ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 205-210

A Comparison of Neural Networks, Linear Controllers, Genetic Algorithms and Simulated Annealing for Real Time Control

M. Chiaberge[1], J.J. Merelo[2], L.M. Reyneri[1], A. Prieto[2], L. Zocca[1]

[1] Dipartimento di Elettronica, Politecnico di Torino
C.so Duca degli Abruzzi, 24 - 10129 TORINO - ITALY

[2] Departamento de Electrónica y Tecnología de Computadores
Facultad de Ciencias, Universidad de Granada, 18071 GRANADA - SPAIN

Abstract

[1] This paper reports a performance comparison between traditional (linear PID controller) and cognitive methods (neural networks, genetic algorithms and simulated annealing) applied to the problem of real-time control. A one-axis magnetic bearing is considered as a case study and cognitive methods have been successfully applied to its control. Comparisons are made using a real test setup. It is shown that hybrid approaches provide the best performance among all the methods analysed.

1 Introduction

Modern VLSI silicon technology provides interesting solutions to the problem of high reliability, fault tolerant, low power control of several systems. In many cases very cheap devices can be designed ad-hoc and manufactured. These can be used within critical subsystems such as, for instance: space robots, manipulators for harsh environments, on-board control of micro-mechanical systems, very fast response or strongly non-linear and unstable systems, etc. A crucial point in the design of a critical control system is the selection of an appropriate algorithm. In this paper traditional solutions to the problem are compared with modern ones based on *cognitive principles*, in terms of performance, training time and hardware complexity. The following solutions have been considered:

- **Linear PID controller**, which is the reference method. This is a commonly used method, with a well-established theoretical background [1]. It is well adapted only to linear, preferably stable, systems. This is a vey simple method which requires a very small silicon area.
- **Neural Network controllers (NN)**, which are based on non-linear matrix-vector multiplications [2]. They are used here as generic function approximators to map the desired input-output characteristic of the controller. The interesting advantage of NNs is that they are *trained from examples*, therefore they do not need any algorithmic model of the problem. The drawback is that they require a reference controller (to train from), which limits the performance of the network. NN controllers have very regular silicon implementations.

[1]This work has been partially supported by the *Inter-University Cooperation "Integrated Neural System for Robotic Applications" between Italy and Spain*

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 205-210

- **Genetic Algorithms (GA)** are powerful optimization methods [3] which can be used within control systems in conjunction with other techniques (e.g. PID, NN, etc.), either to find the best controller parameters (for PID) or the best network topology (for NN) [4]. They are substantially "trial and error" methods based on the way biological evolution works. A drawback is that for each "generation" of the algorithm, several competely different solutions need to be verified and stored but just one will be the final one. This wastes most of the area of a silicon device (for storage of unused solutions).
- **Simulated Annealing (SA)** is an optimization method [5] similar to the physical process of heating up a solid until it melts, followed by cooling it down until it crystallizes into a perfect lattice. This method is in some respect similar to GA but it requires a smaller amount of memory.
- **Hybrid Approach (GA+SA)** is the combination of a Genetic Algorithm and Simulated Annealing, which keeps the advantages of both methods, namely: lower memory requirements of SA and better performance of GA.

None of these methods (except for PID) can be efficiently used alone, for control problems, while the combination of two or more of them can provide interesting advantages. The scope of this work is to establish the feasibility of using the above methods when controlling a real-world system. In the whole study we have always kept in mind the constraint of an analog silicon implementation, which are: low memory requirements, insensitivity to computation errors and small complexity of the algorithm primitives.

2 Description of the Case Study

We have considered for this comparison a *one-axis magnetic bearing* [6], which is a rather simple problem, yet non linear and intrinsically unstable. As shown in fig. 1.a, a metallic sphere S is lifted by means of an electromagnet E. The magnetic force F_E shall balance the force of gravity mg and the inertia $m\ddot{x}$:

$$F_E \approx K_E \frac{I_E^2}{(x+x_0)^2} \approx mg - m\ddot{x} \tag{1}$$

where K_E and x are the magnetic constant of the system and the sphere's displacement from the steady position. It is possible to linearize this relationship:

$$\frac{K_E I_E^2}{x_0^2} - mg \approx \frac{2K_E I_E^2}{x_0^3} x - m\ddot{x} \tag{2}$$

which shows an unstable equilibrium point when $I_E \approx \sqrt{\frac{mg}{K_E}} x_0$.

The system electromagnet-sphere is controlled as part of a closed loop, where a PID is commonly used to make it stable. The PID performs properly only in the linear zone around the desired position $x = 0$.

3 Comparison Among the Approaches

Several combinations of methods have been applied and compared, as described in next paragraphs. Among others: a PID with coefficients euristically tuned; a NN trained after a PD using BackPropagation; a GA used to optimize the PID coefficients; a NN trained using GA; a SA used for PID tuning; a hybrid algorithm (GA+SA) for PID tuning.

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 205-210

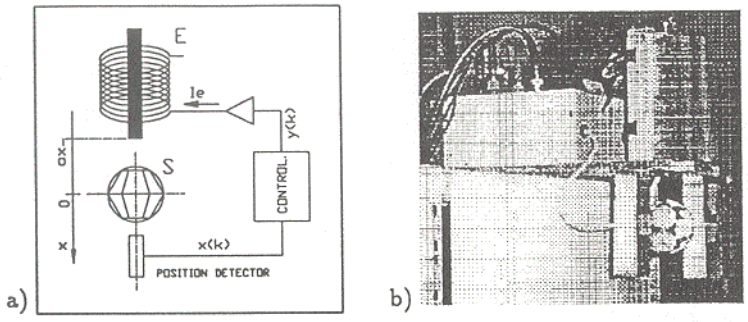

Figure 1: a) Block diagram of magnetic bearing; b) Photograph of the test setup

One major problem which arises when using such methods in practice is the need to deal with real hardware instead of simulators. This implies that several precautions are to be taken to avoid the undesired damage of the controlled system during the training phase which, as such, has a long initial phase during which the stimuli provided by the controller are far from being correct. An unstable system can therefore reach unwanted states, or even be damaged. In our case we have used a stand which limited the excursion of the ball to within ±5mm from the desired point. For all methods described further (except that in sect. 3.4.4) training has been performed on the real test setup shown in fig. 1.b. Every genome or network was trained (or tested) for a duration of 5 seconds.

The performance of all the methods analysed is measured in terms of the average position error. The *mean absolute error* (MAE) of the ball has been used:

$$\epsilon_M \triangleq \frac{1}{T}\int_T |x| dt, \tag{3}$$

which has also been used as the fitness function of Genetic Algorithms. Usually, the best results were obtained with the ball steadily oscillating few tenths of mm from the 0 level at about 5 Hz.

3.1 Neural Network Architecture

The neural network used is a three-layers feed-forward 3-5-1 neural network with 3 inputs: the time-sampled ball position x_k, the difference of position with respect to the previous measurement, $\Delta x_k = (x_k - x_{k-1})$ and the integral of all the measured positions $\sum_k x_k$, (these correspond to the 3 terms of a PID). The output neuron conveys its value to the electromagnet. All neurons are sigmoid units.

3.2 Neural network trained from a PD

A digital PD controller [1], whose transfer function is derived from an analog one, is used to train a 3-5-1 neural network similar to that described in sect. 3.1. The input configuration is chosen on the basis of the algebraic relation of the controller:

$$y_k = a \cdot y_{k-1} + b \cdot x_k + c \cdot \frac{\Delta x_k}{\Delta t} \tag{4}$$

where y_{k-1}, x_k and $\frac{\Delta x_k}{\Delta t}$ are the inputs. For training we used a BackPropagation algorithm with random initial weights and the PD output to simulate the feedback signal y_{k-1}. The main result is that training can be executed either on-line or off-line

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 205-210

using sinewaves as inputs to both the PD and the NN and that weights are indipendent of the sampling frequency f_t (provided that $f_t \geq 250Hz$). In order to compare the results with the other methods, an *iteration* is a block of 3000 epochs.

3.3 PID heuristically tuned

The controller used below is a linear PID with a transfer function given by:

$$y_k = K_p x_k + K_d \frac{\Delta x_k}{\Delta t} + K_i \sum_k x_k + K_0 \tag{5}$$

where K_p, K_d, K_i, K_0 are respectively the proportional, derivative, integral and constant terms of the controller. The PID has been used both alone (with coefficients tuned heuristically) and in conjunction with the other algorithms described further. A PID has a very compact silicon implementation.

3.4 Genetic algorithms

Two kinds of objects are evolved in this work by means of a GA: the weights of a neural network and the parameters of a PID.

In GA theory, it is usually necessary to put related parameters together in the genome. In our case, we keep close together in the *genome* the weights corresponding to the same hidden layer neuron. In this way, valid building blocks composed of hidden layer weights can combine to give good solutions to our problem. Each weight is given 2 bytes and scaled to $[-1, \ldots, 1]$ when decoding into the phenotype. The other parameters of the GA used are: 0.5 as crossover probability and 0.01 as mutation probability.

The fitness measure used is that defined in sect. 3 (i.e. $f = \epsilon_M$), to encourage solutions that keep the ball as close to $x = 0$ as possible. In some cases the fitness included also a measure of ball speed, to encourage those solutions which are steadier.

The population is composed of 50 chromosomes. The chosen method of selection has been *elitist* [3], the 20% of the population with worse fitness is eliminated, while the best 20% mate with the rest of the remaining population to substitute the former.

3.4.1 Genetically trained NN with random initial weights

The initial population was composed of neural networks with random weights.

Results were very bad; barely one network in each generation managed to make the ball rise from the rest position and make it fly for few fractions of a second. This shows that the combination of weights that make a good controller is very sparse. Since there was no valid building block in the first few generations, it was very difficult to obtain even one network capable of properly controlling the ball.

3.4.2 Genetically Trained NN with good initial population.

In this case, the initial population was formed by the weights found in sect. 3.2, with a mutation rate of 0.01, and from which the weights exactly equal to the first one were eliminated. Some good combinations of weights were found but they did not perform much better than those in sect. 3.2.

3.4.3 PID optimized by a genetic algorithm.

First, a set of good parameters for the PID was found heuristically (see sect. 3.3), in order to find the correct range of values. Within this range, the parameters were randomly chosen and the GA found the best solution, giving very good controller performance (see table 1). The final solution was found after few generations (see fig. 2).

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 205-210

Method	T_C (iterat.)	ϵ_∞ (mm)	Size of population	Memory size (words)	Number of trials	Training
PID (3.3)	-	0.75	1	4	-	on HW
NN (3.2)	12	0.69	1	38	12	on SW
GA (3.4.3)	8	0.23	50	200	400	on HW
SA (3.5)	10	0.68	4	16	80	on HW
GA+SA (3.6)	17	0.30	4	16	136	on HW

Table 1: Performance comparison of traditional and cognitive methods

3.4.4 NN genetically trained to emulate a PID.

GAs are able to find the best combination of weights for any NN problem; the only drawback is that it may take a long time. An initial population of random weights were generated. The NNs were compared against the PID described in sect. 3.3, by inputting a sinusoidal signal from 3 to 7 Hz to both the PID and the NN at the same time. In this case training was carried out off-line on a simulator.

We decided to accept an error of 1%, since the PID is already guaranteed to work well. This level of accuracy has not been achieved during training. In order to make it work properly, a bigger population and more generations should be used but obviously this increases memory and CPU time.

3.5 Simulated annealing for PID tuning.

With the SA, the solutions were chosen randomly and evaluated by a fitness function (the same than the GA algorithm) and, if the fitness of the new solution was less than the previous one, it was accepted with a probability of:

$$P(a) = e^{-\frac{\Delta f_k}{T}} \tag{6}$$

where T is the current "temperature" of the system and $\Delta f_k = (f_k - f_{k-1})$ is the difference between the energy (fitness) of the actual solution and the previous one. With an annealing schedule of T as $1/ln(k)$, the best solution is obtained after about 10 iterations and with a population of only 4 elements. This approach is slower than the one with GA but this algorithm does not need a large population of possible solutions and in the perspective of a hardware implementation, this is an important factor.

3.6 Hybrid approach: GA+SA

The best results can be obtained with a hybrid algorithm (GA+SA) [7, 8] wich combines standard GA operators to create new solutions and the SA methods for the other parts of the algorithm. A solution better than the one with SA is found in less than 20 iterations, with the same number of genomes as SA (see tab. 1). A hybrid algorithm with just mutations but no crossover has also been tried, as shown in 2.

4 Conclusion

Several cognitive methods have been analysed, tried and compared, for the control of an unstable system. Table 1 compares the different methods in terms of: Number T_C of iterations needed to reach an error close to the asymptotic error e_∞ (namely $\epsilon_M \leq 1.5e_\infty$); the asymptotic error e_∞, the size of the learning population (i.e. number of *genomes* or *weight sets* needed); size of memory needed (i.e. number of weights or

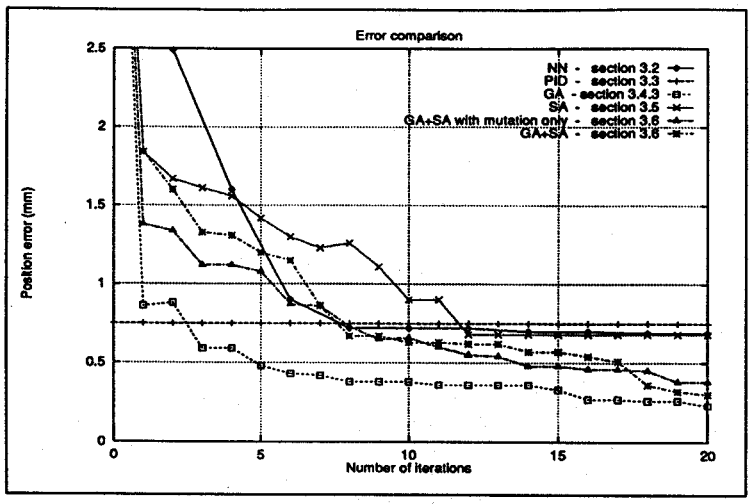

Figure 2: Evolution of the error during training for different cognitive algorithm

PID coefficients times the size of the population); number of "trials" (i.e. number of times the controlled system is stimulated with different genomes before the controller reaches $\epsilon_M \leq e_\infty$).

It is clear that the use of the hybrid algorithm GA+SA combines the main properties of the standard GA operators (efficient sampling of the solution space, stochastic hill-climbing, memory of the later stages of search) with those of SA (small population, efficient use of memory, easy hardware implementation).

References

[1] J.J. D'Azzo, C.H. Houpis, "Linear Control System Analysis and Design: Conventional and Modern", *McGraw-Hill*, 1988.

[2] P.J. Verbos, "An Overview of Neural Networks for Control", *IEEE Trans. on Control Systems*, Vol. 11, No. 1, January 1991, pp. 40-41.

[3] L. Davis, "Handbook of Genetic Algorithms", New York, *Van Nostrand Reinhold*, 1991.

[4] J.J. Merelo, M. Patón, A. Cañas, A. Prieto, F. Morán, "Optimization of a Competitive Learning Neural Network by Genetic Algorithms", in *New Trends in Neural Computations*, J. Mira, J. Cabestany and A. Prieto eds., Springer Verlag, 1993, pp. 185-192.

[5] E. Aarts, J. Korst, "Simulated Annealing and Boltzmann Machines", New York, *John Wiley & Sons*, 1989.

[6] H. Bleuler, E. Burdet, D. Diez, C. Gaähler, "Non Linear Neural Network Control for a Magnetic Bearing", in *Proc. of Ascona Workshop on Industrial Applications of Neural Networks*, Ascona (CH), September 16-20, 1991.

[7] Dan Adler, "Genetic Algorithms and Simulated Annealing: A Marriage Proposal", IEEE International Conference on Neural Networks 1993, San Francisco (CA), March 28- April 1, 1993.

[8] S.W. Mahfoud, D.E. Goldberg, "A Genetic algorithm for parallel simulated annealing", in *Parallel Problem Solving from Nature, 2*, R. Männer, B. Manderick eds., *Elsevier Science Publishers B.V.*, 1992.