

A comparison of SPSA method and compact genetic algorithm for the optimization of induction motor position control

F. Cupertino, E. Mininno, D. Naso, L. Salvatore

POLITECNICO DI BARI

Via Re David 200 - 70125

Bari, Italy

Tel.: +39 / 080 – 5963769.

Fax: +39 / 080 – 5963410.

E-Mail: {cupertino,naso,mininno,salvatore}@deemail.poliba.it

URL: <http://dee.poliba.it/cemd>

Keywords

Asynchronous motor, Adjustable speed drive, Highly dynamic drive, Variable speed drive, Vector control.

Abstract

This paper describes the implementation of self-optimizing embedded control schemes for induction motor drives. The online design problem is formulated as a search problem and solved with stochastic optimization algorithms. The objective function takes into account the tracking error, and is directly measured on the hardware bench. In particular, we compare two efficient optimization algorithms, a Simultaneous Perturbation Stochastic Approximation method, and a Compact Genetic Algorithm. Both search strategies have very small computational requirements, and therefore can be directly implemented on the same processor running the control algorithm.

Introduction

Embedded control systems are becoming increasingly widespread in industrial automation. In these systems, the actuators are equipped with relatively-low-cost microcontrollers that can also perform self-tuning and adaptation functions. The authors of this paper are engaged with the development of computationally efficient and reliable self-tuning strategies for such embedded controllers. In the case of position control of induction motors (IMs), the problem is made particularly challenging by the typical cascaded structure of the control loops (see Fig.4), which are often tuned consecutively. Clearly, this approach neglects the possible interactions between the cascaded loops, and may lead, in principle, to suboptimal solutions. The idea investigated in this paper is to optimize the parameters of position and speed controllers of the IM simultaneously and online, using efficient and “computationally-light” optimization algorithms. In particular, two classes of effective algorithms are considered in order to investigate and compare their performances: the Simultaneous Perturbation Stochastic Approximation (SPSA) method and the Compact Genetic Algorithm (CGA).

SPSA is based on a highly efficient approximation of the gradient based on loss function measurements. In particular, on each iteration the SPSA only needs two loss measurements to estimate the gradient, regardless of the dimensionality of the problem. Moreover, the SPSA is grounded on a solid mathematical framework that permits to assess its stochastic properties also for optimization problems affected by noise or uncertainties. Due to these striking advantages, SPSA has been recently used as optimization engine for many adaptive control problems (see e.g., [1-3], or the extensive survey in [4]).

CGAs [5, 6] are evolutionary algorithms that mimic the behavior of conventional Genetic Algorithms (GAs) by evolving a probability vector (PV) that describes the hypothetical distribution of a population of solutions in the search space. A CGA iteratively processes the PV with updating mechanisms that

mimic the typical selection and recombination operations performed in a standard GA until a stopping criterion is met. [5] showed that the CGA is almost equivalent to a standard GA with binary tournament selection and uniform crossover on a number of test problems, and also suggested some mechanisms to alter the selection pressure in the CGA. The main strength of the CGA is the significant reduction of memory requirements, as it needs to store only the PV instead of an entire population of solutions.

With respect to the rich literature about online optimization of control systems, the main contribution of our work is the application of particularly “light” and effective algorithms, and their effective implementation on the same microcontroller used for the digital feedback control, thus overcoming the typical need of an external computer to carry out the controller optimization task. Moreover, since our research aims at finding an adequate compromise between noise rejection, speed of convergence and quality of the final solution, we evaluate and discuss the relative performance of the two families of algorithms on an extensive experimental campaign on a IM bench.

SPSA Algorithm

This section provides a short overview of the basic concepts related to SPSA methods. As technical details are thoroughly described in related literature [4], here we only overview the essential concepts. Consider the problem of finding the minimum of a differentiable loss function $L_{no}(\mathcal{G}): R^p \rightarrow R$ (The subscript *no* here is used to indicate that the loss measurements are affected by noise, whose distribution must satisfy some important conditions [4]). There is a large variety of stochastic algorithms that could be used to find the value of \mathcal{G} (say \mathcal{G}^*) that minimizes $L_{no}(\mathcal{G})$ [12]. The SPSA method computes the estimated value $\hat{\mathcal{G}}$ at the $(k+1)$ th iteration as

$$\hat{\mathcal{G}}_{k+1} = \hat{\mathcal{G}}_k - a_k \hat{g}_k(\hat{\mathcal{G}}_k) \quad (1)$$

where $\hat{g}_k(\cdot)$ is the estimated gradient at the k th iteration, and a_k is a gain scheduled to decrease over iterations with the law $a_k = a/(k+A)^\alpha$, where a , A , and α are positive configuration coefficients. SPSA estimates $\hat{g}_k(\cdot)$ using the following “simultaneous perturbation” method. Let $\Delta_k = [\Delta_{k1} \ \Delta_{k2} \ \dots \ \Delta_{kp}] \in R^p$ be a vector of p mutually independent zero-mean random variables (satisfying the conditions described in [4]), and let the sequence of vectors $\{\Delta_k\}$ be a mutually independent sequence with Δ_k independent of $\hat{\mathcal{G}}_j$, $j = 0, 1, \dots, k$. The basic SPSA method computes two new points in the solution space, and evaluates the corresponding loss as follows

$$y_k^+ = L_{no}(\hat{\mathcal{G}}_k + c_k \Delta_k) \quad (2)$$

$$y_k^- = L_{no}(\hat{\mathcal{G}}_k - c_k \Delta_k) \quad (3)$$

where c_k is a gain sequence $c_k = c/(k+1)^\gamma$, and c and γ are nonnegative configuration coefficients. Then, the estimation of the gradient at the k th iteration is computed as $\hat{g}_k(\hat{\mathcal{G}}_k) = \frac{y_k^+ - y_k^-}{2c_k} [\Delta_{k1}^{-1} \ \Delta_{k2}^{-1} \ \dots \ \Delta_{kp}^{-1}]^T$. It can be noted that all the elements of the vector $\hat{\mathcal{G}}$ are perturbed simultaneously, and that only two measures of the loss are needed to estimate the gradient independently of the size of $\hat{\mathcal{G}}$. Moreover, as the sequence $\{\Delta_k\}$ is usually obtained with a Bernoulli ± 1 distribution with equal probability for each value, the perturbations have the same amplitude for all the components of $\hat{\mathcal{G}}$. It has been proven [4] that under certain conditions the bias in $\hat{g}_k(\cdot)$ as an estimate of $g(\cdot)$ tends to zero as $k \rightarrow \infty$, and $\hat{\mathcal{G}}_k$ converges “almost surely” to \mathcal{G}^* . Literature also provides effective and theoretically valid values for most configuration coefficients [4, 7]

Compact Genetic Algorithm

This paper focuses on a CGA selected from recent literature that adopts a non-persistent form of elitism. In order to introduce the non-persistent elitist version of the CGA, in this section we provide a quick overview of three different CGAs, referring interested readers to [8] for further details. The pseudocodes of the three CGAs are summarized in Figg. 1-3. The basic CGA (Fig.1) mimics the order-one behavior of a standard binary-coded GA using a PV as a statistical descriptor of the population. Each element of the PV is a real value between 0 and 1. One gene of the PV represents the relative frequency of “1” and “0” in the i -th gene of the hypothetical population. During the initialization of the algorithm, each element of the PV is set equal to 0.5, so as to represent a uniform randomly distributed population. At each iteration of the algorithm, as shown in Fig.1, two new individuals are generated on the basis of the current PV, they compete with each other in a binary tournament, and then the PV is updated with a mechanism rewarding the winner of the tournament. For example, if i -th gene of the winner is “0” (“1”) and the i -th gene of the loser is “1” (“0”) then the i -th gene of PV is decreased (increased) of a factor $1/n$, where n is the size of the population. The termination condition of the algorithm is reached when all the probabilities converge to zero or one, and the PV becomes a binary vector describing the final solution.

The two variants proposed in [8] are called “persistent elitist CGA” (peCGA) and “non persistent elitist CGA” (neCGA). With respect to the standard CGA described in Fig.1, the main peculiarity of the peCGA is that the “winner” of the tournament is always kept in memory, and replaced only if a new individual with better fitness is discovered by the CGA. This can be practically obtained modifying the CGA as described in the pseudocode of Fig.2.

It is known that strong elitism may lead to premature convergence and to sub-optimal solutions. For this reason, the neCGA introduces a control parameter η limiting the “persistence” of the elitist solution. As in the peCGA, the loser is always replaced by a new individual generated from the PV. But, the winner of the competition between the elitist and the newly generated solution is passed to the next generation only if the so-called “length of inheritance” θ does not exceed the predefined threshold η . In other words, if a possible solution wins η consecutive competitions, it is replaced by a randomly generated one. This behavior can be obtained with the pseudocode in Fig. 3. This strategy can be very useful not only to maintain the necessary genetic diversity in the simulated population, but also to overcome the effects of the noise on fitness function measurements.

Parameters n : population size,

Step 1. Initialize iteration index and probability vector

$k=1$, for $j=1$ to m do $x_j^{(1)} := 0.5$

Step 2. Generate two chromosomes from the PV

$a := \text{generate}(x^{(k)}); \quad b := \text{generate}(x^{(k)});$

Step 3. Competition

$winner, loser = \text{compete}(a, b)$

Step 4. Update (mean and variance of) PV

for $j=1$ to m do

if $winner[j] \neq loser[j]$

if $winner[j] == 1$ $x_j^{(k+1)} = x_j^{(k)} + 1/n;$

else $x_j^{(k+1)} = x_j^{(k)} - 1/n;$

endif

endif

Step 5. Update iteration index and go to step 2 if the stopping criterion is not satisfied.

Fig. 1. Generic cGA pseudocode

Parameters E_{chrom} : elite chromosome

Step 2*. Generate one chromosome from the PV

if the first generation

$E_{chrom} := \text{generate}(x^{(1)});$

endif

$a := \text{generate}(PV^{(k)});$

Step 3. Competition

$winner, loser = \text{compete}(E_{chrom}, a)$

$E_{chrom} := winner$

Fig. 2. persistent elitist CGA (peCGA)

Parameters E_{chrom} : elite chromosome

θ : the present length of inheritance,

η : the allowable length of inheritance

Step 2. Generate one chromosome from the PV

if the first generation

$\theta := 0; E_{chrom} := \text{generate}(x^{(1)});$

endif

$a := \text{generate}(PV^{(k)});$

Step 3. Competition

$winner, loser = \text{compete}(E_{chrom}, a)$

if $\theta < \eta$

$E_{chrom} := winner$

$\theta++;$

else

$E_{chrom} := \text{generate}(x^{(k)});$

$\theta = 0;$

endif

Fig. 3. non-persistent elitist CGA (neCGA)

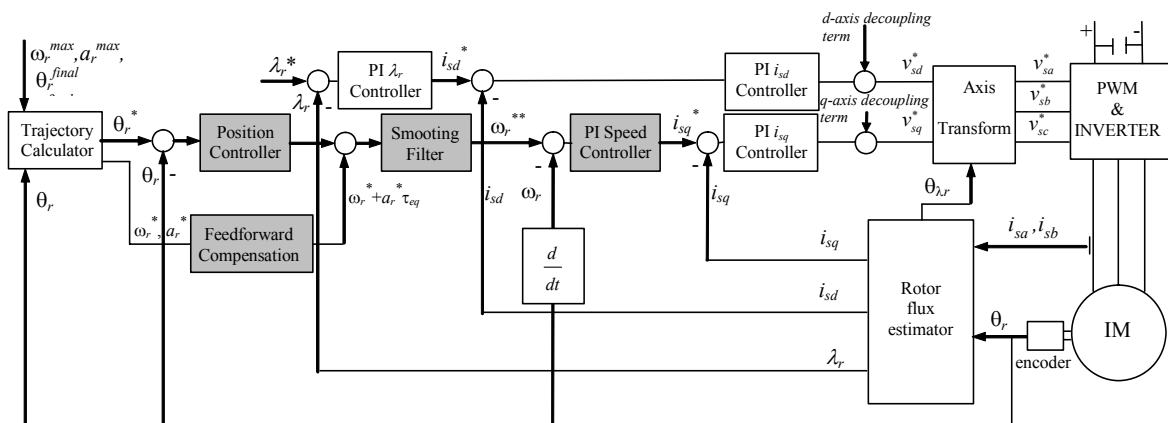


Fig.4. Induction motor drive block diagram

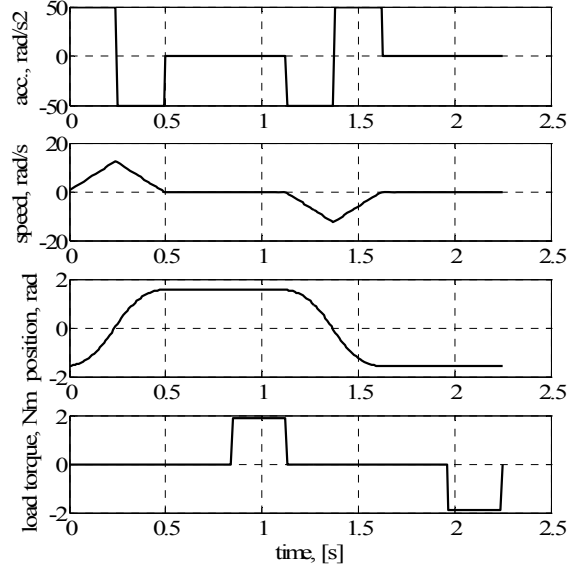


Fig.5. Desired trajectories and load torque applied during the test (two consecutive cost function evaluations are shown).

The loss function

The design of the experiment for loss measurement plays a fundamental role in the success of the online design. In this paper, the position reference signal corresponds to the minimum time trajectory for a rotation of π radians (see Fig.5). After 0.7 s from every change of the reference signal, a step change of load torque (from 0 to 70% of motor rated torque) is applied, in order to evaluate also the overall disturbance rejection. At the end of the loss evaluation experiment (after 1.125 s) the load torque is suddenly removed (see Fig.5).

The profiles of motor position, speed, and control action of the PI speed controller (the current $i_{sq}^*(t)$) are acquired during the experiment, and used to compute the loss, which essentially takes into account three criteria, i.e. the tracking performance, the disturbance rejection, and the smoothness of the control action. Namely, we define the following loss function (to be minimized):

$$L(\mathcal{G}) = \int_{T_{EXP}} \left(\alpha_1 f_1(\theta) + \alpha_2 f_2(\omega(t)) + \alpha_3 f_3(i_{sq}^*(t)) \right) dt \quad (4)$$

where T_{EXP} is the duration of the loss evaluation experiment, α_j represent positive weights, and f_j are three performance indices defined as follows:

$$f_1(\theta(t)) = \begin{cases} |\theta^* - \theta| & \text{if } \frac{|i_{sq}^*(t)|}{|i_{sq_{max}}^*|} < 0.99 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

$$f_2(\omega(t)) = \begin{cases} |\omega^* - \omega| & \text{if } \frac{|i_{sq}^*(t)|}{|i_{sq_{max}}^*|} < 0.99 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$f_3(i_{sq}^*(t)) = \begin{cases} |i_{sq}^{*F}(t) - i_{sq}^*(t)| & \text{if } \frac{|i_{sq}^*(t)|}{|i_{sq_{max}}^*|} < 0.99 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

The first two functions f_1 and f_2 take into account the tracking performance and disturbance rejection.

More specifically, the position θ and speed ω tracking errors are considered only in the time intervals in which the current feed $i_{sq}^*(t)$ is lower than a predefined threshold isq_{max} (the tracking error due to controller saturation cannot be further reduced). The third function f_3 compares the control action $i_{sq}^{*F}(t)$ filtered by a first-order linear filter with time constant $\tau=0.02$ s, with the unfiltered actual action $i_{sq}^*(t)$ itself. As smoother control actions give lower values of the integral of f_3 , this index is intended to penalize controllers with an excessively oscillatory control action, which may cause stresses for the IM producing vibrations, acoustic noise, and extra losses. The hardware scheme computes each index contributing to the total loss online, i.e. updating its value at each time sample of the experiment. The online value of each loss term is constantly monitored, and whenever it exceeds a predefined threshold the current experiment is immediately stopped. This allows the system to detect unstable (or highly unsatisfactory) solutions well before the involved signals reach potentially dangerous values. If a monitored index exceeds the prescribed threshold, the value of the loss is multiplied by a penalty factor and assigned to the individual, and the algorithm proceeds with another experiment. In this way, the stochastic search is never interrupted until the terminating condition occurs. On average, less than 1% of the experiments are prematurely interrupted due to bad performance, while the remaining iterations always generate stable solutions.

The weights α_j used in loss aggregation permit to emphasize or reduce the contribution of each single performance index in the final value of the loss. In this paper, the α_j are set heuristically, i.e. performing preliminary experiments with changed weights until the desired trade-off between indices is achieved.

Finally, the vector of parameters optimized by SPSA is defined as:

$$\mathcal{G} = [k_{pw} \quad k_{iw} \quad k_{pos} \quad \tau_{sm} \quad \tau_{eq}]^T \quad (8)$$

in which k_{pw} and k_{iw} are the proportional and integral gain of the anti-windup discrete-time PI speed controller (see Fig.1), k_{pos} is the gain of the proportional position controller, τ_{sm} is the time constant of the first order smoothing filter and τ_{eq} is the equivalent time constant of the position control loop (generally estimated offline with system identification procedures [9-11]). The stopping criterion is the maximum number of loss function calls chosen equal to 200 (i.e. the whole experiment lasts 225 seconds, although the entire algorithm converges in about one third of this interval).

The main advantage of the proposed embedded optimization scheme is the reliability of final results. While all the model-based techniques expressly rely on the accuracy of the model (generally used in simulations), in the online tuning case the effects of the actual and unknown high-order phenomena and nonlinearities are fully accounted in the loss measurement, and the final controller (the one generating the smallest loss) is ready-for-use with known performance. This permits to obtain automatic design tools that do not require skilled expertise for system modelling, or trial-and-error controller optimization.

Summary of experimental results

The overall scheme of the control system is illustrated in Fig.4. The whole control scheme is implemented in discrete-time on the dSPACE 1104 real-time control board, equipped with a 250 MHz Motorola PPC working as microcontroller. The microcontroller runs the control scheme and a stability supervision algorithm that interrupts the experimentation of badly performing solutions. The various modules have different sampling times. In particular, controller and stability supervisor run with a 200 μ s sampling time, while the optimization algorithms have a 1.125 s sampling time.

The IM nameplate parameters are as follows: voltage 220 V, current 3.1 A, power 750 W, speed 2860 rpm. The IM is loaded using a torque controlled brushless generator, mounted on the same shaft (see Fig.6).

The execution time of a complete iteration of the control scheme is about 100 μs , while one iteration of the SPSA requires about 180 μs , and one iteration of the neCGA requires about 600 μs in the considered case of five parameters to be optimized each represented with 16 bits. Although SPSA is less demanding, both algorithms could be implemented with any microcontroller capable to implement vector control. For both SPSA and neCGA, one cycle of the optimization algorithm is executed once every 5625 iterations of controller difference equation, and the increase of computer cost due to the introduction of the optimization algorithms is negligible, as it amounts to $(100 \times 180) / (100 \times 5625) = 0.032\%$ of the cycle time for the SPSA and 0.11% for the neCGA (see fig. 7).

For the SPSA method there are several configuration parameters that must be selected appropriately, which include c and γ , used to define the perturbed points for gradient estimation (2-4), and a , A , and α used to find the new solutions in (1). For this task, we have performed an extensive preliminary configuration study using a simulation model of the IM, also basing the investigation on the general suggestions provided by literature [2, 4, 7]. For our specific problem, the SPSA algorithms can find a reasonable solution in about 200 iterations, with $A=20$ (10% of the expected iterations to find optimum), and γ and α both equal to 0.3. These values make c_k and a_k reach small and almost constant values after about 150 iterations, letting the SPSA spend the final 50 iterations in local refinements the of controller. For the values of a and c , a grid of possible value was investigated, and they were set as follows $a=0.0183$ and $c=0.03$.

For the neCGA only two parameters have to be chosen: the population size n and the allowable length of inheritance η , under the condition $\eta < n$ [6, 8]. The reduced number of parameters greatly simplifies the initial commissioning of the neCGA. After a preliminary configuration study we chose $n=25$ and $\eta=12$. In the tests we performed, the choice $\eta \cong n/2$ always gave satisfactory results. After proper tuning, the performances of both algorithms were comparable. They permit to obtain a very effective closed loop scheme in less than three minutes, in a fairly repeatable and noise-tolerant way, and therefore can be considered fully compliant with the requirement of most embedded control devices (see fig.8). In order to investigate how much a proper tuning affects the performances of both algorithms, several test have been repeated using different configuration parameters and the average results reported in tables I-III. In the case of SPSA for the values of a and c , and in the case of neCGA for the value of n , a grid of possible values was investigated around the best known configuration parameters. In particular, each configuration was tested 100 times (considering different starting points in the search space in the case of SPSA) and calculating the average final loss function value (table I) and the percentage of satisfactory runs (i.e., with the final loss function below a predetermined threshold of satisfaction – see table II).

The performances of both algorithms during these tests were comparable. The neCGA reached an average loss function lower than SPSA and its performances were less influenced by configuration parameters (see table II). Even if the cost function improvement is not much appreciable in terms of performances of the IM, the results suggest that neCGA can be profitably used in our online problem. On average, the neCGA resulted a little faster than SPSA, as demonstrated by the average number of iterations needed to reach a satisfactory loss function value reported in table III. Fig. 9 shows the loss function and one of the five optimized parameters during typical runs of both algorithms. The outputs of SPSA method exhibits a “path-following” behaviour typical of gradient based algorithms while neCGA is more exploratory (it emulates a population-based search) and its outputs have less regular changes.

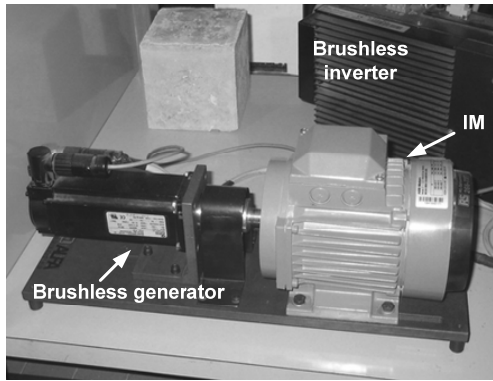


Fig. 6 – Equipment used during the experiments.

Table II: percentage of satisfactory runs

SPSA			
	c=0.03/5	c=0.03	c=0.03*5
a=0.0183/5	92 %	64 %	54 %
a=0.0183	86 %	90 %	84 %
a=0.0183*5	44 %	90 %	78 %
neCGA			
	n=25/5	n=25	n=25*5
$\eta=n/2$	90 %	92 %	85 %

TABLE I – average final loss function value

SPSA			
	c=0.03/5	c=0.03	c=0.03*5
a=0.0183/5	0.91	0.99	0.99
a=0.0183	0.83	0.81	0.88
a=0.0183*5	0.97	0.84	0.83
neCGA			
	n=25/5	n=25	n=25*5
$\eta=n/2$	0.85	0.70	0.72

TABLE III – average iterations needed to reach a satisfactory loss function level

SPSA			
	c=0.03/5	c=0.03	c=0.03*5
a=0.0183/5	92	82	93
a=0.0183	46	47	64
a=0.0183*5	46	23	25
neCGA			
	n=25/5	n=25	n=25*5
$\eta=n/2$	36	25	28

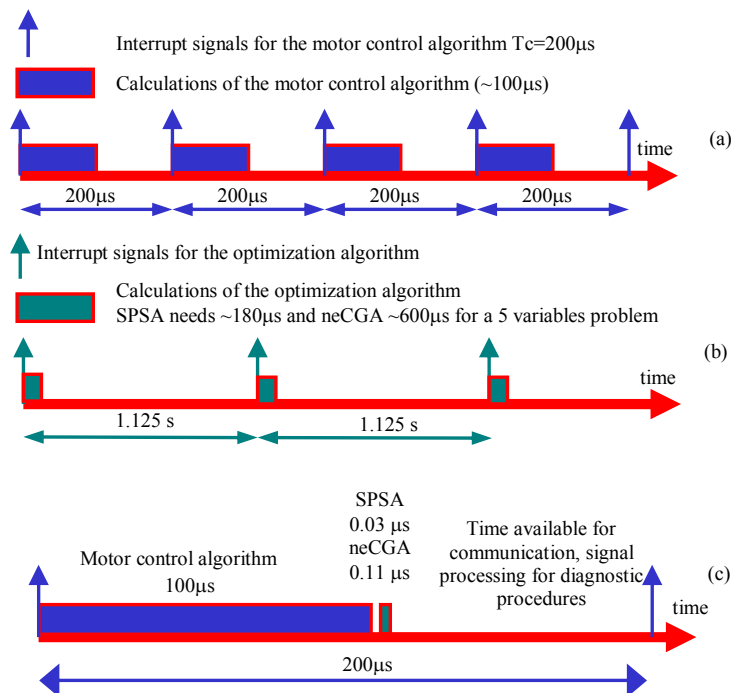


Fig. 7 – Distribution over time of the microcontroller calculations to perform (a) vector control and (b) optimization of the cascaded controllers. Figure (c) shows the average increase of computer cost due to the introduction of the optimization algorithms. For ease of representation the three figures do not use the same time scale.

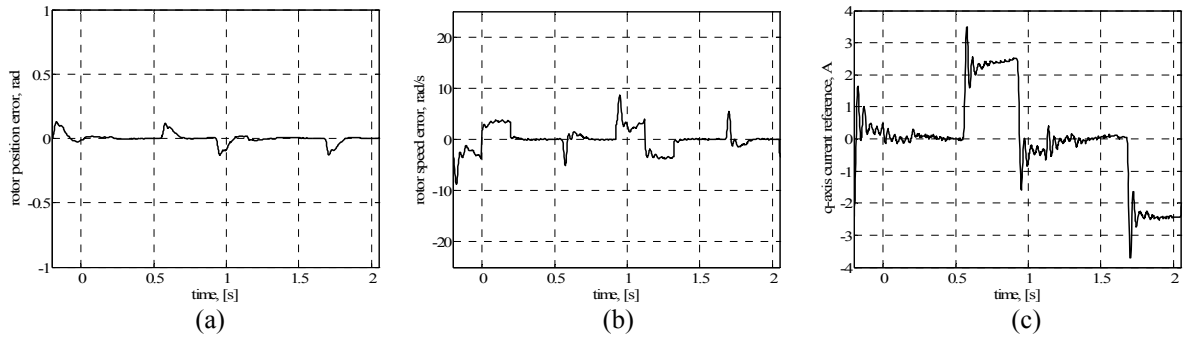


Fig. 8. Position tracking error (a), speed error (b), and current reference (c) under external load torque disturbance (the shown figures are obtained with SPSA algorithm, those obtained with neCGA are similar and omitted for brevity)

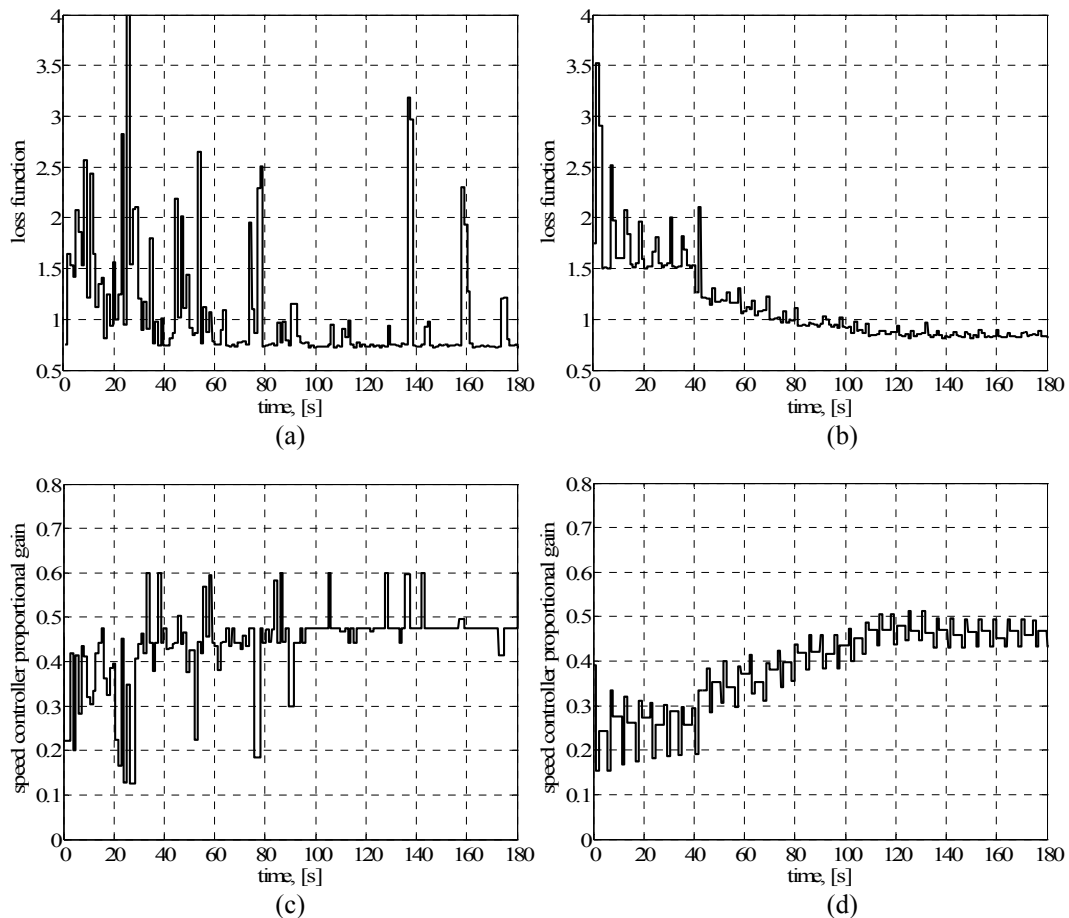


Fig. 9 –Loss function (a) and (b) and speed controller: proportional gain (c) and (d) during a typical run of neCGA (a) and (c) and SPSA algorithm (b) and (d) (the other optimized parameters are similar and omitted for brevity).

Conclusion

This paper described the implementation of self-optimizing embedded control schemes for position controlled induction motor drives. In particular, we have extensively compared two efficient optimization algorithms, a SPSA method, and an elitist CGA. Both search strategies have very small computational requirements and can be directly implemented on the same processor running the control algorithm. Both considered algorithms permit to obtain a very effective closed loop scheme in less than three minutes, in a fairly repeatable and noise-tolerant way, and therefore can be considered fully compliant with the requirement of most embedded control devices. The experiments demonstrated that the SPSA is about three times less demanding in the considered optimization

problem (five parameters to be optimized, each represented with 16 bits). On average the neCGA permitted to reach better solutions to the search problem using a reduced number of iterations and its performances resulted less sensitive to the configuration parameters. Concluding the comparison, it could be stated that the SPSA may be preferable for the reduced stress caused to the hardware during the search process. On the other hand, it is also necessary to remark that SPSA is sensitive to the choice of the initial individual. To obtain the same repeatability of the neCGA, it is necessary to retune the SPSA setting to make the SPSA more exploratory. In such a case, the behaviors of the two algorithms become almost indistinguishable. This research has many open aspects currently under investigations, including the evaluation of more complex parameterized controller schemes (e.g., NNs).

References

- [1] Alessandri A., Parisini T.: Nonlinear Modeling of Complex Large-Scale Plants Using Neural Networks and Stochastic Approximation, *IEEE Transactions on Systems, Man and Cybernetics*, vol.27, pp.750-757, 1997.
- [2] Ji X.D., Familoni B.O.: A Diagonal Recurrent Neural Network-Based Hybrid Direct Adaptive SPSA Control System, *IEEE Transactions on Automatic Control*, vol.44, pp.1469-1473, 1999.
- [3] J.C. Spall, J.A.Cristion: A Neural Network Controller for Systems with Unmodeled Dynamics with Applications to Wastewater Treatment, *IEEE Transactions on Systems, Man. And Cybernetics- Part B: Cybernetics* 27 no.3, pp. 369-375, 1997.
- [4] J. C. Spall, Introduction to stochastic search and optimization: estimation, simulation, and control, John Wiley and Sons, Hoboken, NJ, 2003.
- [5] G. Harik, F.G. Lobo, and D. E. Goldberg, The compact genetic algorithm, *Trans. Evolutionary Computation*, vol.3, pp. 287-297, Nov. 1999.
- [6] F. Cupertino, E. Mininno, D. Naso: Real-valued compact genetic algorithms for embedded microcontroller optimization, accepted for publication on *IEEE Trans. on Evolutionary Computation*.
- [7] F. Cupertino, E. Mininno, D. Naso, A comparative analysis of SPSA algorithms for induction motor adaptive control, *Proc of IEMDC 07, IEEE International Electrical Machines and Drives Conference*, Antalya, Turkey, 3-5 May 2007.
- [8] C. W. Ahn and R. S. Ramakrishna, Elitism based Compact Genetic Algorithms, *IEEE Trans. Evolutionary Computation*, vol.7, No.4, August 2003.
- [9] F. Cupertino, E. Mininno, D. Naso, B. Turchiano, L. Salvatore, On-line genetic design of anti-windup unstructured controllers for electric drives with variable load, *IEEE Transactions on Evolutionary Computation*, Vol.8., n.4, pp. 347-364, 2004.
- [10] H. Grob, J. Hamann, G. Wiegartner, *Electrical feed drives in automation*, Siemens, 2001.
- [11] R. Krishnan, *Electric Motor Drives, Modeling, Analysis and Control*, Prentice Hall, 2001.
- [12] K. Passino, "Biomimicry for Optimization, Control, and Automation", Springer, US, 2005.