Development of microprocessor, time optimized stepper motor driving algorithm

Łukasz Przeniosło and Marcin Hołub

***Abstract*—In the paper a stepper motor closed loop driver algorithm is presented which emphasizes the processor overhead minimization in which it is implemented, as well as rotor average torque and set position acquisition. The program was tested as a simulation in MATLAB environment and on a physical driver based on 32 bit microcontroller connected with a motor and an encoder. Exemplary results are introduced in the work.**

***Index Terms*—Stepper motor; motor driver; microprocessor; optimization; closed loop control.**

I. Introduction

STEPPING motors are a synchronous, brushless electric machines in which discrete electrical pulses applied to the motor windings generate discrete movement. They can be classified as follows (construction wise): variable reluctance motors (VRM), permanent magnet motors (PM) and hybrid motors. PM have relatively lower torque than VRM and cannot achieve high rotation speeds but are a lot cheaper. Hybrid motors share the efficient features of both of them. They have acceptable torque, maximum rotation speed and step resolution [1]. Stepper motors are used in various industrial applications where low power is required and low speed, torque, fast dynamics and precise positioning are important factors, for example in medical applications for accurate medicament dosage with peristaltic pumps or pipettes and as motion control actuators in dialysis equipment [2]. Those motors and their control system are in the region of interest of this work.

There are studies carried out on improving the stability of micro stepping techniques [3] as well as sensorless driving techniques to control the motor in an open loop [4]. However they do not focus on optimal usage of power inverter resources.

A comprehensive controller consists of many real time computation procedures in order to precisely drive a stepper motor in a closed loop, i.e. commutations setting, winding currents and rotor position/ speed measurements, controller set points adjustment, generation and following of position trajectories etc. [5] The "control loop" alone takes a lot of processor time in which it is implemented. If the device also has to execute other operations, such as communication or operating system tasks, the time spent in motor control routines has to be reduced significantly in order to meet all tasks timing requirements and prevent system starvation.

Łukasz Przeniosło and Marcin Hołub were with the Department of Power Systems and Electrical Drives, Westpomeranian University of Technology, ul. Sikorskiego 37, 70-313 Szczecin

Manuscript received 04.03.2017; accepted 22.05.2017

TABLE I
Features of stepper motor operating modes [6]

Mode	Wave		Fullstep		Halfstep		Microstep	
	Bi[a]	Uni	Bi	Uni	Bi	Uni	Bi	Uni
Torque	50%	25%	100%	50%	50-100%	25-50%	50%	25%
Driver complexity	fine	low	fine	low	high	high	high	high
Move smoothness	low	low	low	low	high	high	high	high
Resolution	low	low	low	low	fine	fine	high	high

[a] Bipolar/ unipolar winding powering mode

II. Stepper motor modes of operation

Stepper motors can be powered in several ways (modes). Each mode is only advantageous for the motion control system if used in a proper application. Table I [6] lists the parameters of each operating mode. As can be seen bipolar winding powering scheme is more efficient than unipolar one at all points (of course in cases where motor windings allow such operation). That is why only this scheme was implemented in the test controller. Because of system flexibility, a smart motor driver has to smoothly shift between different types of operation to provide an optimal rotor movement, thus an optimal load transportation in a practical setup.

Broadly available stepper motor drivers in an SoC (System on a Chip) package do not permit a smooth transition between operation modes (for example DRV8825 from Texas Instruments or A4989 from Allegro Microsystems). "Full-sized" stepper motor drivers available as separate device are out of the scope of this paper, as they do not require any driving algorithms, instead high level commands through a serial communication line are used for control.

III. The experimental setup

A. Top layer control scheme

In order to drive the motor in a developed device, a separate regulator paths have been used for position and current control (Fig. 1). The set angular position command (expressed in mechanical degrees) is fed into the position PID regulator. Its output feeds the velocity PID regulator. Feedback for those regulators are obtained from the incremental rotary encoder mounted on the motor shaft. The last parameter is the winding

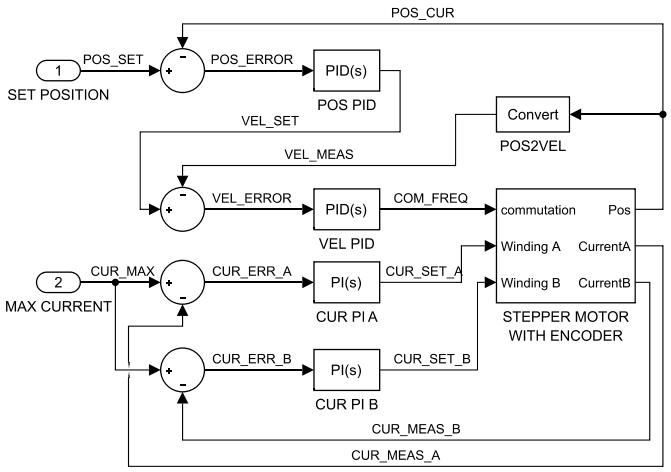

Fig. 1. Simplified control block diagram

current commutation frequency. The higher the frequency, the faster the stepper motor rotor rotates, however one has to take into account the torque – speed characteristics of the machine. In a parallel operation, motors winding currents are controlled by two dedicated PI controllers.

B. Preliminary calculations

In order for the procedure to work, some input parameters have to be provided for the initialization script.

- Number of full steps per revolution (FS): this parameter describes how many full steps have to be performed in order for the stepper motor shaft to rotate by 360 degrees. Common values for hybrid stepper motors are 200, 360 and 400.

- Maximum micro steps per full step (MS): this value defines in how many sub-steps a full step is taken. Together with full steps per revolution this parameter decides on the final stepper motor resolution (Equation 1). This resolution is not related to the used encoder resolution.

$$M_{RES} = FS \cdot MS \tag{1}$$

  Where:
  - M_{RES} – motor resolution,
  - FS – full steps per revolution,
  - MS – micro steps per full step.

  It is preferable if the MS parameter is a power of 2 in order to make the calculations easier and avoid using fraction numbers in the microcontroller.

- Maximum commutation frequency [Hz] (CM): This variable is dependent on the application complexity and motor torque – speed characteristics. It decides on how fast the motor windings current can be commutated. Commutation procedure is executed in a periodic task that provides the higher processor time overhead the more often it is executed. Thus if the processor is fast (high clock frequency) and free of other time consuming tasks, the maximum commutation frequency can be high (keeping in mind that required torque has to be in that frequency range). On the other hand, if the stepper motor control is just one of many other tasks executed by the processor, too high CM can lead to system starvation. Because of this, an appropriate CM has to be chosen empirically to fit the applications needs.

In the next step, the maximum motor velocity [RPM] (MMV) is calculated (Equation 2). For control loop, it means how fast can the motor shaft be rotated and keep satisfactory torque.

$$MMV = \frac{60 \cdot CM}{FS} \tag{2}$$

MMV should not be higher than nominal velocity under load. This parameter is obtained from the motor datasheet. If Equation 2 gives to high value, one should lower the CM or transform the equation for known MMV to obtain CM (Equation 3).

$$CM = \frac{MMV \cdot FS}{60} \tag{3}$$

Now the amount of gears will be defined (Equation 4). Gears in the algorithm are an electrical equivalent of mechanical transmission system. The difference is that for the motor, the commutation frequency is being changed when the gear changes, not the shaft rotation speed.

$$G_{NR} = \log_2(MS) + 1 \tag{4}$$

Gears change automatically depending on the current rotor velocity. Velocity gear change thresholds (GC_{th}) need to be calculated before the control loop starts. Two different tables are generated, each for when the motor is accelerating and decelerating. This prevents the controller from entering velocity areas where gear is changed too often (an equivalent of a hysteresis).

Listing 1
GEAR CHANGE THRESHOLD GENERATOR ALGORITHM

```
for i = 1 : G_NR
    mmvMaxStages(i) = MMV / (2^(i - 1));
end

mmvMaxStages = fliplr(mmvMaxStages);
mmvMaxStagesUp = mmvMaxStages * 1.1;
mmvMaxStagesDown = mmvMaxStages * 0.9;
```

Listing 1 shows how are the thresholds calculated in MATLAB code. The variable G_{NR} is obtained from Equation 4 and MMV is obtained from equation 2. A vector of generic speed values is generated at first (mmvMaxStages). The vector is flipped in order for higher gears to indicate higher speeds (this step is done for the purpose of clarification of the procedure). Then each element is multiplied by a factor bigger than one for the acceleration vector and less than one for deceleration. In

TABLE II
ROUNDED VELOCITY [RPM] THRESHOLD VALUES GENERATED FOR ACCELERATING AND DECELERATING OF THE ROTOR, FOR CM = 5000 HZ AND MMV = 1500 RPM

Gear	0	1	2	3	4	5	6	7
Microsteps	128	64	32	16	8	4	2	1
Accel [RPM]	13	26	52	103	206	412	825	1650
Decel [RPM]	11	21	42	84	169	338	675	1350

this example acceleration values are 10 % higher than generic ones and deceleration values are consequently 10 % lower (20 % total hysteresis). Table II presents exemplary results for the tested setup.

C. Control loop calculations

After performing all preliminary calculations the motor can be started and the actual commutation (stepping) frequency can be calculated in the control loop, along with the actual gear number.

The experimental setup control loop frequency was set to 1000 Hz. New gear is then obtained every 1 ms from Equation 5. The result is then rounded down to an integer value.

$$G = G_{NR} - \log_2(\frac{MMV_{A/D}}{CV}) - 1 \tag{5}$$

CV variable stands for current rotor velocity [RPM]. It is obtained by calculating the derivative of the rotor position (Equation 6), which is indirectly read from the attached encoder mounted on the shaft.

$$CV = \frac{\Delta S}{\Delta t} \tag{6}$$

Where:

- ΔS – the distance difference between current position and the position obtained in the last Δt time quantum.,
- Δt – time rate at which position measures are taken. In this setup it is 1 ms.

$MMV_{A/D}$ is the last item of the acceleration or deceleration gear threshold vector. The chosen vector is based on the dynamic parameter of the movement (positive or negative velocity error indicates that accordingly in Fig. 1). Using the control loop routine, after obtaining the gear number G, the commutation frequency can be calculated (Equation 7).

$$CC = \frac{M_{RES}\frac{CV}{60}}{2^G} \tag{7}$$

Current commutation frequency CC [Hz] indicates at what rate new steps (or microsteps, which value depends on the G variable) are taken by the motor.

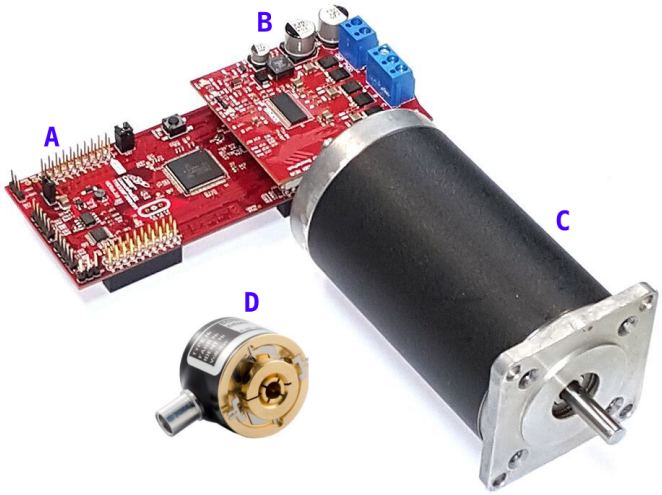

Fig. 2. One of the control systems used for algorithm implementation; A) 32 bit DSP microcontroller based board, B) customized power electronics daughter board, C) 23D-6309C stepper motor (American Precision), D) externally coupled SCH24 rotary incremental encoder (Scancon)

D. Experimental hardware setup

The tests performed for the purpose of this paper were conducted using a universal motor driving platform. Fig. 2 shows the device and the peripherals. This driver is a prototype built upon Launchxl-F28377S Texas Instruments board. The hardware is capable of driving two motors simultaneously without any lags, thanks to the procedure presented in the paper. For the test only one motor was used. The algorithm software layer itself is hardware independent.

E. Experimental results

The data presented in the example diagrams was acquired from the driver (Fig. 2) in real time using a serial protocol, while the motor was running. Each sample was taken at 10 ms rate. The data was then plotted using MATLAB. Fig 3 presents the first test performed at a medium CC of 5000 Hz. V_{max} and A_{max} are the maximum velocity and acceleration of the trajectory generator, which is feeding the controller (Fig. 1) with SET POSITION in the control loop. The top plot shows that the real position is acquired very precisely and smoothly (0.018° resolution). The velocity regulator has little jitter (around 0.4 RPM per second), but mostly overlaps the set point curve. Two bottom plots show the calculated gear and commutation frequency variables respectively. For this set of parameters the driver uses the fifth gear at the maximum velocity of 200 RPM, thus it is never leaving the micro stepping mode. It can be noticed that the gears are changed fast during the beginning of acceleration and the end of deceleration phases respectively. Consequently, the stepping frequency changes in the same fashion, never exceeding the 110 % of MVV (for acceleration phase). The highest commutation frequency noted in this test was 5492 Hz.

Fig. 4 shows a similar test where six movement commands are executed as well. This time the maximum commutation

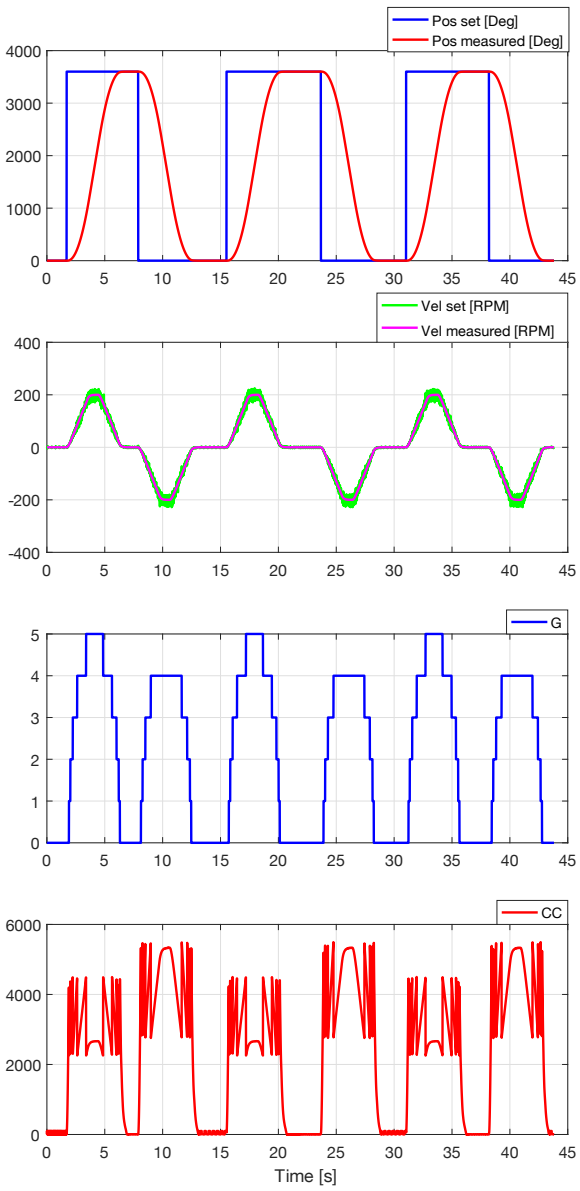

Fig. 3. Example plots for CM = 5000 Hz, V_{max} = 200 RPM, A_{max} = 100 RPM^2

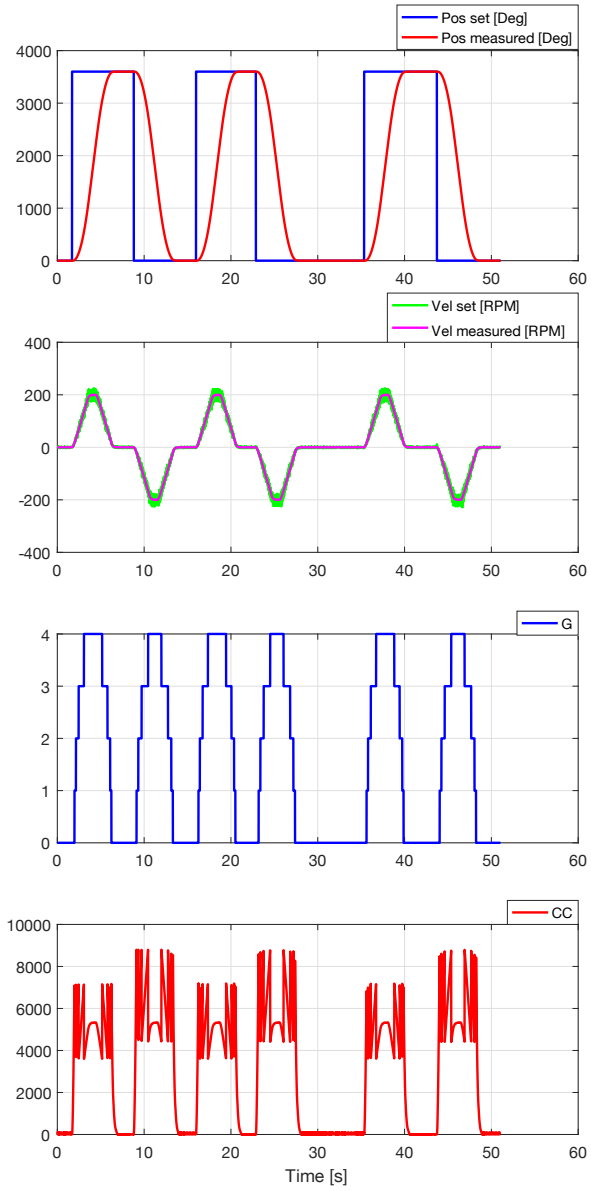

Fig. 4. Example plots for CM = 8000 Hz, V_{max} = 200 RPM, A_{max} = 100 RPM^2

frequency MVV was set to 8000 Hz instead of 5000 Hz. It can be noticed how the highest reachable gear is lower than the one from Fig. 3. It means that the motor is driven with higher micro stepping resolution for longer period of time during the rotation, thus the movement is smoother. This is of practical importance as microcontroller resources are heavily used. A new step is made at 125 μs rate instead of 200 μs, thus, other system procedures are interrupted more often.

Operating on a low gear has an additional drawback. As the stepper motor is a permanent magnet brushless machine, its torque curve decreases along with increasing rotation velocity [6]. In the algorithm, the driver operates in the micro stepping mode for the most of the gears but last. At the last gear, controller can shift into full step mode, as the amount of micro steps is equal to 1. In the full step mode the torque is twice as high as in micro stepping mode (Table I).

In order to force the algorithm to use the full gear range, one has to feed it with low MVV value. Fig. 5 presents exemplary results for MVV set to 1000 Hz. The maximum possible gear number is acquired near the V_{max} as expected. One can notice however that the velocity set curve has a lot higher jitter than in the Fig. 3 and Fig. 4. Again, because

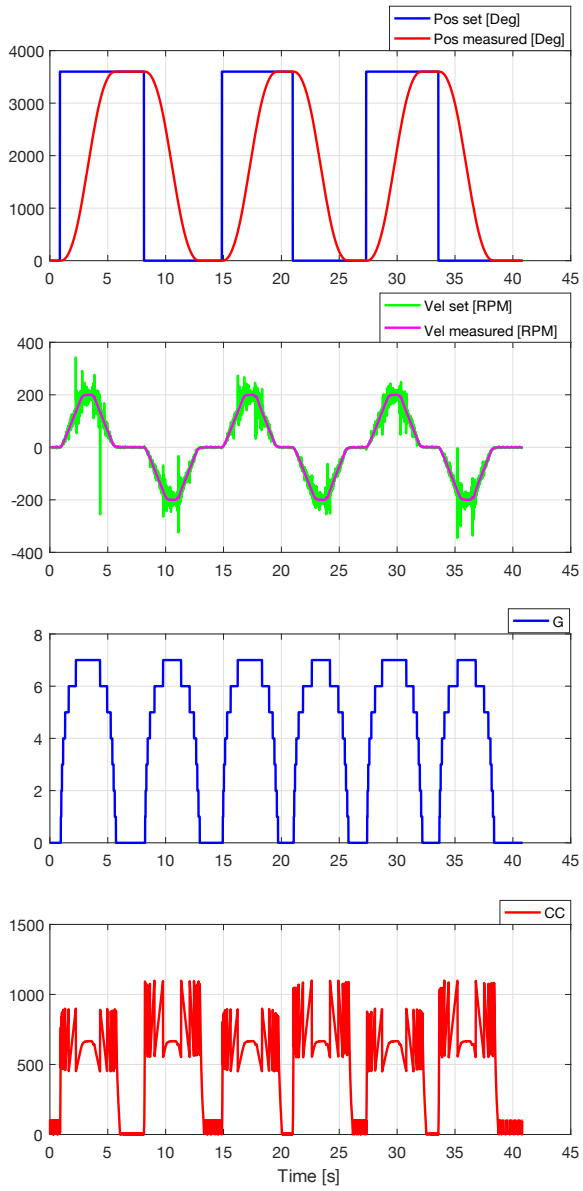

Fig. 5. Example plots for CM = 1000 Hz, V_{max} = 200 RPM, A_{max} = 100 RPM^2

of their construction stepper motors suffer from high torque ripples at low speeds [7].

Table III presents how the resources are used by three most resource heavy functions in the program from Fig. 3. The measurement was made by checking how often the PC (program counter) was located within a certain function address. For example in Table III, program counter was monitored 18674 times (on 70100 total samples) within function motSetNextCCR, which is used for stepper motor commutation. The measurement was taken using True Studio software from Atollic (SWV Statistical Profiling tool).

TABLE III
AMOUNT OF RESOURCES USED IN AN EXAMPLE PROGRAM FOR CM = 5000 HZ, V_{max} = 200 RPM, A_{max} = 100 RPM^2, PC SAMPLES POOL = 70100

Function	Resources in use	No. of PC samples
motSetNextCCR()	26,64 %	18674
taskcontrolLoop()	15,35 %	10760
startSystemTask()	6,17 %	4325

TABLE IV
AMOUNT OF RESOURCES USED IN AN EXAMPLE PROGRAM FOR CM = 1000 HZ, V_{max} = 200 RPM, A_{max} = 100 RPM^2, PC SAMPLES POOL = 79320

Function	Resources in use	No. of PC samples
taskcontrolLoop()	26,83 %	19889
startSystemTask()	14,10 %	8969
motSetNextCCR()	8,21 %	6086

Similar results are shown in Table 4 for the program from Fig 5. It can be noticed that the percentage ratio between the function motSetNextCCR and the other two changed significantly. The former function is executed less often than when the stepping frequency was set to 5000 Hz, leaving more resources for other system tasks and interrupt routines.

IV. CONCLUSIONS

The results show that the microcontroller time optimization procedure during automatic stepper mode control given in the paper is operational and gives good results. It is configurable and should be set adequately to the application. If high precision and smooth motion is desired, one should set maximum commutation frequency. If high torque or low resources are important, then low commutation frequency should be used.

The algorithm was used in several industrial applications with success, including applications like 3D printers and Pick and Place machines.

REFERENCES

[1] Łukasz Przeniosło, "Universal smart electric motors controller for industry applications," *Westpomeranian University of Technology*, 2016.
[2] M. Bendjedia, Y. Ait-Amirat, B. Walther, and A. Berthon, "Position control of a sensorless stepper motor," *IEEE Transactions on Power Electronics*, vol. 27, no. 2, pp. 578–587, Feb 2012.
[3] D. R. Gaan, M. Kumar, and C. G. Majumder, "Variable rate based microstepping of stepper motor using matlab gui," in *2017 Indian Control Conference (ICC)*, Jan 2017, pp. 385–390.
[4] M. Bendjedia, Y. Ait-Amirat, B. Walther, and A. Berthon, "Sensorless control of hybrid stepper motor," in *2007 European Conference on Power Electronics and Applications*, Sept 2007, pp. 1–10.
[5] K. Zawirski, J. Deskur, and T. Kaczmarek, *Automatyka napędu elektrycznego*, 1st ed. Wydawnictwo Politechniki Poznańskiej, Poznań 2012.
[6] J. Przepiórkowski, *Silniki elektryczne w praktyce elektronika*, 2nd ed. BTC, Warsaw 2012.
[7] M. Piccoli and M. Yim, "Cogging Torque Ripple Minimization via Position-Based Characterization," *Robotics: Science and Systems X*, 2015.

Łukasz Przeniosło was born in Gorzów Wielkopolski, Ponand, in 1990. He received the M.S. Degree from Electrical Department, Westpomeranian University of Technology, Szczecin, in 2016. At the moment he is a Ph.D. student in Department of Power Systems and Electrical Drives, Westpomeranian university of Technology, Szczecin.

Marcin Hołub was born in Szczecin, Poland, in 1976. He received the M.S. Degree from the Electrical Department, Westpomeranian University of Technology, Szczecin, in 2000, the Ph.D. degree from the Westpomeranian University of Technology in 2005 and habilitation from the Westpomeranian University of Technology in 2016 He was with the Braunschweig University of Technology, Braunschweig, Germany, for three years. He is currently a member of a Plasma Research Team with the Electrical Engineering Department, and a member of the Executive Board of the BalticNet-PlasmaTek Network. He authored over 50 articles covering the fields of control systems, power electronics and plasma supply systems.