7th Symposium on Computer Aided Control Systems Design, Gent, Belgium, April 1997

FLEXIBLE REAL–TIME ALGORITHMS ADAPTING THEMSELVES TO ALTERNATING LOAD SITUATIONS [1]

Johann Pfefferl * **Georg Färber** *

* *Laboratory for Process Control and Real Time Systems*
Prof. Dr.–Ing. G. Färber
Technische Universität München, Germany
{pfefferl,faerber}@lpr.e-technik.tu-muenchen.de

Abstract. In many applications, approximate results are often sufficient to achieve an acceptable behavior of a real–time system. Imprecise calculations could be necessary due to a transient overload situation caused by extraordinary events. To handle this exceptional work as well as the normal tasks, it is possible to reduce temporarily the computation time of the normal jobs by applying a more intelligent and flexible form of algorithms. In this paper, algorithm structures are presented, which are suitable to implement the mentioned methods. The basic idea is to offer multiple algorithm alternatives of different complexities for solving a problem. Then one can decide with the knowledge of the deadline and of the computation time which of the alternatives can be executed in time. These decisions will happen at runtime. To prove the applicability of the proposed methods on a real system, the theoretical considerations are devolved to a practical example consisting of a vehicle convoy. In this system, a normally realized control unit is substituted by an imprecisely working one. The effects of this action are described and analyzed with regard to the normal implementation. Therefore, a number of simulation results are presented. There exists also an experimental setup for verification in a real environment.

Keywords. real–time algorithms, load adaptation, imprecise computation techniques and resulting errors, control systems

1. INTRODUCTION

In the last few years, many advances in computer technology have accelerated the development and the application of computer–based automation. Most advanced automation systems utilize digital computers to support important monitoring and control functions. Examples include intelligent vehicle control, mechatronic applications and process control.

Such a system is only dependable, when the compound of digital control unit and controlled environment are robust and fault tolerant. These requirements imply, that the whole system shows only a slightly different behavior in the case, when a transient, small disturbance interferes the normal operation. Most system designs are realized in such a way, that the systems tolerate these faults. For example, these faults can be caused by sensor failures and the result could be a deviating trajec-

[1] The work presented in this paper is supported by the *Volkswagen–Stiftung*, Hannover, Germany, as part of the interdisciplinary research project "Integration of distributed mechatronic systems with special regard to real–time behavior".

tory. Although the corrupted trajectory doesn't follow the desired, exact trajectory, the system performance may nevertheless by adequate, when the error tends to zero or is within an acceptable bound after a specified amount of time.

The effects of corruption of the system due to controller or sensor failures are often accounted for by introducing additive and multiplicative noises to the machine model or control device. For instance, the publication (Gundes, 1992) analyzes the internal stability of multiple control systems in the presence of sensor and actuator failures. The general aspects of robust stability and integrity are discussed in many books on controller design (e.g. (Doyle *et al.*, 1992)).

This general robustness of most systems can be exploited to implement a new and more flexible type of control system. All the control algorithms have to fulfill real–time requirements with regard to timeliness. A timing fault is said to occur, when a real–time process delivers its result too late. To handle such situations correctly, a new approach, called *Imprecise Computation*, was proposed recently as a means to avoid these timing faults (Liu *et al.*, 1987*c*; Liu *et al.*, 1987*b*). This technique relies on making approximate results available at the deadline, that are of poorer, but acceptable quality, instead of aborting the system operation because of exceeding the time limit. This strategy relies on the fact, that a system does not leave the stable state by the occurrence of short transient corruptions.

This paper outlines an approach to avoid deadline violations in real–time systems by implementing real–time tasks in a somewhat different manner than usual. Some possibilities will be presented in section 2. After describing ways to implement real–time applications more flexible, the theoretical concept will be verified on a practical example of a vehicle convoy system in section 3.

2. LOAD ADAPTIVE REAL–TIME ALGORITHMS

2.1 *Application model*

Many real–time systems are used to control a machine, a mechatronic system or a specific process. For that reason, a multitude of the application tasks has to be activated with a strict periodic scheduling policy. Of course, many systems also require a mechanism to handle sporadic or aperiodic occurrences of external and internal events. Therefore, this technique is often used to implement specific system functions more efficiently. Other advantages are the increasing flexibility and the ability to realize more complex applications in an elegant way.

A disadvantage of event driven system components is the fact, that the occurrence of a single event or the combination of events is not predictable in advance without an exact process model (Gresser, 1993; Thielen, 1994). If such a model does not exist, but events should still be used as a powerful method, the described real–time algorithms, which adapt themselves to the actual load situation, are a good alternative to guarantee timing constraints.

2.2 *Classes and Properties*

A number of computations perform their operations in successive steps and phases (Dolev *et al.*, 1982). After each phase of computation the problem is solved with a better accuracy then at the previous step. If this behavior is true, the algorithm is said to be monotone with regard to result quality. Therefore, as more computation time is spent to problem solving, the more the result gets correct.

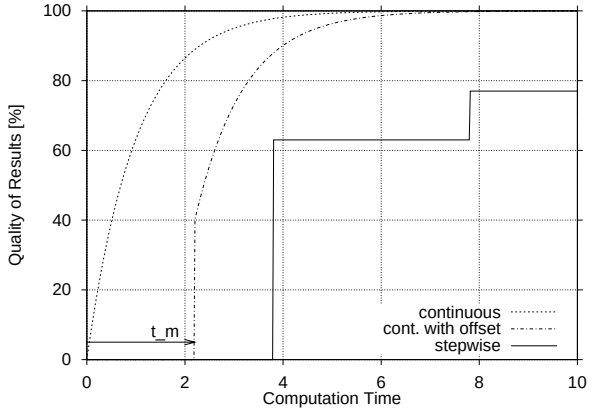

Fig. 1. The spent computation time and the type of real–time algorithm influence the quality of its result

This fact corresponds to the continuous quality function in Fig. 1. Often it is the case, that the precise or exact result is reached not by linear but exponential convergence. The marked duration t_m represents the minimum amount of time, which an computation scheme needs to product a first useful result. A third class of algorithms behaves more like the staircase function as shown in Fig. 1. Each step in the figure indicates the end of another block of computation or the finishing of a previous iteration loop. These steps are called *milestones* in the terminology as used in (Liu *et al.*, 1987*a*). The whole figure should also express, that different computation methods could vary in multiple orders of time requirements. For example, image processing needs much more time than the calculation of digital filters to obtain a reasonable quality.

2.3 *Computation variants*

2.3.1. *Multi-phase approach*

There exist a number of algorithms, which try to find a solution for a problem by solving the corresponding equations with an iterative or recursive procedure (Basu, 1980). Examples are calculations based on Taylor expansion (e.g. $\sin x$) or Newton Raphson methods to get the roots of nonlinear equation systems. Another possibility is, that the calculation is split in multiple separate blocks. The successive block uses the results of the previous one to improve the quality of the already obtained intermediate result.

These structures can be used to react on timing restrictions by truncating the execution at the transition from one block (iteration) to the next one. The central idea is to record meaningful partial results obtained at specific points in the execution of a computation. In the event that no more work (phases) can be done due to deadline appearance, the last recorded values are used as an approximation of the exact result.

In practice, the algorithm must be divided into a mandatory and one or more optional parts to maintain a quality, which is necessary to hold the system in a stable operating mode.

2.3.2. *Selection method*

Many problems can be solved by a number of different ways. The solutions may differ concerning the quality of their results. A better quality is often correlated with more computation time spent to solve the problem. The existence of different strategies can be used to realize a multi–version formulation of an algorithm.

```
if(RTSusableTime() >= RuntimeBlock1)
    Block1
elsif(RTSusableTime() >= RuntimeBlock2)
    Block2
elsif(RTSusableTime() >= RuntimeBlock3)
    Block3
...
else
    MinimalBlock
```

Fig. 2. Algorithm structure of an alternative computation scheme

Depending on the available computation time and the actual load situation on the computer node, a pertinent method is selected from the several alternatives available. Refer to Fig. 2 to get an overview, how these alternative algorithms can be merged together to establish a single unit. As you can see, a special system call `RTSusableTime()` is invoked to get the available amount of execution time for the calling task. With the knowledge of the runtime of the different alternatives, the task is capable to decide which calculation can be performed in time. The information of the available computation time has to be provided by an appropriate runtime system.

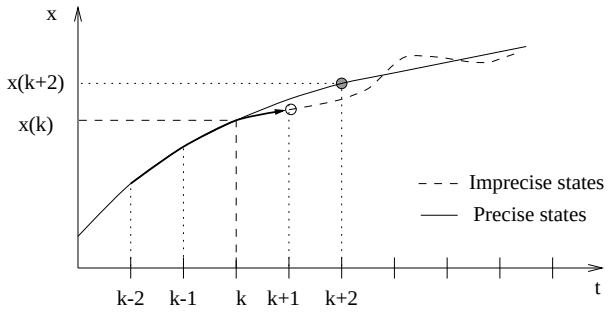

Fig. 3. Substitution of the normal algorithm by a less time consuming variant

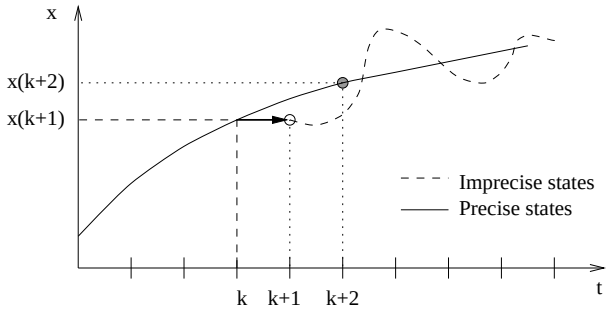

Fig. 4. Acceleration of computation by reusing the old results

For example, a discrete algorithm, calculating the linear control law

$$\begin{aligned} x(\nu+1) &= Ax(\nu) + Bu(\nu) \\ y(\nu+1) &= Cx(\nu) + Du(\nu) \end{aligned} \tag{1}$$

can be subdivided into the following four alternatives. They are listed by decreasing complexity.

Normal mode The complete algorithm with all its features is calculated (I/O operations, matrix multiplications, ...).

Substitution mode The whole control law is computed in a different, less time consuming manner. For example, an interpolation or prediction method replaces the original one to approximate the next required result. Refer to Fig. 3 to see, how this could be done.

Reduced mode The calculation of the internal controller states is omitted. Instead, the previous states

Fig. 5. Experimental setup of the vehicle convoy system

are passed to the actual ones without changes. Fig. 4 illustrates this action. The calculation of the outputs is done with the normal equations.

Exception mode The complete calculation is canceled because of very sparse available time. All results of the previous step are utilized.

Another application example for the proposed selection method is statistical programming which handles a great number of input data. When time is concise, the size of the input data set can be reduced.

3. EFFECTS OF ADAPTIVE COMPUTATION

To study the influence of imprecise computations on real applications, a practical example is chosen. This example is part of an actual research project (Richert *et al.*, 1994). One aspect of this project is to apply control system design methods to develop mechanical models and suitable controller designs. Another aim is to examine new real–time concepts. The vehicle convoy system, as shown in Fig. 5, acts as a platform to devolve and evaluate the theoretical results on a real system.

3.1 *Simulation Environment*

The concept of a real–time algorithm, which adapts oneself to the actual load situation of the control computer system, is applied to the problem of a concatenated convoy of two road vehicles. These two vehicles are connected by a rigid tow bar. The front vehicle is driven by a human person. The rear vehicle is equipped with a steering actuator.

To allow an efficient system verification, the complete convoy was modeled in a suitable modeling language (MATLAB). The model itself is composed of a number of modules. The resulting mathematical description can be used to simulate and examine the convoy dynamics. More informations about the development of the model and the identification of the system parameters can be retrieved from the publications (Rükgauer *et al.*, 1995*a*; Rükgauer *et al.*, 1995*b*; Slama and Rükgauer, 1995; Raste and Müller, 1995).

The whole convoy model takes into account 10 degrees of freedom and 14 nonlinear differential equations represent the dynamics. This multi–body system serves as a realistic representation for simulation purposes.

3.2 *Modified controller structure*

The steering actuator is connected to a control unit which tries to minimize the track deviation of the two cars automatically.

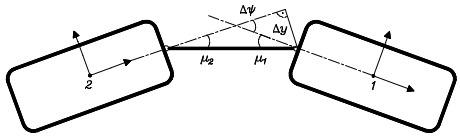

Fig. 6. Simplified description of the vehicle convoy

Inputs to the controller are the two tow bar angles μ_1 and μ_2 (see Fig. 6). The controller forwards the estimated steering angle to the corresponding actuator.

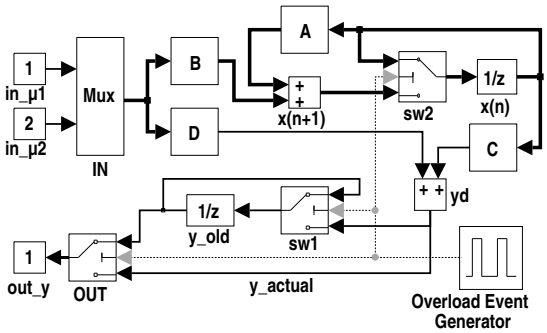

Fig. 7. Controller for the simulation of load adaptive algorithms

The control structure consists of a linear discrete system as described by equation (1) with 7 states and a

sampling rate of 500Hz. This structure was extended by a unit which allows to simulate the occurrence of transient overload situations with adjustable duration and period (Fig. 7). The implemented imprecise calculation strategy is "**Exception mode**" as mentioned in section 2.3.2.

3.3 *Simulation results*

The maneuver chosen to investigate the impact of load adaptive algorithms on the system behavior is a ISO lane change at a velocity of $20\frac{m}{s}$.

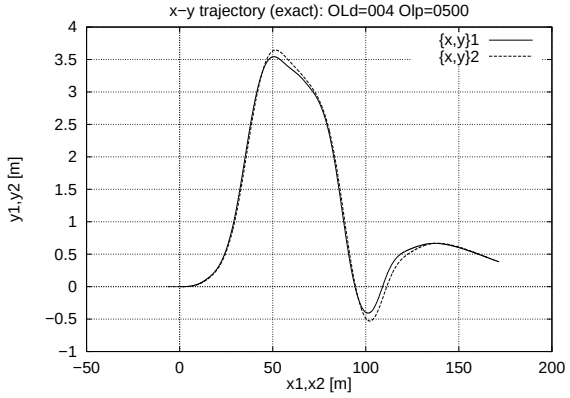

Fig. 8. Precisely controlled convoy

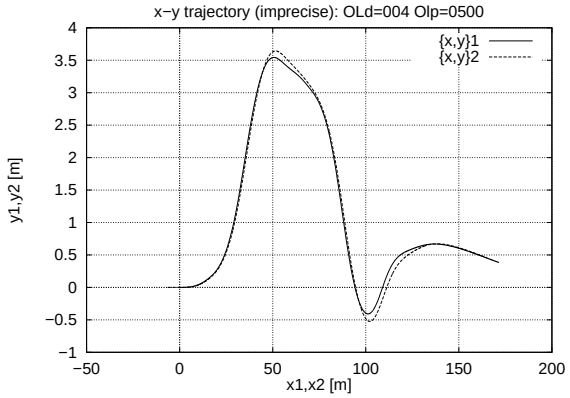

Fig. 9. Imprecisely controlled convoy

Fig. 8 respectively 9 shows the behavior of the system with a precise respectively imprecise control law. The imprecise calculation is activated every $500ms$ for a duration of $4ms$ (2 consecutive control loops).

The resulting deviations of the y–position (with a maximum of $6mm$) and the internal controller states are displayed in Fig. 10 and Fig. 11. The vertical spikes in Fig. 11 indicate a new corruption due to reduced control computation. From the state fault diagram (Fig. 11) one can derive, that the controller tries to recapture the normal operating states. A while after a spike the error function shows a falling behavior.

All results presented above concern to a single overload situation ($4ms$ duration with a period of $500ms$). Fig. 12 shows the context between the duration of a disturbance and the expected average error of the location y of the rear vehicle. The error shown is the averaged deviation over the full simulation period. The resulting error possesses an approximately linear dependence. The proportion of imprecise to precise calculation ranges from 0.8% ($4ms$) to 44% ($220ms$). The obtained results with a duration $\geq 100ms$ are irrelevant for practical use but prove anyhow that the loss of quality obeys always the same rules regardless what corruption length is injected.

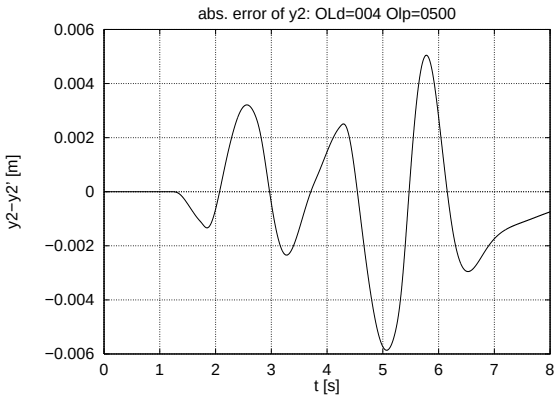

Fig. 10. Absolute error of the y–position of the rear vehicle

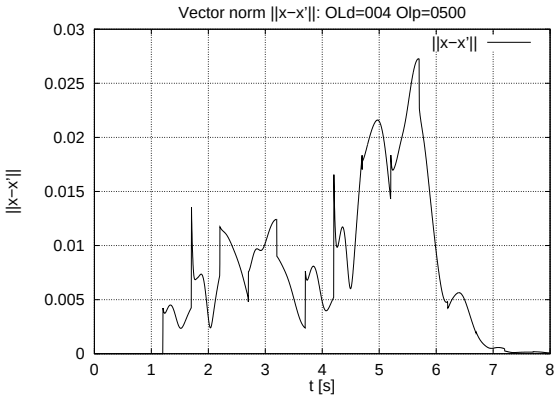

Fig. 11. Error $\| \mathbf{X}_p - \mathbf{X}_{imp} \|$ of control states

4. SUMMARY

In this paper, a possibility is proposed to design a real–time system more flexible and dependable by applying the discussed algorithm structures. The approaches presented are not difficult to implement and do not involve

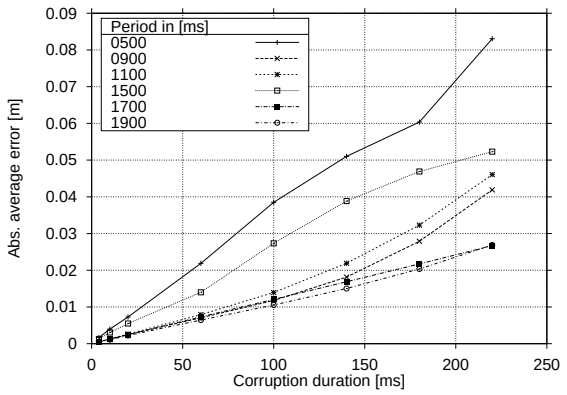

Fig. 12. Average error $\frac{1}{N}\sum_{\nu=1}^{N}|\Delta y_\nu|$ of the y–trajectory as a function of the corruption duration

much temporal overhead. Instead of viewing errors as special case, one can suggest that an error is a type of inaccuracy which can be tolerated in many applications. This is especially true, when a system is highly dynamic with regard to aperiodic, sporadic events or the availability is more important than the accuracy. Of course, there exist applications which require an absolute exact result to operate correctly and the use of the proposed methods is not recommended. For the future, it will be necessary to verify the proposed structures on additional algorithm types to confirm a global applicability.

In the second chapter of the paper, the imprecise control mechanism has been applied to a practical example. The simulation results prove that the proposed concepts can be used to solve real–time problems where the load of the executing instance is not known a priori. More detailed simulations, which were not presented here, have also shown that too many occurrences of overload situations or too long durations of them make the system unstably. But this would be the consequence of a design, which is dimensioned by nature with to less computation resources.

To apply the proposed algorithms in a real system it is necessary to equip it with a run–time system supporting requests about the time which can be consumed by a specific task. Considerations to this theme exist already from different research groups and are also part of the current investigations of the project.

5. REFERENCES

Basu, S. K. (1980). On development of iterative programs from function specifications. *IEEE Transactions on Software Engineering* **SE–6**(2), 170–182.

Dolev, D., M. Klawe and M. Rodeh (1982). An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms* **3**, 245–260.

Doyle, J., B. Frabcis and A. Tannenbaum (1992). *Feedback Control Theory*. Macmillian Publishing Company.

Gresser, Klaus (1993). An event model for deadline verification of hard real–time systems. In: *Proc. Fifth Euromicro Workshop on Real Time Systems*. IEEE. Oulu, Finland. pp. 118–123.

Gundes, A. (1992). Stability of feedback systems with sensor or actuator failures. *International Journal of Control* **56**(4), 735–754.

Liu, Jane W. S., K.-J. Lin and S. Natarajan (1987*a*). Imprecise results: Utilizing partial computations in real–time systems. In: *Proc. 8th Real–Time Systems Symposium*. IEEE. San Jose, CA. pp. 210–217.

Liu, Jane W. S., K.-J. Lin and S. Natarajan (1987*b*). Scheduling real–time, periodic jobs using imprecise results. In: *Proc. 8th Real–Time Systems Symposium*. IEEE. San Jose, CA. pp. 252–260.

Liu, Jane W. S., K.-J. Lin, S. Natarajan and T. Krauskopf (1987*c*). Concord: A system of imprecise computations. In: *Proc. COMSAC*. IEEE. Tokyo. pp. 75–81.

Raste, T. and P.C. Müller (1995). Modelling and control of mechatronic systems by decentralized descriptor systems. In: *Proc. Third Conference on Mechatronics and Robotics* (J. Lückel, Ed.). Teubner, Stuttgart. Paderborn, Germany. pp. 432–445.

Richert, J., A. Rükgauer, U. Petersen, V. Hadwich, T. Raste, K. Gresser and J. Pfefferl (1994). Integration verteilter Systeme der Mechatronik mit besonderer Berücksichtigung des Echtzeitverhaltens. Interdisciplinary Research Report Az.: I/67975-9. Uni–GH Paderborn. Fachbereich 10 Automatisierungstechnik Prof. Dr.-Ing. J. Lückel.

Rükgauer, A., U. Petersen and W. Schielen (1995*a*). Lateral dynamics of towed vehicles. In: *Proceedings of the 2nd ROVA International Conference* (C.O. Nwagboso, Ed.). Bolton, England.

Rükgauer, A., U. Petersen and W. Schielen (1995*b*). Mechatronic steering of a convoy vehicle. In: *Proc. Third Conference on Mechatronics and Robotics* (J. Lückel, Ed.). Teubner, Stuttgart. Paderborn, Germany. pp. 403–416.

Slama, L. and A. Rükgauer (1995). Dynamic models for simulation studies of the power steering of road vehicles. *Engineerings Mechanics* **2**(5), 309–318.

Thielen, Herbert (1994). Automated design of distributed computer control systems with predictable timing behaviour. In: *Proc. 12th IFAC Workshop on Distributed Computer Control Systems* (J. A. de la Puente and M. G. Rodd, Eds.). IFAC. Toledo, Spain. pp. 47–52.