# Using Genetic Algorithms to Reduce Jitter in Control Variables Transmitted over CAN

Fernanda Coutinho, José Fonseca, Jorge Barreiros, Ernesto Costa

**The wide use of CAN-based distributed systems in embedded control applications triggered the research on the problem of transmission network induced jitter in control variables. In this paper we introduce a variant of the classical Genetic Algorithm, which we call Progressive Genetic Algorithm, and show how it can be used to reduce jitter suffered by periodic messages by imposing them initial phasing and/or changing this parameter on-line. The approach can be applied when there is synchronisation between nodes (e.g. using time-triggered communication on CAN) or when a centralised dispatching scheme such as the one of FTT-CAN [1] is in use. The algorithm was tested with two well-known sets of messages, the PSA [2] and the SAE [3]. It is shown that it is possible to completely eliminate jitter if the adequate transmission rate is available and, if not, a satisfactory reduced jitter can be obtained.**

## 1. Introduction

The distributed control system approach is gaining wider use in some activities such as industrial control and automotive industry. In traditional systems, control tasks such as data acquisition, control algorithm execution and actuation are carried out independently by each one of the system's nodes. On a distributed system each of these tasks can be done on a different node. This means that control variables must be communicated across control units. This communication is usually supported by a real-time communication network which can be based on CAN [4], [5].

Control variables are typically periodic, with the actual period being specified by design requirements and control specifications. In a stand-alone system, it is usually possible to conform exactly to these requirements, because all the necessary resources are available to the node. However, in distributed systems, the network is a limited resource which needs to be shared by all the nodes. One consequence of this is that it may be impossible to ensure a constant time interval between successive instances of a message carrying a control variable. This fluctuations of the period are designated by jitter, and can be represented by:

$$j_{i,k} = ts_{i,k} - (k * T_i + \phi_i) \qquad (1)$$

Where $j_{i,k}$ is the jitter for the $k$-th instance of variable/message $i$ in a set of $M$ periodic messages (variables), $ts_{i,k}$ is the actual time of transmission for that instance, $T_i$ is the period of variable $i$ and $\phi_i$ is the time of the beginning of transmission of instance 0, also known as *initial phase*.

It is possible to measure the overall message jitter by integrating the individual jitter for each message over a time interval with the length of the system's macro-cycle. The interval of interest ($I$) for observing jitter is the macro-cycle, the least common multiple (LCM) of the individual messages' period. Overall system jitter (OSJ) can thus be calculated as:

$$OSJ = \sum_{i=1}^{M} \sum_{k=1}^{\frac{I}{T_i}} j_{i,k} \qquad (2)$$

Several problems related to transmission jitter have already been analysed and reported. In [6], Hong explores some problems caused by jitter-induced period variation. Specifically, if the instanta-

neous period becomes too high or too low it may fall out of the admissible data input rate for the receiver. Additional studies by Stothert et al. [7] show the degradation in the performance of feedback-loop controllers due to jitter in sampling and actuation variables. In [8], G. Juanole shows that transmission jitter in a CAN network affects the phase margin of a control loop.

In this paper, a novel technique for jitter reduction based on genetic algorithms is present. First, the concept of network jitter and associated problems are discussed. Next, genetic algorithms are introduced and briefly explained, following an example of a simple application to this problem. An enhanced genetic algorithm variant is then presented, and the paper is finished by exposing and discussing the results achieved.

## 2. Jitter Minimisation

It is obvious from (1) and (2) that OSJ is ultimately dependent of $M$, $T_i$ and $\phi_i$ ($I$ is directly dependent of $Ti$, and $ts_{i,k}$ can be determined if $M$, $I$, $T_i$ and $\phi_i$ are known).

$M$ and $Ti$ are application dependent values that cannot be easily changed without major impact on system's design and schedulability. The remaining free variable is $\phi_i$, which will be used as optimisation parameter. That is, optimisation is achieved by finding an adequate set of initial phasings $\phi_i$ for which OSJ is minimum.

This jitter minimisation technique is application-independent, and can be easily applied to any off-line scheduled fieldbus. It is well suited for time-triggered fieldbuses, such as FIP [9], TTP [10] or the upcoming time-triggered CAN (ISO TC22 / SC3 / WG1 / TF6). In fact, all that is necessary is to find an optimised phasing set for the initial scheduling table (Figure 1).

It is also possible to use this technique for on-line optimisation in a system that supports dynamic changes to the phasing set. One such system may be FTT-CAN [1]. This would allow gradual improvement of jitter over time, being a good supplement to other capabilities

such as on-line admission of new messages.



Periodic Message Set
{$m_i$, i=1...M}, {$\phi_i$=0, i=1...M}

Scheduler

Intermediate Scheduling Table   {$\phi_i^k$, i=1...M}

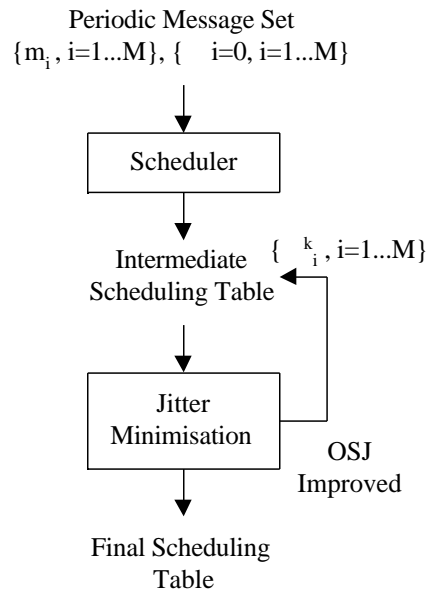Jitter Minimisation

OSJ Improved

Final Scheduling Table

Figure 1 – Applying jitter minimisation

## 3. Genetic Algorithms

Genetic algorithms (GA) are a class of stochastic optimisation mechanisms for multidimensional and multi-modal search spaces based on biological principles such as natural selection and genetics [11][12].

There is a great variety of methods which may fall under this classification, but the general algorithm may be described by the following procedure:

```
Procedure GA
t=0;
Initialise P(t)
Evaluate P(t)
While stoping_criterion_false do
    t=t+1
    P'(t)=select_from P(t-1)
    P''(t)= use_op_modification P'(t)
    Evaluate P''(t);
    P(t)=merge P''(t), P(t-1)
End_do
```

The GA operates by iteratively replacing a set of candidate solutions for the problem with a new, eventually better set. The algorithm stops when a predefined condition (such as a given number of iterations) is met. These sets are usually referred to as the *population*

and each candidate solution as an *individual.*

Each individual is normally composed by several smaller elements called *genes*, which can hold different values or *alleles.* The set of genes that each individual contains is sometimes designated by *genome.*

On start-up, the initial population is usually (but not always) generated randomly. A new set of individuals is generated by applying some genetic operators such as *crossover* and *mutation* to selected individuals from the previous population. The new population results from the combination of the new individuals with the old population. This combination can have different proportions from each set. In a *total replacement* GA the old population is completely discarded, while in a *steady-state* GA, only one or two individuals can change across consecutive populations.

Selection of individuals for generating new candidate solutions is usually based on the *fitness*, which is a measure of the quality of the solution one individual represents. The best solutions are those corresponding to individuals that have higher fitness.

GA's are successful at solving many difficult problems due to their capability of exploring and exploiting complex search spaces efficiently.

For increased efficiency, domain-specific knowledge is usually included in the algorithm. This is reflected in the selection of appropriate data structures and by "tuning" the genetic operators [13].

## 3.1 A Simple GA Approach for Jitter Minimisation

Jitter minimisation can be achieved by using a simple implementation of a standard genetic algorithm. We will refer to this implementation as the "simple GA" (sGA).

The genome is a simple vector that holds the values of $\phi_i$ for all the messages. Each allele is an integer within the range $[0, T_i[$ that represents the initial phase of message $i$.



Genotype

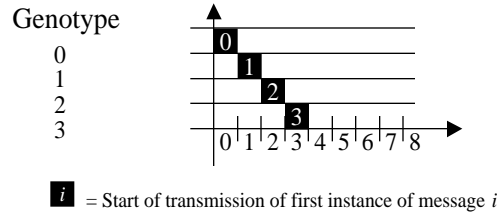$i$ = Start of transmission of first instance of message $i$

Figure 2 - Phases representation.

The best solutions are those that generate smaller OSJ, so the fitness of each individual is computed as $1/(OSJ+1)$. OSJ is computed by simulating network usage during a macro-cycle (this simulation is described in greater detail in a following section).

The genetic operators used are one-point crossover and mutation. Crossover mimics the sexual reproduction mechanism as seen in nature and works as follows: two individuals (the *progenitors*) are randomly chosen from the population with a weighted probability proportional to their fitness. Then, the genomes of both progenitors are split in two portions at a random point. Finally, two new individuals (the *descendants*) are generated concatenating each portion of the genome with the complementary portion coming from the other progenitor's genome.

Crossover will be applied with 100% probability. This means all descendants are generated with crossover.

Mutation is a simpler operation where a new genome is derived from the original by inserting a random change in the value of one of the genes. This operation is performed after crossover with low probability (3%).
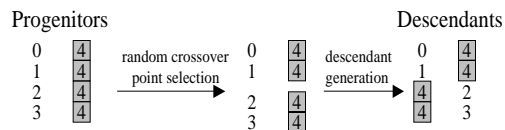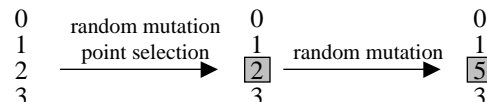


Figure 3-One-point crossover.



Figure 4 - Mutation operator.

The initial population is generated randomly. Then, at each iteration, two

descendants are generated using the crossover operator (and eventually mutation). Any descendant who is better (fitter) than the worst individual currently in the population, will replace it in the next iterations population. So, at most two individuals can enter or leave the population in each iteration.

The algorithm ends when the population is stable for a consecutive number of iterations (200 iterations). This happens when none of the descendants are better than the worst individual during this timespan.

### 3.2 OSJ Computation

In this problem, fitness evaluation is made by running an event-driven simulator that computes network usage for during a macro-cycle. The duration of this macro-cycle is equal to the LCM of the messages' periods.

This simulator holds the system's current state in two vectors: one holds the elapsed transmission time of messages and other holds the time for the next release or end of transmission of each message. This information allows the simulator to make variable time increments that are as large as possible. These variable time increments allow the simulator to scan the system's state along the macro-cycle efficiently.



Figure 5 - Simulation example

On start-up, the network is considered to be free of use. This may not be accurate, as some messages can "wrap-around" the macro-cycle. To eliminate this effect, simulation results are considered only after a predetermined transient time as elapsed.

The transient time is determined by simulation of network usage for a worst case of simultaneous release of messages. The duration of the transient state is equal to the time it takes for the message with lower priority to end transmission. From this point onward, the

system's state can be determined without knowledge of its previous state.

### 3.3 Progressive Genetic Algorithm (proGA)

Although the sGA is functional and offers satisfying results, it is rather inefficient because fitness evaluation time is dependent on the LCM of the periods of all messages. This can be quite significant for complex systems with a large number of messages.

This fact has lead to the development of a variant GA where the number of messages simultaneously analysed is reduced, with a correspondent decrease in the LCM.

This variant, the progressive GA (proGA), works by solving the problem for consecutively bigger subsets of messages of the original set until all the messages are considered.

The overall algorithm is described bellow:

- Solve the problem for the set of the two messages with higher probability. This is done using the sGA previously described.
- Add the next message with the highest priority to the previous set. Initialise a new population with the results for the previous set (+ a random phase for the new message). Solve the problem for this set of messages using the sGA, using the computed population as the initial one.
- Repeat this process until the set of messages includes all the messages in the original system.

This algorithm is more efficient and effective than the sGA, because part of the solution is computed using only a reduced number of tasks. All the results presented here were obtained with the proGA.

One additional advantage of the proGA is the fast availability of a partial solution for the messages with higher priority. This feature can make this procedure usable as part of an on-line scheduler and, since the most of the bandwidth is usually required for the messages with

lower period (the ones with higher priority under Rate Monotonic), can lead to partial solutions quite close to the actual optimum solution.

## 4. Experimental Results

For assessing the quality of the algorithm, several tests were conducted with two message sets usually known as the SAE (Society of Automotive Engineering) [3] and PSA (Peugeot Societé Automobile) [2], which define a typical set of control variables for automatically guided vehicles and automobiles, respectively. CAN networks with transmission rates of 125 Kbit/s and 250 Kbit/s were considered for determining the time values for the transmission of each message. The message set was scheduled with the Rate Monotonic Scheduling policy.

No experiments were conducted on the SAE benchmark at 125 Kbit/s because at this speed the set of messages isn't schedulable (in the worst case scenario of simultaneous release of messages).

Several experiments were conducted using varying resolutions for the initial phases' specification. These resolutions were set at 1 µs, 10µs and 100 µs.

Because of the large size (over 50 messages) of the SAE set, there is only a presentation of the optimal phasing for the messages of higher priority.

All times for initial phasing are presented in µs and execution time average in s. OSJ values obtained for best random phase sets were selected after the generation of 5000 random sets for each situation.

In the tables below it can be seen that the OSJ obtained with the genetic minimisation is, in the worst case which is the PSA at 125Kbps, one order of magnitude below the OSJ of random phase sets. When the bus is not so loaded as in that case the improvement in OSJ is much higher and it is possible to obtain a complete elimination of jitter as in the SAE set at 250Kbps, even with a coarse timer resolution of 100µs.

| Condition | OSJ |
|---|---|
| Simultaneous release of messages | 3679408 |
| Best random phase set (1 µs resolution) | 103613 |
| Best random phase set (10 µs res.) | 103524 |
| Best random phase set (100 µs res.) | 110900 |

**Table 1. Jitter values for not optimized systems (PSA at 125 Kbit/s).**

| Condition / Resol. | 1µs | 10µs | 100µs |
|---|---|---|---|
| Lowest OSJ | 12640 | 12640 | 22640 |
| Average OSJ | 13947 | 13364 | 14619 |
| Worst OSJ | 16256 | 15272 | 16304 |
| Exec. Time Average | 160 | 171 | 124 |
| Nº of Experiences | 20 | 20 | 20 |

**Table 2. Summary of optimization results for PSA benchmark at 125 Kbit/s.**

| Variable / Resol. | 1µs | 10µs | 100µs |
|---|---|---|---|
| 1-engine controller | 0 | 0 | 0 |
| 2-wheel angle sens. | 1200 | 3170 | 13200 |
| 4-AGB | 6778 | 12740 | 1800 |
| 7-device x | 1800 | 2720 | 11800 |
| 3-engine controller | 5956 | 16320 | 15900 |
| 5-device x | 15909 | 6060 | 4000 |
| 9-device y | 8221 | 11970 | 14300 |
| 6-device x | 18383 | 21900 | 5900 |
| 8-bodywork sensor | 14072 | 14260 | 28200 |
| 11-AGB | 3976 | 24280 | 8300 |
| 10-engine controller | 24047 | 4190 | 18200 |
| 12-device x | 12686 | 28640 | 22600 |

**Table 3. Best phasing set for PSA benchmark variables at 125 Kbit/s.**

| Condition | OSJ |
|---|---|
| Simultaneous release of messages | 1764704 |
| Best random phase set (1 µs res.) | 5962 |
| Best random phase set (10 µs res.) | 5544 |
| Best random phase sets (100 µs res.) | 5488 |

**Table 4. Jitter values for not optimized systems (PSA at 250Kbit/s).**

| Condition / Resol. | 1µs | 10µs | 100µs |
|---|---|---|---|
| Lowest OSJ | 0 | 0 | 0 |
| Average OSJ | 0 | 0 | 0 |
| Worst OSJ | 0 | 0 | 0 |
| Exec. Time Averag | 28 | 28 | 28 |
| Nº of Experiences | 20 | 20 | 20 |

**Table 5. Summary of optimization results for PSA benchmark at 250 Kbit/s.**

| Condition | OSJ |
|---|---|
| Simultaneous release of messages | 8435636 |
| Best random phase set (1 µs res.) | 156783 |
| Best random phase set (10 µs res.) | 163546 |
| Best random phase set (100 µs res.) | 143952 |

**Table 6. Jitter values for not optimized systems (SAE at 250 Kbit/s).**

| Condition/Resol. | 1$\mu$s | 10$\mu$s | 100$\mu$s |
|---|---|---|---|
| Lowest OSJ | 0 | 0 | 0 |
| Average OSJ | 68 | 178 | 545 |
| Worst OSJ | 230 | 1374 | 1964 |
| Exec. Time Average | 540 | 739 | 928 |
| N$^o$ of Experiences | 20 | 20 | 20 |

**Table 7. Summary of optimization results for SAE benchmark at 250 Kbit/s.**

| Variable / Resol. | 1$\mu$s | 10$\mu$s | 100$\mu$ |
|---|---|---|---|
| 7-Accelerator Position | 0 | 0 | 0 |
| 9-Brake Pressure, Line | 3975 | 4740 | 3500 |
| 8-Brake Press., Master Cyl | 1225 | 3910 | 4400 |
| 32-Clutch Press. Control | 4382 | 4190 | 4100 |
| 49-Proc. Motor Speed | 4745 | 2080 | 4700 |
| 42-Torque Command | 633 | 3390 | 3800 |
| 43-Torque Measured | 1477 | 4460 | 900 |
| 11-Tran. Clutch Line Press | 954 | 1820 | 1200 |
| 29-High Contactor Control | 6855 | 5420 | 7400 |
| 30-Low Contactor Control | 8400 | 3120 | 300 |
| 14-Hi&Lo Cont. Op/Close | 10358 | 8000 | 7100 |
| 25-12V Pwr Ack I/M Cont | 11772 | 17730 | 600 |
| ... | ... | ... | ... |

**Table 8. Best phasing set for a sample of SAE benchmark variables at 250 Kbit/s.**

Additional experiments revealed that the proGA is more efficient than the simple GA. The proGA is more time-effective and offers better results consistently [14]. The results also show that a timer resolution of 100$\mu$s seems to be enough to get good results out of this jitter reduction technique. In fact, the best results with that resolution were equal to results obtained with the highest resolutions in every test case.

Tables 2, 5 and 7 show that the speed of the proGA algorithm is insufficient for direct on-line full optimisation. However, as the computational overhead increases almost exponentially with the initial number of messages considered [15], the algorithm's iterative approach opens the possibility of implementing an on-line optimiser. The successive partial results of the messages with higher priority can be supplied to a communications dispatcher thus making the system evolve to a near-optimal message phasing. This may also prove to be effective, because the optimisation of the messages with higher priority just occupies a small percentage of the total execution time of the algorithm, and those are the messages with most impact on overall bandwidth occupation.

## 5. Conclusions

In this paper a technique based on genetic algorithms is proposed to reduce network induced jitter in periodic message transmission in fieldbuses. This technique requires that it is possible to impose the release instant of the different instances of the periodic messages. It can then be applied in CAN when a protocol such as FTT-CAN is used or with a time-triggered solution like the one under standardisation through ISO (TC22 / SC3 / WG1 / TF6).

The experimental results show that jitter can be strongly reduced or even completely eliminated if the fieldbus is not too loaded with the message transmission.

Although the technique imposes a relatively high computational overhead, at least considering the usual processors available in distributed embedded systems, it is possible to apply it incrementally using a variant of the algorithm in which successive partial solutions are obtained for the messages with higher priority. This seams to make feasible the development of an on-line optimiser (in software or hardware) that can operate with a communications scheduler or dispatcher.

## 6. Bibliography

[1] L. Almeida, J. Fonseca - "*A Flexible Time-Triggered Communication System Based on the Controller Area Network*", Proceedings of FeT'99 - Fieldbus Systems and their Applications Conference, Magdeburgo, Germany, September of 1999.

[2] K. Tindell, A. Burns, "*Guaranteeing Message Latencies on Control Area Network (CAN)*", Proceedings of ICC'94 (1st International CAN Conference), Mainz, Germany, 1994.

[3] N. Navet, Y. Song, "*Performance and Fault Tolerance of Real-Time Applications Distributed over CAN (Controller Area Network)*", CiA – CAN in Automation Research Award, 1997.

[4] Thomesse, J.P., "*A Review of the Fieldbuses*", Annual Reviews in Control, 22 pp. 35-45, 1998.

[5] L. Almeida, M.L. Chavez, J.A. Fonseca, J.-P. Thomesse, *"Real time communications in manufacturing"* Proceedings ISAS'99 The 5th. Int'l Conference on Information Systems Analysis and Synthesis, Orlando, USA, July/August 1999.

[6] S. Hong, "*Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control System*s", IEEE Transactions on Control Systems Technology, Vol. 3, Nº 2, June 1995.

[7] Stothert, et al "*Effect of Timing Jitter on Distributed Computer Control System Performance*", Proc. 15 IFAC Workshop DCCS'98 – Distributed Computer Control Systems, September 1998.

[8] G. Juanole, "*Modélisation et Évaluation du. Protocole MAC du Réseau CAN*", École d'été ETR'99 – Applications, Réseaux et Systèmes, ENSMA, Poitiers – Futuroscope, France, September 1999.

[9] P. Leterrier, "*The FIP Protocol*", WorldFip Europe, 2-4 Rue de Bône, 92160 Antony – France, 1992.

[10] H. Kopetz, G. Grünsteidl, "*TTP – A Protocol for Fault-Tolerant Real-Time Systems"*, IEEE Computer, 27(1), 1994.

[11] J. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Michigan, 1975.

[12] T. Back, D. Fogel, Z. Michalewicz (eds). "Handbook of Evolutionary Computation", Oxford University Press, New York, 1997.

[13] Z. Michalewicz, "*Genetic Algorithms + Data Structures = Evolution Programs*" (3rd, revised and extended edition), Springer-Verlag, Berlin,1999

[14] J. Barreiros, E. Costa, J. A. Fonseca, F. Coutinho, "*Jitter reduction in a real-time message transmission system using genetic algorithms*", Proceedings of the CEC 2000 – Conference of Evolutionary Computation, USA, July 2000

[15] F. Coutinho, J.A. Fonseca, J. Barreiros, E. Costa "*Jitter Minimization with Genetic Algorithms*", Proceedings WFCS'2000, 3rd IEEE International Workshop on Factory Communication Systems, Porto, Portugal, September 2000

## 7. Authors

Fernanda Coutinho
Instituto Superior de Engenharia de Coimbra, Quinta da Nora
3030 Coimbra, Portugal
Telef: +351 239 790200
Fax: +351 239 790270
Email: fernanda.coutinho@isec.pt

José Fonseca
Universidade de Aveiro / IEETA
Deptº Electrónica e Telecomunicações
3810 Aveiro, Portugal
Telef: +351 234 370984
Fax: +351 234 381128
Email: jaf@det.ua.pt

Jorge Barreiros[*]
Centro de Informática e Sistemas
Universidade de Coimbra, Polo II
3030 Coimbra, Portugal
Telef: +351 239 790273
Fax: +351 239 790270
Email: jmsousa@isec.pt
Homepage:http:\\www.usec.pt\~jmsousa

Ernesto Costa
Centro de Informática e Sistemas
Universidade de Coimbra, Polo II
3030 Coimbra, Portugal
Telef: +351 239 701419
Fax: +351 239 701419
Email: ernesto@dei.uc.pt
Homepage:http:\\www.dei.uc.pt\~ernesto

---

[*] Assistant Teacher at Instituto Superior de