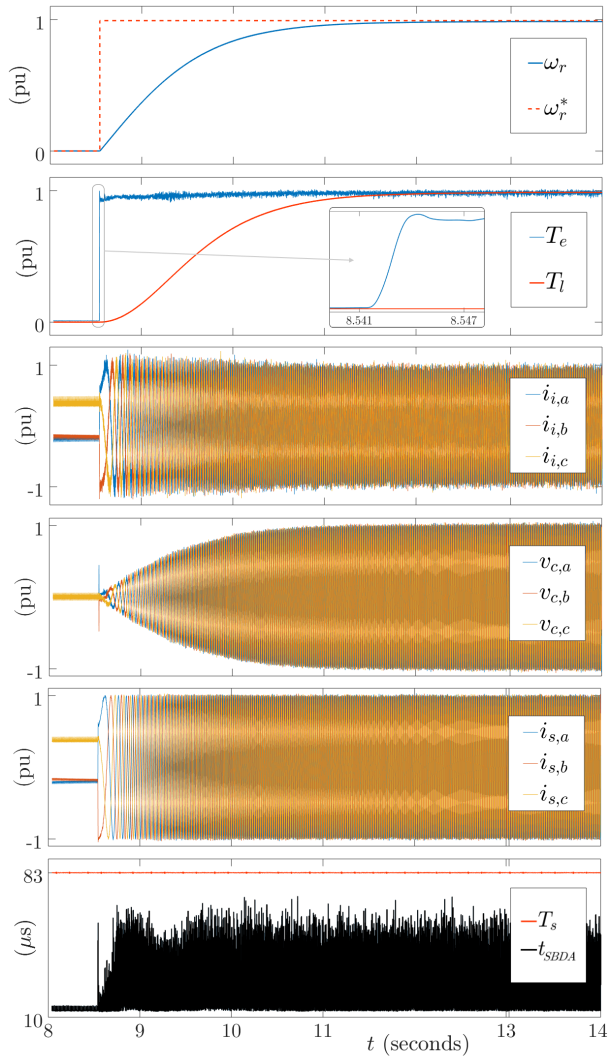

Fig. 7. System waveforms corresponding to a variation in rotor speed from 0pu to 0.911pu. Rotor speed ω_r, developed torque T_e, inverter current $\boldsymbol{i}_i$, capacitor voltage $\boldsymbol{v}_c$, stator current $\boldsymbol{i}_s$ and controller computation time t_{SBDA}.

$N = 8$ controller was selected with the resulting waveforms in Figure 7. The rapid speed-step in the reference ω_r^* can be observed, together with the rotor speed ω_r, delayed due to the inertia of the system. The rapid demand for an increase of the motor speed effects a large transient deviation $(\omega_r - \omega_r^*)$, which subsequently leads to the speed controller issuing a maximum torque request. This, almost immediate torque response T_e is maintained at a maximum until the required speed is attained. The enlarged view of the machine torque exhibits the rapid torque adjustment made by the predictive controller; a settling time of less than 10ms is achieved with minimal overshoot. During the acceleration process, the inverter current is slightly elevated while the capacitor voltage builds. Importantly, the stator current is optimally maintained throughout the transient period. The controller termination time, supported by the SBDA remained well within the sample period bound.

VII. Conclusion

An alternative sphere decoding algorithm was proposed to facilitate the FCS-MPC of a three-level NPC inverter driving a medium-voltage induction machine via an intermediate LC-filter in real time. The inherent characteristics of the Sphere block decoding algorithm with preprocessing were demonstrated with MATLAB© simulations. It confirmed the expected increase in computational complexity, i.e. flop count, but also showed that the SBDA becomes more competitive as the prediction horizon lengthen. The average termination times of the control algorithm in the theoretical MATLAB© simulations supported the statement that flop count is a crude approach to the measurement of the efficiency of an algorithm. Real-time simulation of the drive system on an OPAL-RT© 5600 simulator showed that the SBDA is efficient in solving the underlying FCS-MPC problem in real time for prediction horizons up to eight. The computability of the SBDA is mainly contributed to a general reduction in memory traffic, effected by the online block-matrix processing and supportive offline preprocessing.

References

[1] T. Geyer, "A comparison of control and modulation schemes for medium-voltage drives: emerging predictive control concepts versus PWM-based schemes", *IEEE Transactions on Industrial Applications*, vol. 47, no. 3, pp. 1380–1389, Mar. 2011.

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

[3] M.D. Dorfling, "Practical implementation of long-horizon direct model predictive control", *Masters dissertation, Stellenbosch University*, 2018.

[4] R. Baidya, "Multistep Model Predictive Control for Power Electronics and Electrical Drives", *PhD dissertation, University of New South Wales*, 2018.

[5] T. Geyer and D. Quevedo, "Multistep finite control set model predictive control for power electronics", *IEEE Transactions on Power Electronics*, vol. 29, no. 12 pp. 6836–6846, Feb. 2014.

[6] P. Karamanakos, T. Geyer and R. Kennel, "Reformulation of the long-horizon direct model predictive control problem to reduce the computational effort", *Energy Conversion Congress and Exposition (ECCE)*, pp. 3512-3519, Sep. 2014.

[7] R. Baidya, R. Aguilera, P. Acuna, R. Delgado, T. Geyer, D. Quevedo and T. Mouton, "Fast multistep finite control set model predictive control for transient operation of power converters", *Annual Conference of the Industrial Electronics Society*, pp. 5039–5045, Oct. 2016.

[8] P. Karamanakos, T. Geyer, and R.P. Aguilera, P. Ricardo, "Computationally efficient long-horizon direct model predictive control for transient operation.", IEEE Energy Conversion Congress and Exposition (ECCE), pp. 4642-4649, 2017.

[9] D. Watkins, "Fundamentals of Matrix Computations", *John Wiley & Sons*, vol. 2, pp. 9–11, Aug. 2004.

[10] T. Geyer, "Model predictive control of high power converters and industrial drives", *John Wiley & Sons*, pp. 234–250, Nov. 2016.

[11] J. Cassels, "An introduction to the geometry of numbers", *Springer Science & Business Media*, 2012.

[12] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity", *IEEE transactions on signal processing*, vol. 53, no. 8, pp. 2806-2818, Jul. 2005.

[13] G. Golub and C. Van Loan, "Matrix computations", *The Johns Hopkins University Press*, vol. 3, p.17, Dec. 2012.