Direct Particle Swarm Repetitive Controller with Time-Distributed Calculations for Real Time Implementation

Piotr Biernat, Bartlomiej Ufnalski, and Lech M. Grzesiak

Institute of Control and Industrial Electronics,
Faculty of Electrical Engineering, Warsaw University of Technology,
75 Koszykowa Str., Warsaw 00-662, Poland
{piotr.biernat, bartlomiej.ufnalski, lech.grzesiak}@ee.pw.edu.pl
http://www.ee.pw.edu.pl

Abstract. In this paper, real-time implementation of recently developed direct particle swarm controller for repetitive process is presented. The proposed controller solves the dynamic optimization problem of shaping the control signal in the voltage source inverter. The challenges in real time implementation come from limited sampling period to evaluate a candidate solution. In this paper, the solution of PDPSRC time-distributed calculation is presented. This method can be implemented using digital signal controllers (DSC).

Keywords: particle swarm optimization (PSO), time-distributed calculations, real-time implementation

1 Introduction

The particle swarm optimization is a population-based evolutionary algorithm (EA) modelled after biological swarming such as bird flocking, fish schooling, herding of land animals or collective behavior of insects. This method is usually applied to find the best solution of static, non-changing problems. However, many real world problems are dynamic i.e. variable load of the inverter. In this case, the PSO algorithm has to track the progression of the function optimum through the space as closely as possible. In [1] the plug-in direct particle swarm repetitive controller (PDPSRC) for the sine-wave constant-amplitude constant-frequency voltage source inverter (CACF VSI) is presented. To find the best control signal, the swarm itself is a repetitive controller cooperating in parallel with the controller shaping the signal behaviour along the pass (in the p-direction). The algorithm maintains a swarm of individuals called particles, where each particle stores all samples of the control signal along the pass. During the exploration process, all particles are

rated according to user-defined cost function. The detailed explanation of the proposed algorithm is presented in [8].

In the basic control algorithm [2], all the samples have to be remembered and rated in the end of pass or trial. Although the calculations to update one sample of the particle are not computational burden, updating all samples in all particles in off-the-shelf industrial microcontrollers, most of the time cannot be completed within one sampling period.

The real-time implementation in microcontrollers of the PSO algorithm [3, 4] is much more complicated compared to a simulation model [5, 6]. There issues that need to be considered such as limited calculation time (due to sampling time) and size of memory. In view of these drawbacks, this paper introduces PDPSRC time-distribution calculation which prevent from conducting all calculations in one sample time. The main idea is to split up processes that can be calculated incrementally, e.g. calculating mean squared error (MSE), and do not require knowledge of a whole signal. The accuracy of the proposed code has been verified in a numerical simulation.

2 Particle Swarm Optimization Time-Distributed Calculations

In a particle swarm optimization, each particle represents a possible solution. Particles travel through search space in cooperation with other particles and are rated by the user defined cost function. The position of the particle is then determined based on the best solution found for the particle itself and the best solution in its surroundings.

In a conventional approach, updates of all particles are conducted after the last sample of the swarm based on gathered particles' response data. Therefore, the real-time system requires huge computing power. However, instead of using more powerful hardware, the PDPSRC algorithm should be organized in such a way that the calculation will be performed in every sample time.

The major problem is how to organise the algorithm computational burden variables such as fitness function (calculated for a whole particle). The proposed solution is to divide bigger problems into smaller ones calculated in every sample time. The flow chart consisting of time-distributed PDPSRC calculations is presented in Fig. 1. The colours green, yellow and red indicate respectively calculations performed in each sample time, after each particle and after a whole swarm.

The particles start with some random position and random speed. Next, the control signal is applied to the plant and the response is gathered. For each particle, the algorithm has to calculate their fitness:

- $VE_j(p)$ – error component, difference between the reference voltage and filter capacitor voltage for *j-th* particle *p-th* sample,
- $D_j(p)$ – control dynamics component of *j-th* particle *p-th* sample,
- $Fitness_j$ – fitness function of *j-th* particle

The pseudo-code is presented in Algorithm 1.

```
VE_j=VE_j(p-1)+(u^err)^2
D_j=D_j(p-1)+(q_j(p)-q_j(p-1))^2
if (end of particle){
VE_j=VE_j*•_1
D_j=D_j*•_2
Fitness_j= VE_j+D_j*•}
```

Algorithm 1: Calculation of fitness function

The first two functions, $VE_j(p)$ and $D_j(p)$, are calculated by incrementing them in each sample time and β is the penalty factor for control dynamics. After the last sample of the particles, the fitness function is calculated by summing them. Next, the coefficients are set to zero for the next particle. Factor δ_1 is the reciprocal of the number of samples in a single particle and δ_2 is also the reciprocal of the number of particle samples, but without the first one.

A diversity [7] calculation of the swarm can also be distributed in every single sample time. First, the algorithm has to find the maximum and minimum of the particles' position.

```
if (swarm_j(p)>div_max(p))
    div_max(p)= swarm_j(p);

if (swarm_j(p)<div_min(p))
    div_min(p)= swarm_j(p);
```

Algorithm 2: Searching maximum and minimum of particles position.

Each *p-th* sample in all of the particles is compared to find the highest and lowest position. After every swarm iteration the diversity is calculated in every sample time, based on previous iterations *div_min* and *div_max* variables.

```
swarm_div=0.5*(div_max(p)-div_min(p);
if(swarm_div<diversity_limit)
    swarm_dir=-1.0;
else(swarm_div>=diversity_limit)
    swarm_dir=1.0;
```

Algorithm 3: Calculation of diversity and choosing the repel or attract mode

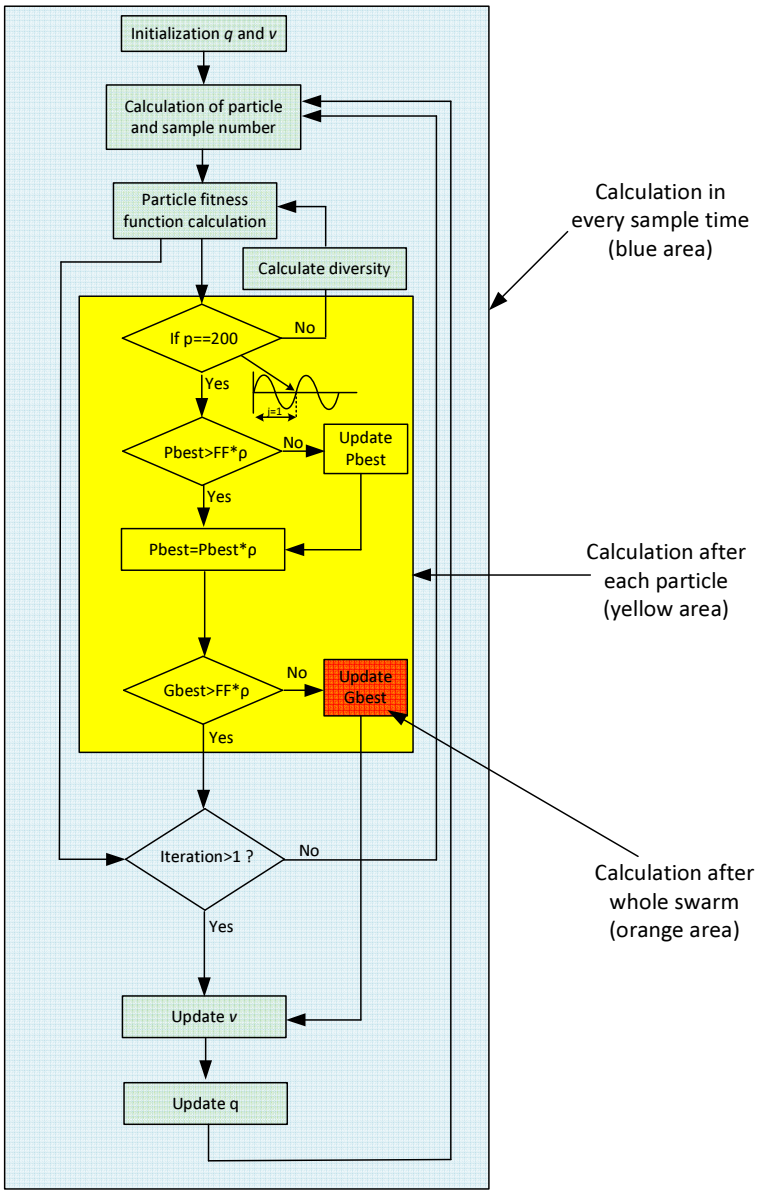

Fig. 1. PDPSRC time-distributed calculations

The result is then compared by the user defined diversity threshold and switching between attract (*swarm_dir=1)* and repel *(swarm_dir=-1)* modes. After the last sample of the particle, the algorithm has to check if the actual cost function is better than the corresponding personal best fitness. If the proposed solution is better the table consisting of the best solutions is updated, if not the actual solution evaporate at the constant rate ρ.

```
if (end of particle)
  if (fitness_j is less then pbest_FF*•)
    Pbest_table.update();
    pbest_FF = fitness_j
  else
    pbest_FF = FF_j*•
  end
end
```

Algorithm 4: Updating new personal best solution or evaporation actual solution

In the same sampling period the algorithm is checking if the current cost function value is better than the best solution found so far in the given swarm.

```
if (fitness_j < Gbest_FF)
    Gbest=pbest_j;
    Gbest_FF= fitness_j;
end
if (last sample of swarm)
Gbest_table.update();
end
```

Algorithm 5: Checking if actual solution is better than previous one

Data stored in global best solutions should be updated after the last sample of the swarm in synchronous mode.

In the first loop, the algorithm only tests the proposed solutions and gathers the required data: fitness function, personal best solutions, global best solution and swarm diversity. From the second loop, the speed and position update rules mechanism is applied. The position of the particles is then determined based on the best solution for the particle itself and its surroundings. Updated speed for the corresponding particle matrix element is calcu-

lated in every single sample period. The new velocity and position is calculated as follows

```
if(second loop or more){
  update velocity v_j(p+1)=eq.(1)
  velocity clamping v_j(p)=V_clmp(p)
  update position q_j(p+1)=q_j(p)+v_j(p)
}
```

Algorithm 6: Particles position and velocity with clamping updates

The new velocity is calculated by :

$$v_j^{p+1} = c_1 v_j^p + c_2 r_q \delta\left(q_j^{pbest} - q_j^p\right) + c_3 r_g \delta\left(q_j^{gbest} - q_j^p\right) \quad (1)$$

where: c_1, c_2 and c_3 are the inertia, cognitive and social coefficients, and r_q, r_g are random numbers between 0 and 1. Random numbers are uniformly distributed in the unit interval and are generated in each sample for each particle in every iteration. There are many different solutions for PSO randomisation like: the particle-wise randomization, the dimension-wise randomization, the list-based PSO, the list-based PSO with a memory-cheap table [8].
The algorithm of tracking the minimum of the cost function is then repeated until the system is in operation. A graph that represents the distribution of the calculation in time is presented in Fig. 1. The green tags represents the calculation carried out in every algorithm entrance to PWM interruption, the yellow ones constitute the calculations after every particle, and red ones after each trial.

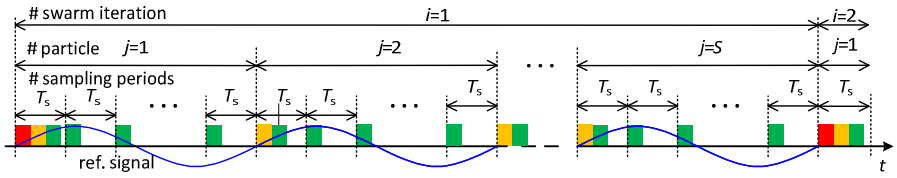

Fig. 2. Time-distributed calculations for real time implementation

In table 1 the parts of the PDPSRC are presented with explanation in which moment they can be calculated.

Table 1. Time-distribution of PDPSRC algorithm

Function	**Fig. 1 marker**	**Time-distributed**
Fitness function	green	Yes (calculated in each sample)
Swarm diversity	green	Yes (calculated in each sample)
Velocity update	green	Yes (calculated in each sample)
Position update	green	Yes (calculated in each sample)
Find personal best	yellow	Partly (calculated after each particle)
Personal best update	yellow	Partly (calculated after each particle)
Find global best	yellow	Partly (checking after each particle)
Global best update (synchronous mode)	red	No (updated after whole swarm)

3 Model of the Plant and Numerical Experiment Results

The time-distributed PSO code has been tested using the simulation model in Matlab/Simulink and Plecs toolbox [9] presented in Fig. 2. A plug-in direct particle swarm repetitive controller has been used as a k-direction controller (pass to pass) and it is required to support it with a controller in the p-direction (sample to sample). As a non-repetitive controller (p-direction), the full-state feedback has been implemented to increase damping in the plant.

In the simulation model the PWM voltage source inverter is approximated using a controlled voltage source with the LC filter on the output. Selected parameters of the model and swarm have been given in Table 3 and 4.

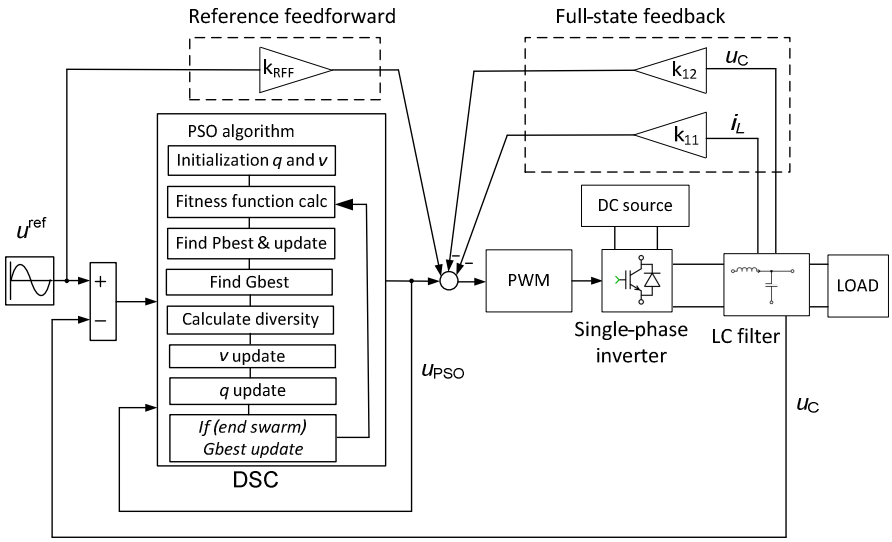

Fig. 3. Schematic diagram of the proposed repetitive control system.

Table 2. Parameters of the simulation model

Parameter	Symbol	Value
DC-link Voltage	U_{DC}	450 V_{DC}
Reference voltage	U^{ref}	325 V
Reference frequency	f^{ref}	50 Hz
Filter inductance	L_f	300 µH
Filter capacitance	C_f	160 µF
Switching frequency	f_{sw}	10 kHz
Load power	-	ca. 5kW

Table 3. Parameters of the swarm

Dimensionality of the problem	α	200
Number of particles	S	25
Evaporation constant	ρ	1.1
Penalty factor	β	0.25
Velocity clamping level	V_{clmp}	9.0

The behaviour of the plant with full state feedback and without PSO controller is depicted in Fig. 3. The simulation model has been tested under two kinds of loads:

1. Resistive load (ca. 3kW), applied for first 150s,
2. Diode rectifier (ca. 2kW), applied from 150s to 500s.

The algorithm starts with the near zero initial condition. First, the tests are conducted with only the resistive load. Next instead of the resistive load, the diode rectifier is switched on. The comparison between a system with and without the PDPSRC algorithm is presented in Fig. 3 and 4. The evolution of the root mean square error for the PSO-based repetitive is presented in Fig. 6.

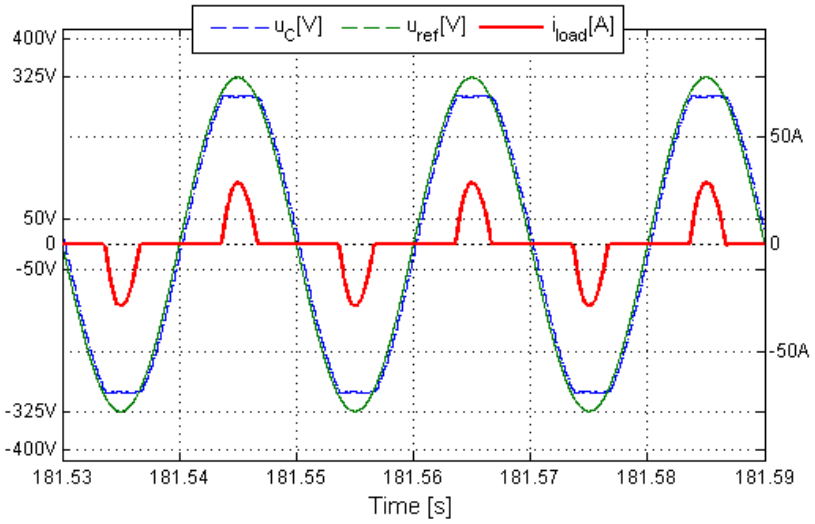

Fig. 4. Reference and capacitor voltage without PDPSRC (only FSF is implemented)

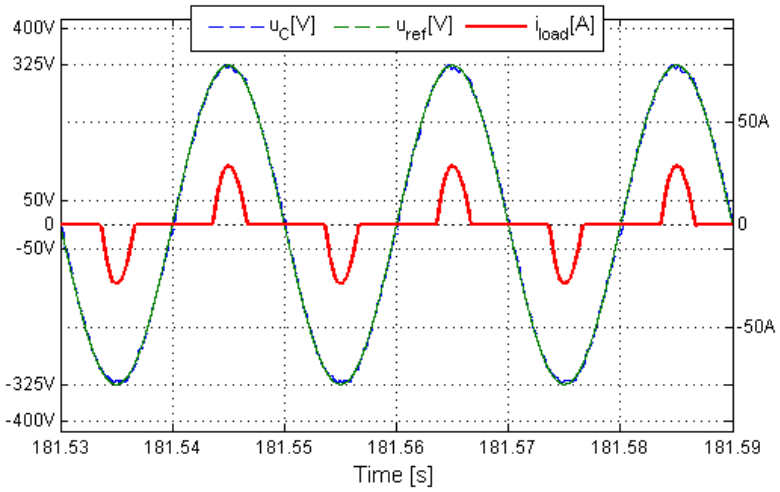

Fig. 5. Reference and capacitor voltage with PDPSRC

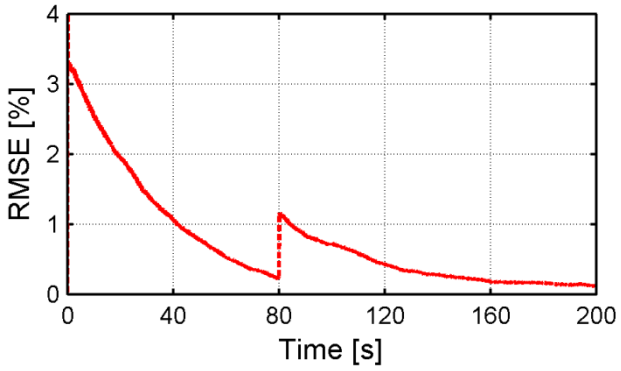

Fig. 6. Evolution of the root mean square error for the PSO-based repetitive controller (in percent of reference voltage)

The PSPSRC time–distributed calculations reduce the computational burden of the
algorithm. Implementation tests on the TMS320F2812 controller using code composer studio [10] show that the time required for calculations in one sample period does not exceed 20μs (green+yellow+red tags). Therefore, time-distributed calculation makes it possible to implement the algorithm in off-the-shelf industrial microcontrollers. During the implementation procedure, an additional problem shows up. This problem is connected with the controller's memory size, which should be large enough to store about 15000 samples (Table 4).

Table 4. Data need to be stored in PDPSRC (for sampling time 10kHz, reference frequency 50Hz, swarm size 25 particles).

Parameter	Type	Number of samples
Swarm position	Float	5000
Swarm velocity	Float	5000
Personal best	Float	5000
Global best	Float	200
Diversity	Float	400
Others	Float + integer	ca. 100
	TOTAL	ca. 15700

For example the digital signal controller eZdspTMS320F2812 has a 18k-word internal memory and additional 64K words off-chip SRAM memory. The float type variable needs 4 bytes of memory space. Thus, for all variables the required space must be at least 60kB. Therefore, to solve this issue the controller external memory must be used, on which the swarm samples will be stored.

4 Conclusions

The plug-in direct particle swarm controller with time-distributed calculations for constant-amplitude constant-frequency inverter with LC output filter has been presented. By distribution of the algorithm calculation in the sampling period, real-time implementation of the PDPSRC algorithm in microcontrollers can be realized. It has been proved that without such an organized code, would not be possible its implementation in typical off-the-shelf industrial microcontrollers such as eZdspTMS320F2812.
An experimental tests of shaping the voltage control signal in physical CACF VSI are planned to be continued.

Acknowledgements. The research work was supported by the statutory funds of the Electrical Drive Division for 2014.

References

1. Ufnalski, B., Grzesiak, L.M.: A plug-in direct particle swarm repetitive controller for a single-phase inverter. Electrical Review (Przeglad Elektrotechniczny, paper in English, open access at pe.org.pl) pp. 1-6 (2014)
2. Eberhart, R.C., Shi, Y., Kennedy, J.: Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation). Morgan Kaufmann Publishers, 1st edn. (Apr. 2001)

3. Liu, W., Chung, I., Cartes, D., Leng, S.: Real-Time Particle Swarm Optimization based Current Harmonic Cancellation based PMSM Parameter Identification. Power & Energy Society General Meeting, 2009. PES '09. IEEE
4. Liu, W., Liu, Li., Cartes, D.: Efforts on Real-Time Implementation of PSO. Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE
5. Huang J., Li H., Guo J.: Application of PSO in the improved real-time evolution. 11th International Conference on Hybrid Intelligent Systems, 2011.
6. Liu L., Cartes D.: Real time implementation of particle swarm optimization based model parameter identification and an application example. IEEE Congress on Evolutionary Computation (CEC 2008)
7. Zhan Z., Zhang J., Shi Y.: Experimental study on PSO Diversity. Third International Workshop on Advanced Computational Intelligence 2010
8. Ufnalski, B., Grzesiak, L.M.: A comparative investigation on different randomness schemes in the particle-swarm-based repetitive controller for the sine-wave inverter. 7th International Conference on Intelligent Systems IS 2014
9. Plecs simulation software for Power Electronics: http://www.plexim.com/, ver.3.5.3
10. Code composer studio – an integrated development environment. http://www.ti.com/tool/ccstudio, ver. 3.3