# Experimental Evaluation of Sphere Decoding for Long-Horizon Direct Model Predictive Control

Martinus Dorfling*, Hendrik Mouton*, Petros Karamanakos†, Tobias Geyer‡
* Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, Matieland, 7602
Stellenbosch, South Africa
Email: 17041317@sun.ac.za, dtmouton@sun.ac.za
† Faculty of Computing and Electrical Engineering
Tampere University of Technology
PO Box 638, FI-33101
Tampere, Finland
Email: p.karamanakos@ieee.org
‡ ABB Corporate Research
Segelhofstrasse 1K
5405 Baden-Dättwil, Switzerland
Email: t.geyer@ieee.org

## Keywords

≪Control methods for electrical systems≫, ≪Digital control≫, ≪Field Programmable Gate Array (FPGA)≫, ≪Optimal control≫, ≪Three-phase system≫

## Abstract

In the past few years, the interest in model predictive control (MPC) for power electronics has increased significantly. It proves to be a good alternative to space-vector modulation at low switching frequencies. It has been shown that an increase in the prediction horizon improves the system performance. Unfortunately, increasing the horizon leads to an exponential increase in the complexity of the optimization problem. To date, most of the work that has been done is only in simulation, ignoring real-time requirements.

This paper offers a practical implementation of long-horizon direct MPC. Methods on how to efficiently implement the controller on a field programmable gate array are discussed. Sphere decoding is used to solve the optimization problem. Results show that the implementation of a 5-step horizon MPC and a sphere decoder is very efficient and effective, as it requires only 8.6 µs, in roughly 80 % of the cases, to execute the control law with a sampling period of 25 µs when a three-phase neutral-point clamped inverter with an RL load is examined.

## Introduction

At low switching frequencies (well below 1 kHz), MPC offers a substantial decrease in the current total harmonic distortion (THD) compared to space-vector modulation (SVM) [1]. Moreover, MPC also has the benefits of having good transient performance, being well suited for multiple-input multiple-output (MIMO) systems, and constraints that can be imposed on the controlled variables [2]. By extending the prediction horizon (i.e., the number of discrete-time steps over which the system behaviour is predicted), the THD is decreased even further [1]. Under certain conditions MPC almost achieves the THD of

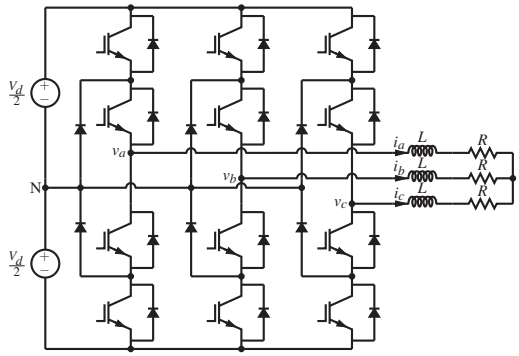



Fig. 1: Three-phase three-level NPC inverter with an RL-load.

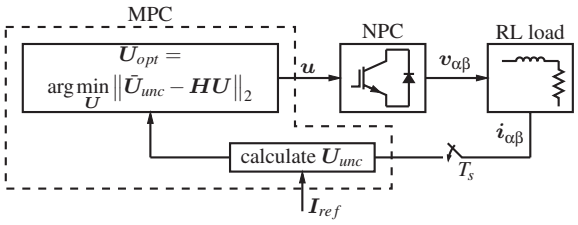



Fig. 2: Control diagram of MPC as ILS.

optimized pulse patterns (OPPs) [1], which is the theoretical lower bound of THD at low switching frequencies.

Direct MPC methods directly manipulate the inverter switch positions and thus bypass the modulator. Inherently, direct MPC necessitates an integer optimization problem to be solved in order to find the optimal switch positions of the inverter. The computational complexity of the optimization problem increases exponentially with the length of the horizon. In order to solve the optimization problem in real time, a branch-and-bound method known as sphere decoding is used [3].

According to the authors' knowledge (and at the time of writing), only up to a prediction horizon of $N = 2$ has been practically implemented [4]. For prediction horizons greater than $N > 2$, only simulation results have been presented [1, 5, 6]. This paper presents an implementation of MPC with a horizon of $N = 5$ in a practical system. The results are focused on the performance of the sphere decoder used to solve the optimization problem (and not the performance benefits of long horizons [1]) in a practical system. Also, the implementation of the controller will be discussed, since efficient use of the available resources in an field programmable gate array (FPGA) is of the utmost importance to achieve long horizons.

# Control problem

## Control law formulation

The neutral-point clamped (NPC) inverter with an RL-load in Fig. 1 is considered. The neutral point N is considered to be at a fixed potential, while the bus voltage $V_d$ is also considered to be constant. The output voltage of a phase arm is given by $v_x = \frac{V_d}{2} u_x$, where $x$ denotes the phase with $x \in \{a,b,c\}$, while $u_x$ denotes the switch position for a given phase arm with $u_x \in \{-1,0,1\}$.

To simplify the modelling process, the 3-phase variable ($abc$) is first transformed to a stationary orthogonal coordinate system ($\alpha\beta$) by using the Clarke transformation, $\boldsymbol{\xi}_{\alpha\beta} = \boldsymbol{K}\boldsymbol{\xi}_{abc}$ where

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

After modelling in the continuous-time domain and then discretizing by using the exact Euler discretization, the discrete-time state-space model is given by

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

where

$$\boldsymbol{A} = e^{\boldsymbol{F}T_s},\ \boldsymbol{B} = -\boldsymbol{F}^{-1}(\boldsymbol{I}_2 - \boldsymbol{A})\boldsymbol{G}\ ,\ \boldsymbol{i}(k) = \begin{bmatrix} i_\alpha(k) \\ i_\beta(k) \end{bmatrix},\ \boldsymbol{u}(k) = \begin{bmatrix} u_a(k) \\ u_b(k) \\ u_c(k) \end{bmatrix},$$

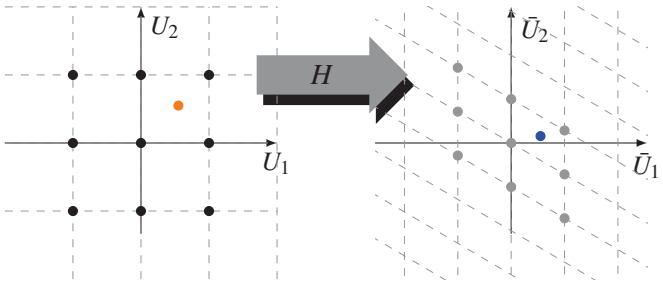



Fig. 3: The $\boldsymbol{H}$ matrix linearly maps all of the possible switching vectors (represented by the black dots) to candidate solutions (represented by the gray dots) for the ILS problem. The unconstrained optimum (represented by the orange dot) is also mapped to the transformed space (and is now represented by the blue dot).

$$\boldsymbol{F} = -\frac{R}{L}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \text{ and } \boldsymbol{G} = \frac{V_d}{2L}\boldsymbol{K},$$

with $T_s$ being the sampling period and $\boldsymbol{I}_2$ the $2\times 2$ identity matrix.

In MPC, a cost function is formulated based on the control objectives. These objectives usually include (but are not limited to) minimization of the tracking error (which relates to the current THD) and the switching effort (which relates to the efficiency of the system). The cost function for $N$ prediction steps is given by

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

where $\boldsymbol{i}_e = \boldsymbol{i}_{ref} - \boldsymbol{i}$ is the error between the reference and the predicted current, $\Delta\boldsymbol{u}(l) = \boldsymbol{u}(l) - \boldsymbol{u}(l-1)$, and $\lambda_u$ is the weighting factor on the switching effort and trades-off between the tracking accuracy of the controller and the switching frequency.

By reformulating (2) as an integer-least squares (ILS) function, the optimization problem becomes [3]

$$\begin{aligned}\boldsymbol{U}_{opt}(k) = &\arg\min_{\boldsymbol{U}(k)} \|\boldsymbol{H}\boldsymbol{U}_{unc}(k) - \boldsymbol{H}\boldsymbol{U}(k)\|_2^2\\ &\text{subject to } \boldsymbol{U}(k) \in \{-1,0,1\}^{3N},\end{aligned} \tag{3}$$

where

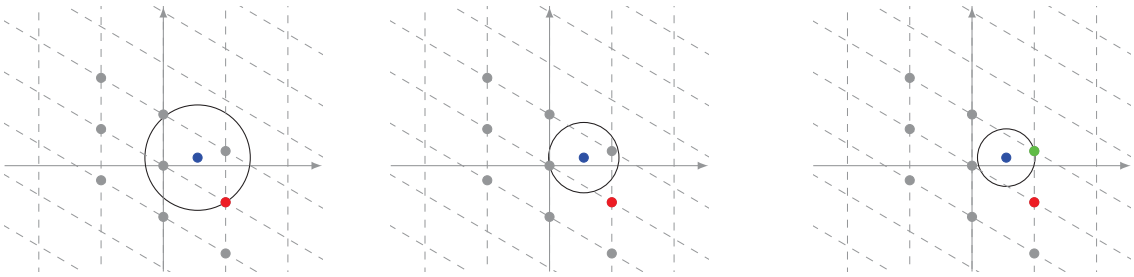
$$\boldsymbol{U}(k) = \begin{bmatrix}\boldsymbol{u}^T(k) & \boldsymbol{u}^T(k+1) & \cdots & \boldsymbol{u}^T(k+N-1)\end{bmatrix}^T$$

is the switching vector over the prediction horizon and $\boldsymbol{U}_{unc}(k) \in \mathbb{R}^{3N}$ is the *unconstrained optimal solution* (i.e., the solution if the integer constraint on $\boldsymbol{U}(k)$ is relaxed), and $\boldsymbol{H} \in \mathbb{R}^{3N\times 3N}$ is a lower triangular matrix. The $\boldsymbol{H}$ matrix generates a truncated lattice (where the points represent candidate solutions) from all of the possible switching vectors,

$$\Lambda = \{\boldsymbol{H}\boldsymbol{U}(k) \mid \boldsymbol{U}(k) \in \{-1,0,1\}^{3N}\}, \tag{4}$$

while also transforming the unconstrained optimum to the transformed search space. This linear transformation is illustrated in Fig. 3. As evident in (3), finding the optimal solution $\boldsymbol{U}_{opt}(k)$ relates to finding the closest lattice point to the transformed unconstrained optimum $\bar{\boldsymbol{U}}_{unc}(k) = \boldsymbol{H}\boldsymbol{U}_{unc}(k)$.

It should be mentioned that once the optimal solution $\boldsymbol{U}_{opt}(k)$ has been found, only the first entry (i.e., $\boldsymbol{u}_{opt}(k)$) is applied to the system. The remaining entries are discarded and the optimization process is repeated at the next sampling instance with new measurements. This provides feedback and robustness to the system and is known as the *receding horizon principle*.



(a) A radius is chosen based on a initial candidate solution.

(b) The tightening of the sphere around the transformed unconstrained solution.

(c) The sphere is tightened until only one point lies within the sphere, which is the optimal solution.

Fig. 4: Principle of sphere decoding.

## Solving the ILS

Solving the ILS problem in an exhaustive manner is not computationally feasible, as the amount of candidate solution is given by $3^{3N}$. To solve (3) in a practical realisable time, sphere decoding is adopted [3]. Sphere decoding involves constructing a sphere with radius

$$\rho(k) = \left\| \bar{U}_{unc}(k) - \boldsymbol{H}\boldsymbol{U}(k) \right\|_2, \tag{5}$$

which is centred at the transformed unconstrained optimum $\bar{U}_{unc}(k)$. The principle of sphere decoding is as follows: First, an initial radius is chosen, as explained later. The sphere is then tightened until only one lattice point (i.e., a candidate solution) lies within the sphere, which is the optimal solution. This procedure is illustrated in Fig. 4.

An important characteristic of the $H$ matrix is its triangularity. Exploiting the fact that $H$ is triangular allows the candidate solution to be assembled entry by entry in a computational efficient manner.. This creates a search tree with depth $n = 3N$ (as shown in Fig. 5) that is traversed while assembling the candidate solution. At every node of the search tree, an intermediate radius is calculated. If the intermediate radius exceeds that of the sphere, the branch is pruned and the sphere decoder backtracks to examine unexplored nodes. When a leaf node (i.e., a bottom node) is reached and the candidate solution is within the sphere, the sphere is tightened. The sphere decoder terminates once the entire search tree is examined and only then can an optimal solution be certain (i.e., an optimality certificate is obtained).

The choice of initial radius is no less important than the sphere decoding itself, as it is essential to achieve a quick search through the search tree. If the radius is too large (and thus includes many candidate solutions), many nodes of the search tree will have to be explored. On the other hand, if the radius is too small, the set of candidate solution will be empty. Two effective initial candidate solutions are the *Babai estimate* [7] and the *educated guess* [3]. The educated guess $\boldsymbol{U}_{ed}(k)$ is based on the previous optimal solution, where the entries of $\boldsymbol{U}_{opt}(k-1)$ are shifted one time step forward and with a repetition of the last entry. This is in accordance with the receding horizon principle. The Babai estimate $\boldsymbol{U}_{bab}(k)$ is based on the unconstrained solution, which is rounded off to the nearest switching vector (i.e., the black dots in Fig. 3). The closer the lattice is to being orthogonal, the more likely the initial guess will be the optimal solution itself.

It is recommended that both the initial radii should be calculated. Under steady state conditions, the educated guess offers a remarkably good initial candidate solution, since in about 80 % of the cases the educated guess requires only one candidate solution to be explored in the search tree [2]. However, during transients the educated guess can be a poor guess. Since the educated guess is based on the previous sample, the guess has no information about the unconstrained solution. Therefore, if a transient occurs and the resulting optimal solution differs significantly from the previous optimal solution, the initial candidate solution is located far from the unconstrained solution, and thus leading to a very large initial radius enclosing many candidate solutions. At higher dimensions (i.e., longer horizons), this effect

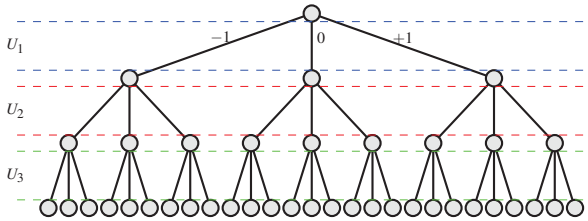



Fig. 5: An example of a search tree with a horizon of $N = 1$. $U_j$ corresponds to the $j$'th entry in $\boldsymbol{U}(k)$.

intensifies. As stated previously, the Babai estimate is based on the unconstrained solution and will be better suited for transients, as the initial candidate solution will be located its vicinity. Therefore, the initial radius will be

$$\rho_{ini}(k) = \min\{\rho_{bab}(k), \rho_{ed}(k)\}, \tag{6}$$

where $\rho_{bab}(k)$ and $\rho_{ed}(k)$ are calculated from (5).

Note that during certain transients, the unconstrained solution could be located far away from the truncated lattice. In these situations even the Babai estimate will have a large initial radius.

## Practical controller implementation

In order to implement long-horizon MPC on an FPGA, the resources at hand must be used sparingly. An inefficient and ineffective implementation of the controller (especially the calculation of the unconstrained solution) can not only limit the horizon, but also the maximum obtainable clock speed (thus limiting the operations per sampling period). It is important to note that the implementation and explanations being given is for a three-phase NPC inverter with an RL load. Different system will have different matrices, but the core process remain the same.

### Calculation of unconstrained solution and initial radius

The unconstrained solution is given by [3]

$$\boldsymbol{U}_{unc}(k) = -\boldsymbol{Q}^{-1}\boldsymbol{\Theta}(k) \tag{7}$$

where

$$\boldsymbol{Q} = \boldsymbol{\Upsilon}^T\boldsymbol{\Upsilon} + \lambda_u \boldsymbol{S}^T\boldsymbol{S} \tag{8}$$

and

$$\boldsymbol{\Theta}(k) = \boldsymbol{\Upsilon}^T[\boldsymbol{\Gamma}\boldsymbol{i}(k) - \boldsymbol{i}_{ref}(k)] - \lambda_u \boldsymbol{S}^T\boldsymbol{E}\boldsymbol{u}(k-1), \tag{9}$$

with $\boldsymbol{i}_{ref}(k)$ representing the reference current over the prediction horizon, and

$$\boldsymbol{\Gamma} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix}, \boldsymbol{\Upsilon} = \begin{bmatrix} \boldsymbol{B} & \mathbf{o}_{2\times3} & \cdots & \mathbf{o}_{2\times3} \\ \boldsymbol{AB} & \boldsymbol{B} & \cdots & \mathbf{o}_{2\times3} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}^{N-2}\boldsymbol{B} & \cdots & \boldsymbol{B} \end{bmatrix}, \boldsymbol{S} = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{o}_{3\times3} & \cdots & \mathbf{o}_{3\times3} \\ -\boldsymbol{I}_3 & \boldsymbol{I}_3 & \cdots & \mathbf{o}_{3\times3} \\ \mathbf{o}_{3\times3} & -\boldsymbol{I}_3 & \cdots & \mathbf{o}_{3\times3} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{o}_{3\times3} & \mathbf{o}_{3\times3} & \mathbf{o}_{3\times3} & \boldsymbol{I}_3 \end{bmatrix} \text{ and}$$

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{o}_{3\times3} & \mathbf{o}_{3\times3} & \cdots & \mathbf{o}_{3\times3} \end{bmatrix}^T.$$

To minimize the use of multipliers on the FPGA, it is essential to use pipelining. Instead of calculating all the entries of $\boldsymbol{\Theta}(k)$ at once, only one entry should be calculated per clock cycle. The coefficients of the equation can then be changed when calculating the next entry. The calculation of the unconstrained

solution will be slower when calculating one entry at a time, but in general the maximum clock speed will increase (depending on the critical path in the FPGA) and thus also the available operations per sampling period.

The $j$'th entry of $\boldsymbol{\Theta}(k)$ is given by

$$\begin{aligned}\Theta_j = &\Upsilon^T_{(j,1)}(\Gamma_{(1,1)}i_1+\Gamma_{(1,2)}i_2-I_{ref,1})+\Upsilon^T_{(j,2)}(\Gamma_{(2,1)}i_1+\Gamma_{(2,2)}i_2-I_{ref,2})\\ &+\Upsilon^T_{(j,3)}(\Gamma_{(3,1)}i_1+\Gamma_{(3,2)}i_2-I_{ref,3})+\cdots+\Upsilon^T_{(j,2N)}(\Gamma_{(2N,1)}i_1+\Gamma_{(2N,2)}i_2-I_{ref,2N})\\ &-\lambda_u Z_j U(k-1)_j,\end{aligned} \tag{10}$$

with

$$\boldsymbol{Z} = \begin{bmatrix}1 & 1 & 1 & 0 & \cdots & 0\end{bmatrix}^T,$$

where $\boldsymbol{U}(k-1)$ is the previous switching vector (not necessarily the optimal solution) and $\boldsymbol{Z}$ is introduced by noting that only the first entry of $\boldsymbol{U}(k-1)$ (i.e., $\boldsymbol{u}(k-1)$) influences $\boldsymbol{\Theta}(k)$. Exploitations for the given system can further be made by noting that all the even entries in the first column of $\boldsymbol{\Gamma}(k)$ are zero and all the odd entries in the second column are zero .

The calculation of the $j$'th entry of the unconstrained solution is given by,

$$U_{unc,j} = -Q^{-1}_{(j,1)}\Theta_1 - Q^{-1}_{(j,2)}\Theta_2 - Q^{-1}_{(j,3)}\Theta_3 - \cdots - Q^{-1}_{(j,3N)}\Theta_{3N}, \tag{11}$$

If the Babai estimate is not used, the calculation of the transformed unconstrained solution $\bar{\boldsymbol{U}}_{unc}(k)$ can be done in one step by pre-multiplying $\boldsymbol{H}$ and $-\boldsymbol{Q}^{-1}$. If the Babai estimate is used, however, the unconstrained solution $\boldsymbol{U}_{unc}(k)$ is also needed in order to round off to the closest switching vector. Fortunately, the calculation of the unconstrained solution and the transformed unconstrained solution is described by the same polynomial and thus only the coefficients need to be changed.

Instead of calculating the radius of the sphere, it is more efficient to use the radius squared. By using the squared radius, the intermediate radii can simply be added together. Also, there is no need to calculate the root, saving more resources. The initial radius squared can be calculated with

$$\rho^2_{ini} = \rho^2_{ini,1} + \rho^2_{ini,2} + \cdots + \rho^2_{ini,n}, \tag{12}$$

where the $j$'th intermediate radius is calculated with

$$\rho^2_{ini,j} = (\bar{U}_{unc,j} - H_{(j,1)}U_{ini,1} - H_{(j,2)}U_{ini,2} - H_{(j,3)}U_{ini,3} - \cdots - H_{(j,3N)}U_{ini,3N})^2 + \rho^2_{ini,(j-1)}, \tag{13}$$

where $\boldsymbol{U}_{ini}$ represents either the Babai or educated guess.

## Delay compensation

In the practical system, there are two delays that can easily be taken into account. The first is the analogue-to-digital (ADC) delay. The ADC converter samples at a fixed time $t_d$ before the sampling period of the controller. The value can then be extrapolated by using a discrete-time state-space representation of the system with a period of the delay,

$$\boldsymbol{i}(k) = \boldsymbol{A}_d \boldsymbol{i}_{sampled} + \boldsymbol{B}_d \boldsymbol{u}(k-1), \tag{14}$$

where

$$\boldsymbol{A}_d = e^{\boldsymbol{F}t_d},\ \boldsymbol{B}_d = -\boldsymbol{F}^{-1}(\boldsymbol{I}_2 - \boldsymbol{A}_d)\boldsymbol{G}.$$

Note that switch position stays constant within a sample period $T_s$.

The controller considers switching only at the sampling instants. Therefore, a computational delay of

one sampling period is added to the system,

$$\boldsymbol{i}(k+1) = \boldsymbol{A}\boldsymbol{i}(k) + \boldsymbol{B}\boldsymbol{u}(k). \tag{15}$$

It is important to notice that with the added computational delay, the optimization problem corresponds to time-step $k+1$ instead of $k$.

## Sphere decoding algorithm

In [8] an FPGA implementable sphere decoding algorithm (SDA) was proposed. Unlike the SDA in [3], this algorithm was formulated to be non-recursive, which makes it well suited for implementation in an FPGA. The algorithm from [8] is shown Algorithm 1.

**Algorithm 1** Non-Recursive Sphere Decoder

1: **function** $\boldsymbol{U}_{opt} = \text{SPHDEC}(\boldsymbol{U}, 3N, \rho^2, \bar{\boldsymbol{U}}_{unc})$
2: **Initialize:**
3: $j = 1$
4: set each element in $\mathbf{sp} = -1$
5: set each element in $\boldsymbol{d} = 0$
6: **while** solutionfound $= 0$ **do**
7: $U_j = \text{sp}_j$
8: $d'^2 = \left\| \bar{U}_{unc,j} - \boldsymbol{H}_{(j,1:j)} \boldsymbol{U}_{1:j} \right\|_2^2 + d_j^2$
9: **if** $\rho^2 \geq d'^2$ **then**
10: **if** $j = n$ **then**
11: $\boldsymbol{U}_{opt} = \boldsymbol{U}$
12: $p^2 = d'^2$
13: $\text{sp}_j{+}{+}$
14: **else**
15: $j{+}{+}$
16: $d_j^2 = d'^2$
17: **end if**
18: **else**
19: $\text{sp}_j{+}{+}$
20: **end if**
21: **for** $q = 3N$ downto 2 **do**
22: **if** $\text{sp}_q > 1$ **then**
23: $\text{sp}_q = -1$
24: $j = q - 1$
25: $\text{sp}_j{+}{+}$
26: **end if**
27: **end for**
28: **if** $\text{sp}_1 > 1$ **then**
29: solutionfound = 1
30: **end if**
31: **end while**
32: **end function**

The algorithm is similar to the one proposed in [3], but instead of a stack, uses two types of pointers to traverse the search tree. The *level pointer*, denoted $j$, keeps track of what level in the search tree is being explored. The pointer ranges from 1 to $3N$. The *switch pointer*, denoted by $\text{sp}_j$, keeps track of which switch position is being evaluated at a given level. There are $3N$ level pointers stored in an array.

Lines 21 to 29 in Algorithm 1 enables the sphere decoder to backtrack and visit unexplored nodes. It evaluates all the switch pointer from bottom to top. If any given level has a switch position that exceeds

1, it means that the level has been fully explored. The sphere decoder then backtracks one level and increments the switch position of the current level. If the switch pointer in the first level exceeds 1, it means that the search tree has been fully explored and the algorithm terminates.

## Results

The controller was written in VHDL and implemented on an Altera Cyclone V 5CSEMA5F31C6 FPGA. The prediction horizon was $N = 5$ (that is $14,348,907$ candidate solutions), and the algorithm (this includes the calculation of the unconstrained solution, initial radius, and the SDA) was clocked at 15 MHz. The FPGA resource usage and system parameters are shown in Table I and Table II respectively.

Table I: FPGA resource usage.

| Resource | Amount |
|---|---|
| DSP blocks | 82/87 (94 %) |
| Logic utilization | 8,384/32,070 (26 %) |
| Total registers | 4,861 |

Table II: System parameters.

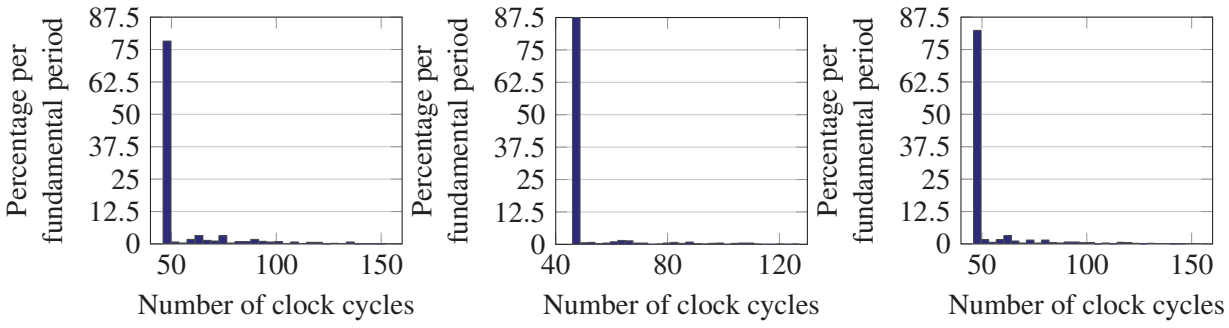
| Parameter | Description | Value |
|---|---|---|
| $V_d$ | Bus voltage | 100 V |
| $R$ | Load resistance | 3.5 Ω |
| $L$ | Load inductance | 2 mH |
| $T_s$ | Sample period | 25 µs |
| $N$ | Prediction horizon | 5 |
| $\lambda_u$ | Weighting factor | 6 |

A clock speed of 15 MHz and a sampling period of 25 µs results in 375 clock cycles per period in which calculations can be done. The first 82 clock cycles are for the calculation of the unconstrained solution and the initial radius. The remaining 293 clock cycles are used for the sphere decoding. At every clock cycle, the SDA investigates a node in the search tree (i.e., all of the lines in Algorithm 1 are executed). SignalTap was used to capture data from the FPGA.

### Steady-state performance

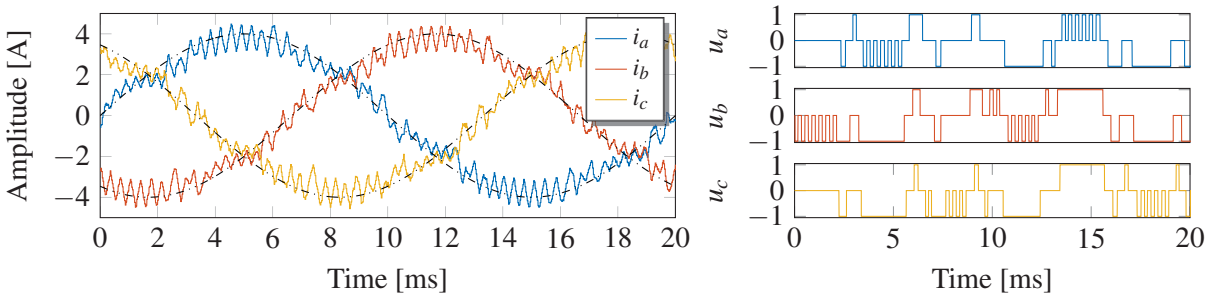
To evaluate the performance of the sphere decoder, the number of clock cycles used to find the optimal solution is a good metric. The histogram in Fig. 6 shows the number of clock cycles required for the sphere decoder to find the optimal solution during a 50 Hz fundamental period, which is 800 samples. For a reference current of 4 A (Fig. 6a), the SDA requires only 45 clock cycles in 78 % of the instances to find the optimal solution. This translates to 3 µs, i.e., 8.6 µs in total, including the time for the calculation of the unconstrained solution and the initial radius. For a reference current of 8 A (Fig. 6b), the SDA requires only 45 clock cycles in 88 % of the instances, while for a reference current of 9.5 A (Fig. 6c) the SDA requires 45 clock cycles in 82 % of the instances. The longest the SDA took to find the optimal solution, considering all three cases in Fig. 6, was 160 clock cycles and occurred only once.

In Fig. 7, Fig. 8, Fig. 9 the phase currents and switch positions of peak reference currents of 4 A, 8 A and 9.5 A are respectively shown.



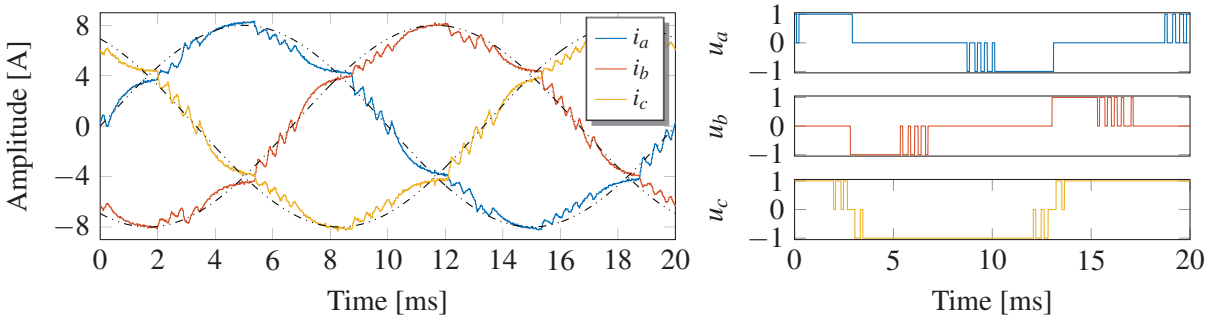

(a) 4 A reference. (b) 8 A reference. (c) 9.5 A reference.

Fig. 6: Distribution of the clock cycles required for sphere decoding during one fundamental period.



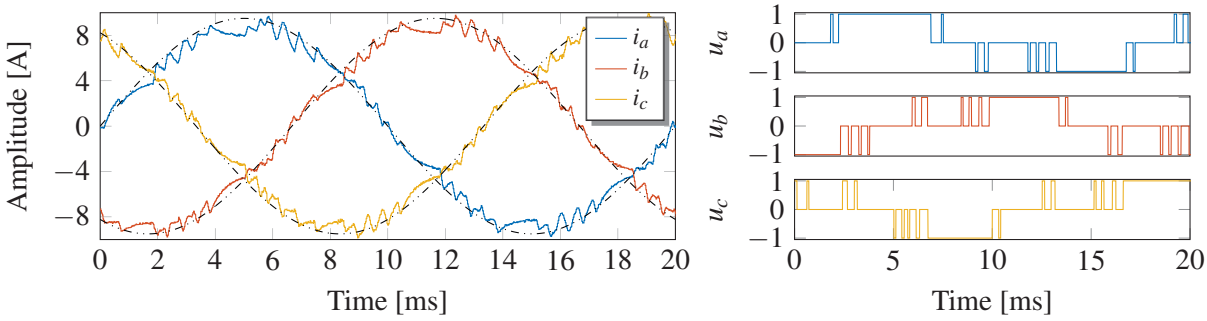

(a) Phase currents. (b) Switch positions.

Fig. 7: Measured waveforms of the phase currents and switch positions with a reference of 4 A. The average switching frequency is approximately 530 Hz.




(a) Phase currents. (b) Switch positions.

Fig. 8: Measured waveforms of the phase currents and switch positions with a reference of 8 A. The average switching frequency is approximately 240 Hz.




(a) Phase currents. (b) Switch positions.

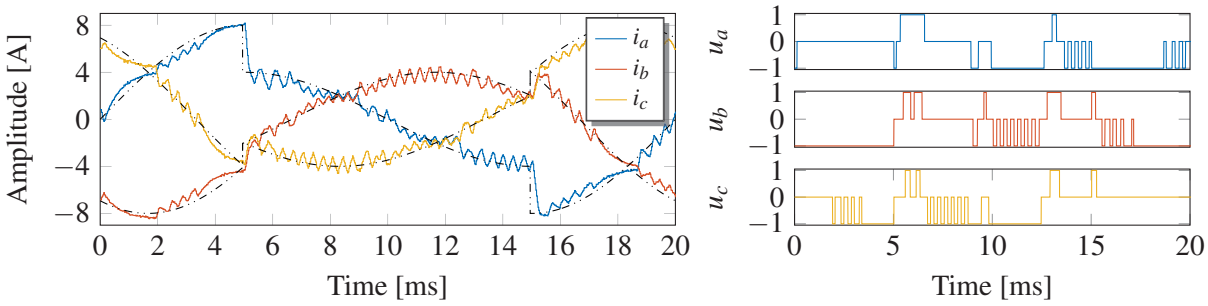
Fig. 9: Measured waveforms of the phase currents and switch positions with a reference of 9.5 A. The average switching frequency is approximately 360 Hz.

## Transient performance

One of the benefits of MPC is its very fast transient response. However, if the optimization problem is not solved (via sphere decoding) within the sampling period, these benefits will be nullified. In Fig. 10, the current reference is stepped from 8 A to 4 A and then back to 8 A. As seen in Fig. 10a, the current is quickly regulated to the new reference.

The SDA performance proved to be minimally affected by the transients. The three sampling instances from when the first reference step was applied required only 45, 99, and 54 clock cycles to find the optimal solution. The second reference step required 120, 111, and 45 clock cycles.

In the case where a certificate for optimally is not found, the Babai estimate is used as the solution [9]. Fortunately, following an empirical analysis using the current parameters, this never occurred under




(a) Phase currents.

(b) Switch positions.

Fig. 10: Measured waveforms of the phase currents and switch positions with a transient from 8 A to 4 A and back to 8 A.

steady-state conditions and barely occurs during transients. The ideal situation would be to apply the incumbent solution found by the sphere decoder.

## Conclusion

This paper presented an implementation of direct MPC with long horizons in an FPGA for a practical system. It was demonstrated that the SDA algorithm solves the optimization well within the chosen sampling period. The limiting factor of the horizon was the available resources on the FPGA. It was shown by using both the Babai estimate and educated guess for the computation of the initial radius, that the SDA is a well-suited solver for the optimization problem during transients.

Future work will include a rigorous analysis on the benefits of long horizons. This also includes how the weighting factor $\lambda_u$ affects the switching frequency and the THD.

## References

[1] 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, No. 3, March. 2015.

[2] T. Geyer: "Model predictive control of high power converters and industrial drives", Wiley, ISBN. 9781119010906, 2016.

[3] T. Geyer and D. E. Quevedo: "Multistep finite control set model predictive control for power electronics", *IEEE Transactions on power electronics*, Vol. 29, No. 12, December 2014.

[4] 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, no. 5, pp. 4007-4020, May 2017.

[5] P. Karamanakos, T. Geyer and R. Kennel: "Computationally efficient optimization algorithms for model predictive control of linear systems with integer inputs", *IEEE 54th Annual Conference on Decision and Control*, December 2015.

[6] P. Karamanakos, T. Geyer and R. Kennel: "Reformulation of the long-horizon direct model predictive control problem to reduce the computational effort", *IEEE Energy Conversion Congress and Exposition*, September 2014.

[7] L. Babai: "On Lovaśz' lattice reduction and the nearest lattice point problem", *Combinatorica*, Vol. 6, No. 1, March 1986.

[8] M. D. Dorfling, H. dT. Mouton, P. Karamanakos and T. Geyer: "Implementation of a sphere decoder in an FPGA for direct model predictive control with long horizons", *Southern African Universities Power Engineering Conference*, January 2017.

[9] P. Karamanakos and T. Geyer and R. Kennel: "Suboptimal search strategies with bounded computational complexity to solve long-horizon direct model predictive control problems", *2015 IEEE Energy Conversion Congress and Exposition (ECCE)*, September 2015.