Differential Evolution Based Nonlinear Model Predictive Speed Control of PMSM Implemented on GPU

KOZUBÍK, M.; FRIML, D.

Proceedings of 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)

eISBN: 978-1-7281-9023-5

DOI: https://doi.org/10.1109/ISIE45552.2021.9576359

Differential Evolution Based Nonlinear Model Predictive Speed Control of PMSM Implemented on GPU

Michal Kozubik
CEITEC - Central European Institute of Technology
Brno University of Technology
Brno, Czech Republic
Email: Michal.Kozubik@ceitec.vutbr.cz

Dominik Friml
CEITEC - Central European Institute of Technology
Brno University of Technology
Brno, Czech Republic
Email: Dominik.Friml@ceitec.vutbr.cz

***Abstract*—In this paper, the novel approach to the nonlinear model predictive speed control of a permanent magnet synchronous motor and its implementation is introduced. The implementation is performed using general-purpose computing on graphics processing unit. The introduced algorithm uses the optimization method based on the differential evolution to get the optimal increment of stator voltage. The proposed algorithm is tested in the processor in the loop simulation with the Simscape model for the simulation of PMSM and the Jetson Xavier embedded device for the algorithm execution. The results show the ability of the algorithm to ensure the reference tracking and to keep the requested variables within their limits.**

***Index Terms*—model predictive control, permanent magnet synchronous motor, differential evolution optimization, general-purpose computing, graphics processing unit, parallel computing**

I. Introduction

Nonlinear model predictive control (NMPC) offers an exceptional power in the multivariable control and the implementation of the constraints. On the other hand, huge computational demands pose a challenge in the real-time computation, which is necessary in the case of permanent magnet synchronous motor (PMSM) control.

The research in the field mostly focuses on the acceleration of the optimization via multicore processors [1]–[3], field programming array (FPGA) [4]–[6], or the graphics processing unit (GPU) [7]–[9]. A huge disadvantage of these algorithms is their focus on a linear model. Therefore, the application of one of these algorithms to PMSM control would require some compromise, such as linearization of the model. This compromise would limit the advantages of using predictive control.

Predictive control based on the nonlinear model accelerated through parallel computation applied to PMSM was introduced in [10], [11]. These approaches were based on the finite control set MPC. Thus, the optimal control law was computed by the search across possible combinations of voltage source inverter. This approach requires even shorter sampling time than continuous control set methods.

Continuous control set methods compute the specific control value in form of input voltage based on the optimization. The applications of parallelism in this area are usually based on a family of optimization algorithms called population-based algorithms. Usage of particle swarm optimization (PSO) algorithm implemented on the FPGA was introduced in [12]. Differential evolution [13] offers one of the most simplistic ways of the agent movement and thus has lesser computational demands than more complex algorithms. Application of differential evolution optimization in NMPC was introduced in [14].

General-purpose computing on GPU is a modern technique used in many subjects of research, such as fluid simulations and neural networks, where exceptional parallel computing performance is needed. Population-based optimization methods are the type of problem, that can be treated similar way. The algorithm, proposed in this article, utilizes the native parallelism of the population-based methods and the parallel processing power of GPU. By this, the optimization problem can be solved in desired time and be used in PMSM control.

The remainder of this paper is organized as follows. Section II defines the PMSM speed control in form of an optimization problem. The following section introduces the proposed algorithm. In Section IV, the results of PIL simulations are shown and discussed. Finally, the paper is concluded in Section V.

II. Problem definition

The most important part of the nonlinear model predictive control is the proper formulation of the optimization problem. The standard form of the optimization problem [15]

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_0(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) \geq 0 \\ & \mathbf{h}(\mathbf{x}) = 0 \end{aligned} \tag{1}$$

consists of three parts. Cost function $f_0(\mathbf{x})$ should contain the requirements put on the control law, e.g., ensure the tracking of the reference signal. The nonequality constraints $\mathbf{g}(\mathbf{x})$ should describe the limits of the variables specific to a given plant.

Finally, the equality constraint $\mathbf{h}(\mathbf{x})$ represents a model of the controlled plant.

After the application of Euler method of discretization, using sampling period T_s, the discrete model of PMSM is

$$\begin{aligned}
i_d(k+1) &= (1 - T_s\frac{R_s}{L_d})i_d(k) + T_s P_p \frac{L_q}{L_d}\omega_m(k)i_q(k) + \\
&\quad + T_s\frac{1}{L_d}u_d(k) \\
i_q(k+1) &= (1 - T_s\frac{R_s}{L_q})i_q(k) - T_s P_p \frac{L_d}{L_q}\omega_m(k)i_d(k) - \\
&\quad - T_s P_p \Psi_{PM}\frac{1}{L_q}\omega_m(k) + T_s\frac{1}{L_q}u_q(k) \\
\omega_m(k+1) &= \omega_m(k) + T_s\frac{3}{2}\frac{P_p}{J}\Psi_{PM}i_q(k) + \\
&\quad + T_s\frac{3}{2}\frac{P_p}{J}\left(L_d - L_q\right)i_d(k)i_q(k) \\
\vartheta_m(k+1) &= \vartheta_m(k) + T_s\omega_m,
\end{aligned} \tag{2}$$

where

i_d, i_q	are stator current components in *dq* frame,
u_d, u_q	are stator voltage components in *dq* frame,
ω_m	is rotor mechanical angular speed,
ϑ_m	is rotor mechanical angular position,
T_l	is load torque,
P_p	is number of pole pairs,
R	is stator winding resistance,
L_d, L_q	are rotor inductance components,
Ψ_{PM}	is permanent magnet flux,
J	is the moment of inertia.

III. Control algorithm

In the proposed algorithm, we used an incremental type of controller. Therefore, the value of the control variable is given by its previous value and the increment. In the case of speed control of PMSM, the control variable is the stator voltage:

$$\mathbf{u}(k+1) = \mathbf{u}(k) + \mathbf{\Delta u}(k) = \begin{bmatrix} u_d(k) \\ u_q(k) \end{bmatrix} + \begin{bmatrix} \Delta u_d(k) \\ \Delta u_q(k) \end{bmatrix} \tag{3}$$

The flowchart in Figure 1 outlines the computation of the optimal increment. The algorithm consists of four parts. During the computation of the optima, three of them are computed iteratively. The necessity of the real-time computation led to the choice of ending condition based on the limitation of the number of iterations (NoI) by the number of generations (NoG). This choice achieves constant optimization time. The parts of the algorithm will be described in the next sections.

A. Initialization

For the initialization of every agent, we use pseudo-random generator with the specific seed for every call of optimization function. In the initialization, the algorithm generates a vector of the increment values. The value of increments is limited to $\langle -U_{DC}; U_{DC}\rangle$, where U_{DC} is the voltage of the supply. It is unnecessary to use a larger increment because it would lead to the exceeding of given constraint.

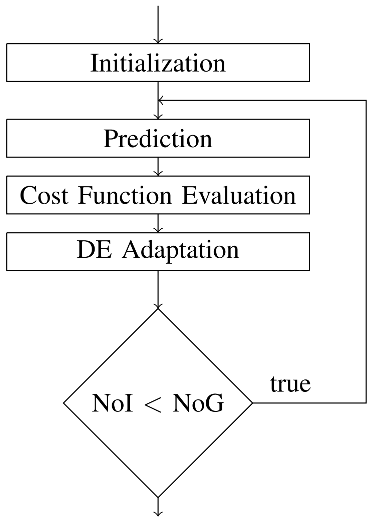

Fig. 1. Algorithm flowchart

Agent $a \in \mathbb{R}^{2N}$ of the optimization algorithm is then represented by the vector

$$a_i = \begin{bmatrix} \Delta u_d(k) & \Delta u_q(k) & \cdots & \Delta u_q(k+N-1) \end{bmatrix}^T, \tag{4}$$

where i is the index of a specific agent and N is the length of prediction horizon. The number of agents is dependent on the platform used for the computation. Choice of the number as the power of two is advised, because of the used optima search algorithm described further.

B. Prediction

By including the incremental type of controller (3) with the standard discrete DQ-frame model of PMSM (2), following model is used for the prediction of the future values of states

$$\begin{aligned}
i_d(k+1) = &\left(1 - T_s\frac{R_s}{L_d}\right)i_d(k) + T_s P_p\frac{L_q}{L_d}\omega_m(k)i_q(k) + \\
&+ T_s\frac{1}{L_d}\left(u_d(k) + \Delta u_d(k)\right) \\
i_q(k+1) = &\left(1 - T_s\frac{R_s}{L_q}\right)i_q(k) - T_s P_p\frac{L_d}{L_q}\omega_m(k)i_d(k) - \\
&- T_s P_p \Psi_{PM}\frac{1}{L_q}\omega_m(k) + \\
&+ T_s\frac{1}{L_q}\left(u_q(k) + \Delta u_q(k)\right) \\
\omega_m(k+1) = &\ \omega_m(k) + T_s\frac{3}{2}\frac{P_p}{J}\Psi_{PM}i_q(k) + \\
&+ T_s\frac{3}{2}\frac{P_p}{J}\left(L_d - L_q\right)i_d(k)i_q(k) \\
u_d(k+1) = &\ u_d(k) + \Delta u_d(k) \\
u_q(k+1) = &\ u_q(k) + \Delta u_q(k).
\end{aligned} \tag{5}$$

Including $\mathbf{\Delta u}$ in the equations of currents leads to the possibility of using a shorter prediction horizon.

In this stage, the parallelism of the evaluation of individual agents is used. This process is accelerated on the GPU. Results of prediction are stored for evaluation of the cost function.

C. Cost Function Evaluation

The cost function is an essential part of every optimization problem. In the case of model predictive control, the cost function should represent the demands put on the controller. The most basic demand on the controller is to ensure the tracking of the reference signal. In our case, the reference signal is requested mechanical angular speed $\omega_{m,r}$. We try to represent

$$\lim_{t\to\infty} \|\omega_{m,r} - \omega_m\| < \epsilon;\ \epsilon \to 0;\ \epsilon \in \mathbb{R}, \tag{6}$$

where $\| * \|$ is $\mathcal{L}_2$ norm, in the form of the cost function.

This can be achieved by using the quadratic form. Thus, the first part of the cost function is the cost of speed tracking error

$$c_{ST}(k) = \sum_{i=1}^{N} w_{ST}(\omega_{m,r}(k+i) - \omega_m(k+i))^2, \tag{7}$$

where w_{ST} is a weighting coefficient of tracking error.

Similarly, in the standard approach to speed control, the reference of the direct component of current is zero. In special cases, a non-zero direct component is used for field weakening. Otherwise, a non-zero value of direct part leads to unnecessary energy consumption and makes the control less efficient. The part of the cost function that deals with the direct part can be also represented as a quadratic form

$$c_{i_d}(k) = \sum_{i=1}^{N} w_{i_d}(i_d(k+i))^2, \tag{8}$$

where w_{i_d} is the weighting coefficient of the direct component value cost.

An optimized cost function is then the sum of the speed tracking part (7) and the direct component of current part (8). Matrix representation

$$\begin{gathered} c(k) = \sum_{i=1}^{N} \mathbf{x} \begin{bmatrix} \frac{w_{ST}}{2} & -\frac{w_{ST}}{2} & 0 \\ -\frac{w_{ST}}{2} & \frac{w_{ST}}{2} & 0 \\ 0 & 0 & w_{i_d} \end{bmatrix} \mathbf{x}^T \\ \mathbf{x} = \begin{bmatrix} \omega_{m,r}(k+i) & \omega_m(k+i) & i_d(k+i) \end{bmatrix} \end{gathered} \tag{9}$$

is sufficient for the unconstrained optimization problem. However, the defined problem consists of two specific constraints.

The first constraint is the limitation of the stator current vector, such that $\mathcal{L}_2$ norm of the vector is smaller than rated current I_R. Figure 2 shows the set of plausible values of the stator current. The well-known formula of the closed disk in Cartesian coordinates describes the given set perfectly. We can use this formula to derive the standard form of inequality constraint

$$g_1 = i_d^2 + i_q^2 - I_R^2 \leq 0. \tag{10}$$

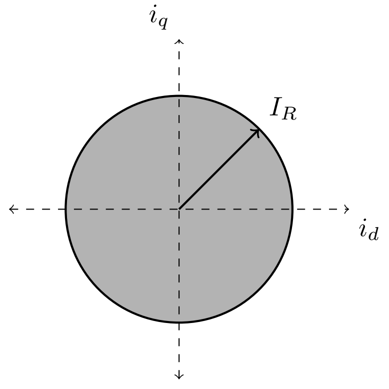

Fig. 2. Set of plausible values of the stator current

Similarly, the limitation of allowed input voltage is represented by a disk with radius given by the voltage of the supply U_{DC}. Thus, the inequality constraint of the voltage is

$$g_2 = u_d^2 + u_q^2 - U_{DC}^2 \leq 0. \tag{11}$$

To deal with the constraints, we use logarithmic barrier function [16] to expand the cost function. The constraints (10, 11) are represented as costs

$$c_I(k) = \sum_{i=1}^{N} -\log(I_R^2 - i_d^2(k+i) - i_q^2(k+i))w_I \tag{12}$$

$$c_U(k) = \sum_{i=1}^{N} -w_U \log(U_{DC}^2 - u_d^2(k+i) - u_q^2(k+i)), \tag{13}$$

where w_I and w_U are the weighting coefficients of the constraints costs.

Finally, used cost function representing specific agent a is given by the sum of the costs (9),(12) and(13)

$$C(k,a) = c(k) + c_I(k) + c_U(k). \tag{14}$$

D. Differential Evolution Adaptation

The section of Differential Evolution Adaptation consists of two parts: the search for minimal cost function and the agent movement.

1) Search for minima: In this stage, the demand for the count of agents in the power of two is utilized. We use the parallel algorithm based on the halving of the given array. In every instance, every thread performs one comparison and stores the result in a new array. The length of the new array is half of the original one. Figure 3 visualizes this parallel search. With this approach, every thread must perform P comparisons for the array of length 2^P. With the single thread computation, it would be $2^P - 1$. Thus, the search for minima is faster and more efficient.

2) Agent Movement: The method of movement of the agents is based on [17] with minor changes. Random choice of individuals is omitted, all agents are drawn towards the agent with the lowest cost function.

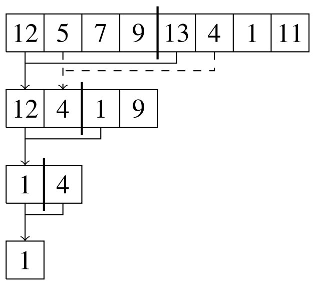

Fig. 3. Visualization of a parallel search for minima

Whether position changes or not is decided randomly by the rule

$$a_{i,j} = \begin{cases} a_{i,j} & \text{if} \quad r_{i,j} < t_r \\ a_{i,j} + s(a_{MIN,j} - a_{i,j}) & \text{if} \quad r_{i,j} \geq t_r \end{cases}, \quad (15)$$

where i is the index of a specific agent and j is the index of dimension which is supposed to change. Parameter $t_r \in (0;1)$ is threshold rate chosen during the implementation. Rate $r \in (0;1)$ is a uniformly distributed random variable generated during every agent movement stage. Parameter $s \in (0;1)$ defines the size of the step during movement.

IV. Simulation Results

This section covers the evaluation of the proposed algorithm. We tested the proposed algorithm using the PIL simulation with Jetson Xavier and the Simscape model. Firstly, the implementation and parameters of the used algorithm are described. Secondly, the results of the simulation are presented.

A. Implementation

Figure 4 shows the topology of the simulation. The UDP protocol was used for the communication between Simscape model and Jetson board. The vector $\mathbf{x}_m(k)$ transferred from the model to CPU consists of measured states $i_a(k)$, $i_b(k)$, $i_c(k)$, $w_m(k)$, $w_{m,r}(k)$ and $\vartheta(k)$. The ARM in Jetson board deals with the communication and transformation of measured states to states used in voltage increment optimization

$$\mathbf{x}_m(k) = \begin{bmatrix} i_d & i_q & w_m & w_{m,r} & u_d & u_q \end{bmatrix}^T, \quad (16)$$

where u_d and u_q contain the previous values of the action value. The vector $\mathbf{x}(k)$ is then stored in the GPU memory and the optimization is performed. After that, the increment is added to the previous value of the voltage and the result is sent to the Simscape model.

Table I shows the parameters of the Differential Evolution algorithm. Both the number of agents and the number of generations affect the time of the computation. The effect of the number of generations is obvious and for the systems with fast dynamics, such as PMSM, should be set as low as possible. The effect on the computation time caused by the number of agents is more significant during the search for the minima. The necessity to check all agents creates a bottleneck

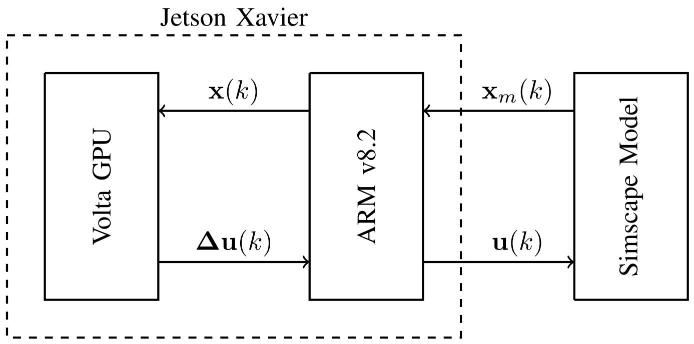

Fig. 4. Simulation topology

TABLE I
Parameters of Differential Evolution

Parameter	Value
Number of Agents	512
Number of Generations	20
t_r	0.5
s	0.6

in algorithm execution and slows it down. The usage of parallel search described previously mitigates this effect. Although, the number of agents is still limited by the used hardware. To achieve the best performance, the number of agents should not exceed the number of available threads, which is dependent on the number of CUDA cores of used GPU. The rule of thumb says that for every CUDA core 16 threads are available. We chose the values of the parameters t_r and s based on the recommendation in [18].

The parameters of the control algorithm are shown in the Table II. The lower bound of the length of the prediction horizon is given by the used model. The maximal value is limited by the requirement of short computation time. The maximal measured execution time, in this case, was $73\,\mu s$. Therefore, the sampling time was set to $100\,\mu s$.

B. Results

The algorithm was tested on the model of PMSM with the parameters in Table III. We tested the behavior of the controlled motor on defined requested angular speed. The initial ramp led to the value of rated angular speed. After settling, in $t = 0.1\,\mathrm{s}$, the load torque changed from $0\,\mathrm{N\,m}$ to $25\,\mathrm{N\,m}$. This tested the ability of the control algorithm to compensate the most common disturbance in motor control

TABLE II
Parameters of the Controller

Parameter	Value
w_{ST}	0.1
w_{i_d}	0.1
w_I	1
w_U	1
N	4
T_s	100 µs

TABLE III
PARAMETERS OF PMSM

Parameter	Value
R_s	$0.91\ \Omega$
L_d	$0.01\ \mathrm{H}$
L_q	$0.01\ \mathrm{H}$
Ψ_{PM}	$1.44\ \mathrm{Wb}$
Pp	3
J	$2.35 \times 10^{-3}\ \mathrm{kg\,m^2}$
ω_r	$120\ \mathrm{rad\,s^{-1}}$
U_{DC}	$560\ \mathrm{V}$
I_R	$5\ \mathrm{A}$

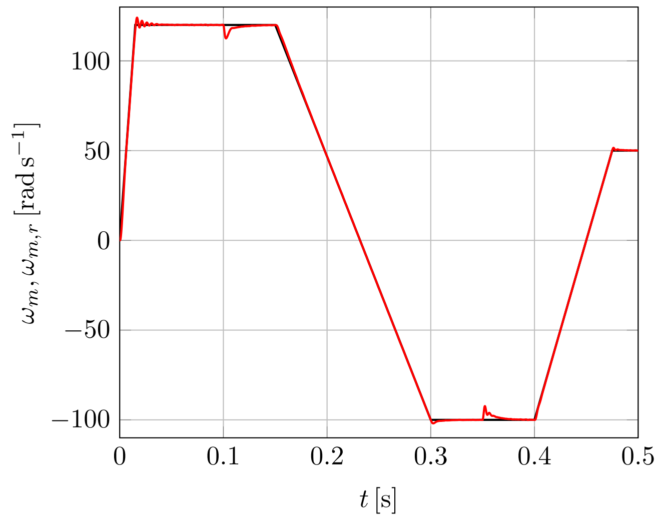

Fig. 5. Results of the experiment; black - reference, red - measured speed.

- load torque. In time $t = 0.15\,\mathrm{s}$, another ramp was used to change the value of angular speed to $-100\,\mathrm{rad\,s^{-1}}$. During this transfer, we tested the ability of the algorithm to slow the motor down and to change the sense of rotation. After settling, the value of load torque changed to $0\,\mathrm{N\,m}$. After that, the sense of rotation changes again and the angular speed reaches the value of $50\,\mathrm{rad\,s^{-1}}$.

Figure 5 shows the results of the experiment. The controller was able to ensure the reference tracking. Reaching the rated angular speed led to a $3\,\%$ overshoot and highly damped oscillations. After transition effects, the controller achieved zero control error. After the change of load torque, the controller was able to compensate it. The second transition resulted in a $2\,\%$ overshoot with no oscillations. Also, the controller was able to compensate the second change of load torque. The last transition had a similar result.

Figure 6 shows the currents during the experiment. The value of the direct component of current, i_d, fluctuated 0, as requested. The quadrature component, i_q, illustrates the behavior of angular speed. Initially, the value raised to generate the torque and to increase the angular speed. After the requested speed was reached, the value of i_q dropped. The presence of oscillations led to oscillations of angular speed. In time $t = 0.1\,\mathrm{s}$, the value of i_q raises again to compensate the change of the load torque. During the transition, the lesser value of i_q led to the requested change of the angular speed. The value of the current changed again during the second change of the load torque and the last transition.

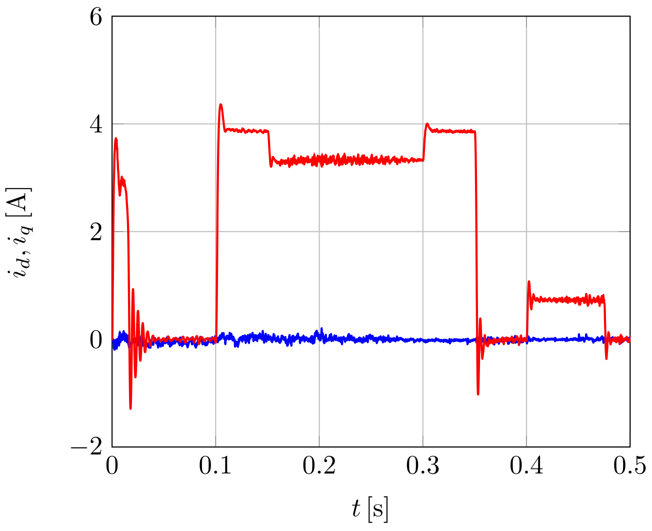

Fig. 6. Currents during the experiment; blue - i_d, red - i_q.

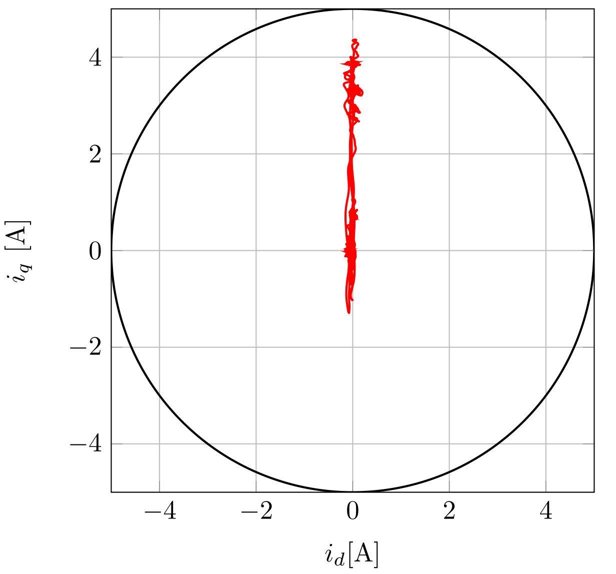

Fig. 7. Current in *dq* reference frame

Figures 7 and 8 show the ability of the controller to keep the requested variables within their limits. As mentioned before, in the case of the PMSM control, these variables are voltage and current. As the figures show, the controller was able to ensure it.

V. CONCLUSION

This paper proposed a novel approach to the application of model predictive control for speed control of permanent magnet synchronous motor. This approach draws on the well-known population-based method of optimization - differential evolution. The proposed algorithm utilizes native parallelism

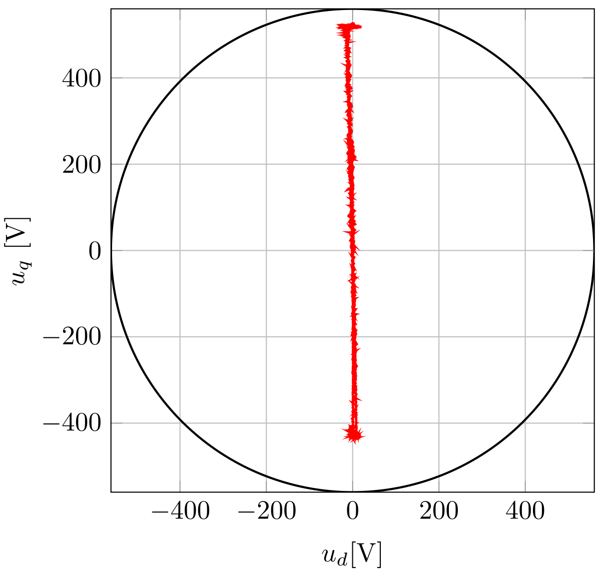

Fig. 8. Voltage in *dq* reference frame

of given method. The optimal increment of the stator voltage is computed using general-purpose computing on the GPU by parallel threads for every agent.

The proposed algorithm was tested in the PIL simulation with the Simscape model for the simulation of the voltage source inverter and PMSM, while the embedded device Jetson Xavier was used for the execution of the control algorithm. Settings of the algorithm, such as sampling period and the number of agents, were chosen according to the limitations brought by the real computation hardware.

The simulation showed the ability of the controller to ensure tracking of the requested angular speed and to compensate the disturbance signal. In Section IV, the ability to keep specified variables within defined limits was discussed.

During the writing of this paper, works on the implementing algorithm on the real physical system are performed. The implementation brings up new challenges. These are the design of a proper interface for the communication between drive and board with GPU and the mitigation of latency caused by the system operating the GPU board.

VI. Acknowledgment

The work has been performed in the project NewControl: Integrated, Fail-Operational, Cognitive Perception, Planning and Control Systems for Highly Automated Vehicles, under grant agreement No. 826653/8A19006. The work was co-funded by grants of Ministry of Education, Youth and Sports of the Czech Republic and Electronic Component Systems for European Leadership Joint Undertaking (ECSEL JU). The completion of this paper was made possible by the grant No. FEKT-S-20-6205 - ”Research in Automation, Cybernetics and Artificial Intelligence within Industry 4.0” financially supported by the Internal science fund of Brno University of Technology. The work was supported by the infrastructure of RICAIP that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857306 and from Ministry of Education, Youth and Sports under OP RDE grant agreement No. CZ.02.1.01/0.0/0.0/17_043/0010085.

References

[1] D. Soudbakhsh and A. M. Annaswamy, “Parallelized model predictive control,” in *2013 American Control Conference*, pp. 1715–1720, 2013.

[2] L. Ferranti and T. Keviczky, “A parallel dual fast gradient method for mpc applications,” in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 2406–2413, 2015.

[3] M. Kögel and R. Findeisen, “Parallel solution of model predictive control using the alternating direction multiplier method,” *IFAC Proceedings Volumes*, vol. 45, no. 17, pp. 369 – 374, 2012. 4th IFAC Conference on Nonlinear Model Predictive Control.

[4] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Bochat, “Parallel implementations of the fast gradient method for high-speed mpc,” *Control Engineering Practice*, vol. 33, pp. 22 – 34, 2014.

[5] A. Wills, A. Mills, and B. Ninness, “Fpga implementation of an interior-point solution for linear model predictive control,” *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 14527–14532, 2011.

[6] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, “Embedded online optimization for model predictive control at megahertz rates,” *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[7] L. Yu, A. Goldsmith, and S. Di Cairano, “Efficient convex optimization on gpus for embedded model predictive control,” in *Proceedings of the General Purpose GPUs*, pp. 12–21, 2017.

[8] A. K. Sampathirao, P. Sopasakis, A. Bemporad, and P. P. Patrinos, “Gpu-accelerated stochastic predictive control of drinking water networks,” *IEEE Transactions on Control Systems Technology*, vol. 26, no. 2, pp. 551–562, 2017.

[9] A. Sadrieh and P. A. Bahri, “Application of graphic processing unit in model predictive control,” in *Computer Aided Chemical Engineering*, vol. 29, pp. 492–496, Elsevier, 2011.

[10] M. Preindl and S. Bolognani, “Model predictive direct torque control with finite control set for pmsm drive systems, part 2: Field weakening operation,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 2, pp. 648–657, 2013.

[11] M. Kozubik and P. Vaclavek, “Speed control of pmsm with finite control set model predictive control using general-purpose computing on gpu,” in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pp. 379–383, IEEE, 2020.

[12] F. Xu, H. Chen, X. Gong, and Q. Mei, “Fast nonlinear model predictive control on fpga using particle swarm optimization,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 1, pp. 310–321, 2016.

[13] R. Storn and K. Price, “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[14] G. H. Negri, V. H. Preuss, M. S. Cavalca, and J. de Oliveira, “Differential evolution optimization applied in multivariate nonlinear model-based predictive control,” in *2015 Latin America Congress on Computational Intelligence (LA-CCI)*, pp. 1–6, IEEE, 2015.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, second ed., 2006.

[16] S. Boyd and L. Vanderberghe, *Convex optimization*. Cambridge university press, 2004.

[17] S. Das and P. N. Suganthan, “Differential evolution: A survey of the state-of-the-art,” *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[18] M. J. Kochenderfer, *Algorithms for optimization*. Cambridge ; London: The MIT Press, 2019.