



Dependability of Explicit DMC and GPC Algorithms

Piotr Gawkowski¹, Maciej Ławryńczuk², Piotr Marusak², Janusz Sosnowski¹,
and Piotr Tatjewski²

¹ Institute of Computer Science

{P.Gawkowski, J.Sosnowski}@ii.pw.edu.pl

² Institute of Control and Computation Engineering

{M.Lawrynczuk, P.Marusak, P.Tatjewski}@ia.pw.edu.pl

Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

Abstract. This paper studies dependability of software implementation of DMC (Dynamic Matrix Control) and GPC (Generalised Predictive Control) Model Predictive Control (MPC) algorithms. Explicit formulation of algorithms is considered in which the control laws are calculated off-line. Dependability is evaluated using software implemented fault injection approach. Tests are performed in the control system of a remotely controlled robot vehicle used in nuclear plants.

1 Introduction

Faults (transient, permanent or intermittent) appearing during system operation may result in logical errors, which can be critical for the realised applications [1, 8]. Transient faults are especially critical as they dominate in contemporary technologies. Hence, an important practical issue is to evaluate dependability of software applications in the presence of faults. It is particularly critical in many reactive systems (e.g. nuclear plants, satellites, aircrafts, chemical industry, medicine). One kind of such applications, erroneous behaviour of which might have some serious consequences, are control algorithms. This paper studies dependability of software implementations of explicit versions of DMC (Dynamic Matrix Control) and GPC (Generalised Predictive Control) Model Predictive Control (MPC) algorithms. The investigation is based on software implemented fault injection, which has been adapted to reactive applications [1, 5]. In particular, a new approach to test result qualification is proposed.

MPC is recognised as the only advanced control technique which has been very successful in practical applications [11–15, 19, 20]. As MPC algorithms use models of processes for calculation of the control policy they can be successfully applied to processes which are difficult to control. In particular, processes with significant time delay for which the PID controller does not give satisfactory control performance. Among different MPC techniques, DMC [3] and GPC algorithms [2] are the most popular. Both algorithms use linear models. The DMC algorithm uses a non-parametric model consisting of step-response coefficients

of the process. Such a model can be easily obtained in industry, but to precisely describe a process many coefficients are needed. The GPC algorithm uses a parametric model in the form of a discrete difference equation. Usually, such models are significantly less complicated in terms of the number of parameters than the step-response ones.

The paper is structured as follows. First, in Section 2, the investigated control algorithms and their explicit formulations are shortly characterised. Next, Section 3 presents the fault injection testbed, experiment set-up and some new aspects of adapting experiments to control algorithms specificity. Finally, Section 4 discusses experimental results and the paper is concluded in Section 5.

2 Model Predictive Control Algorithms

In the MPC algorithms [12–15, 19] at each consecutive sampling instant k a set of future control increments

$$\Delta \mathbf{u}(k) = [\Delta u(k|k) \ \Delta u(k+1|k) \ \dots \ \Delta u(k+N_u-1|k)]^T \quad (1)$$

is calculated assuming that $\Delta u(k+p|k) = 0$ for $p \geq N_u$, where N_u is the control horizon. It is usually done in such a way that the future control error values (i.e. differences between the reference trajectory and the predicted values of the output) are minimised over the prediction horizon N . The following quadratic cost function is typically used

$$J(k) = \|\mathbf{y}^{ref}(k) - \hat{\mathbf{y}}(k)\|^2 + \|\Delta \mathbf{u}(k)\|_{\mathbf{A}}^2 \quad (2)$$

where $\mathbf{y}^{ref}(k) = [y^{ref}(k) \ \dots \ y^{ref}(k)]^T$, $\hat{\mathbf{y}}(k) = [\hat{y}(k+1|k) \ \dots \ \hat{y}(k+N|k)]^T$ are vectors of length N , $\mathbf{A} = \text{diag}(\lambda_0, \dots, \lambda_{N_u-1})$, $\lambda_p > 0$ are weighting factors, $y^{ref}(k)$ is the reference (i.e. the set-point) and $\hat{y}(k+p|k)$ are predicted values of the output.

Typically, $N_u < N$, which decreases the dimensionality of the optimisation problem and leads to smaller computational load. Because only the first element of the determined sequence (1) is applied to the process, the control law is

$$u(k) = u(k-1) + \Delta u(k|k) \quad (3)$$

At next sampling instant, $k+1$, the prediction is shifted one step forward and the whole procedure is repeated. Predicted output values $\hat{y}(k+p|k)$ are calculated using a dynamic model of the process.

When a linear dynamic model of the process is used, it is possible to express the output prediction as the sum of a forced trajectory (which depends only on the future input moves $\Delta \mathbf{u}(k)$) and a free trajectory $\mathbf{y}^0(k)$ (which depends only on the past)

$$\hat{\mathbf{y}}(k) = \mathbf{y}^0(k) + \mathbf{G}\Delta \mathbf{u}(k) \quad (4)$$

where $\mathbf{y}^0(k) = [y^0(k+1|k) \ \dots \ y^0(k+N|k)]^T$. The dynamic matrix \mathbf{G} of dimensionality $N \times N_u$ is comprised of step-response coefficients of the model.

Thanks to using a linear model and the superposition principle (4), optimisation of the cost function (2) becomes a quadratic programming task. Hence, the vector of optimal control input increments is

$$\Delta \mathbf{u} = \mathbf{K}(\mathbf{y}^{ref}(k) - \mathbf{y}^0(k)) \tag{5}$$

where $\mathbf{K} = (\mathbf{G}^T \mathbf{G} + \Lambda)^{-1} \mathbf{G}$ is a matrix of dimensionality $N_u \times N$ which is calculated off-line.

2.1 Dynamic Matrix Control Algorithm

In the DMC algorithm the process dynamics is described, in a convenient way, by a discrete-time, finite step-response model. Thus, for any sampling instant k , output of the model is

$$y(k) = y(0) + \sum_{j=1}^D s_j \Delta u(k-j) \tag{6}$$

where s_i are step-response coefficients, D is the horizon of the process dynamics.

The DMC control law (5) can be expressed in the following form [12, 19]

$$\Delta u(k) = \Delta u(k|k) = k^e (y^{ref}(k) - y(k)) - \sum_{j=1}^{D-1} k_j^u \Delta u(k-j) \tag{7}$$

where $k^e, k_j^u, j = 1, \dots, D-1$ are coefficients calculated off-line. The total number of parameters is D . The obtained explicit control law is a linear feedback from the difference between the set-point trajectory and values of the manipulated variable increments calculated at previous sampling instants. The structure of the explicit DMC algorithm is shown in Fig. 1.

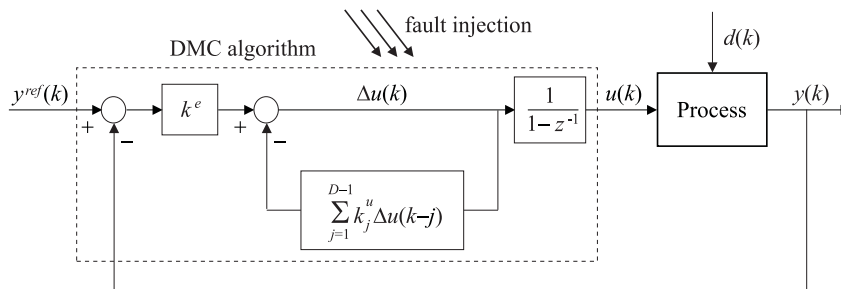


Fig. 1. Structure of the explicit DMC algorithm

2.2 Generalised Predictive Control Algorithm

The GPC algorithm uses a process model in the form of a discrete difference equation describing the process input-output relation

$$y(k) = \sum_{i=1}^{n_B} b_i u(k-i) - \sum_{i=1}^{n_A} a_i y(k-i) \tag{8}$$

where a_i, b_i are coefficients and n_A, n_B define order of the dynamics.

The GPC control law can be expressed in the following form [19]

$$\Delta u(k) = \Delta u(k|k) = k^e y^{ref} - \sum_{j=1}^{n_B} k_j^u u(k-j) - \sum_{j=0}^{n_A} k_j^y y(k-j) \tag{9}$$

where $k^e, k_j^u, j = 1, \dots, n_B$ and $k_j^y, j = 0, \dots, n_A$ are coefficients calculated off-line. The total number of parameters is $n_A + n_B + 2$. The obtained explicit GPC control law is a linear feedback from the reference trajectory, values of the manipulated variable calculated at previous sampling instants and values of the controlled variable measured at previous sampling instants. Structure of the explicit GPC algorithm is shown in Fig. 2. Comparing the structures of both studied algorithms, it is evident that in the DMC algorithm, there is only one feedback from the process output variable and $D - 1$ feedbacks from values of past process input increments. In the GPC algorithm, there are feedbacks from the current process output value and last n_A past values of the output increments and from last n_B past values of the process input increments.

The step-response usually contains a large number of elements. At the same time, the process can be described precisely enough by a discrete difference equation of a relatively low order. Hence, a model used in the GPC algorithm has significantly less parameters than the model used in the DMC algorithm (i.e. $D \gg n_A, D \gg n_B$). It should be stressed, however, that a non-parametric step-response model is obtained on the basis of a simple experiment but it is necessary to conduct a full identification experiment to obtain a parametric model.

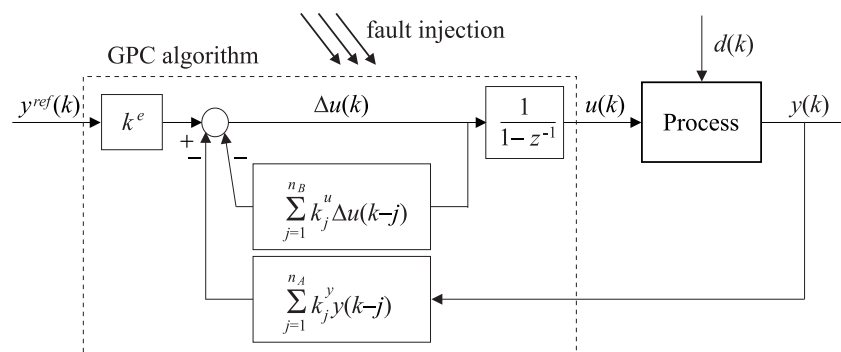


Fig. 2. Structure of the explicit GPC algorithm

3 Fault Injector and Experiment Set-up

The concept of the SoftWare Implemented Fault Injector (SWIFI) is based on the software emulation of a fault during the run-time of the application under test. In this research FITS fault injector is used [7, 17]. FITS is used for many years in research projects and proves to be the one of the most complex, flexible and easy to use SWIFI tool reported in the literature. It uses standard Win32 Debugging API to control the execution of the software application under test. In the whole process the following steps can be distinguished: optional source code instrumentation of the tested application, golden run execution, experiment configuration, fault injection and results analysis.

In order to assure good experiment controllability, the so called *testing areas* are introduced [5]. FITS disturbs directly the application only within those areas. They can be marked in several ways (e.g. with predefined magic sequence at the source code level by subset of code modules). Testing areas can limit the scope of disturbances only to the most interesting parts of the application, making further analysis easier. Additionally, some code within the application can be situated outside disturbance area during experiments. Here, the code of the controlled process model is added. The parts of the tested applications disturbed during the experiments (dashed box) as well as process models (not disturbed) are marked in Fig. 1 and 2. To simplify tracing fault effects some extra modifications of the tested application are introduced. The *user-messages* captured and collected by the FITS during experiments are inserted [17]. This mechanism provides supplementary communication between the tested application and the FITS. As a result, the tested application signals some measures related to internal variables values, output signal deviations etc.

During the so called *Golden Run* (GR – reference execution without faults) the execution trace and reference results are logged. Additionally, statistic information is collected on the tested application (e.g. resource usage, code size, instruction distribution). Those measures help to interpret experimental results and to profile experiments to be done (e.g. by elimination of injecting faults into unused resources). FITS simulates faults by disturbing the running application. In this process an important issue is policy of selection the type, location and time of fault injection (fault triggering). In this study single bit-flip faults within CPU and FPU registers, applications' data and machine instruction code are considered. Faults are injected pseudorandomly in time (program execution) and space (bit position within disturbed resource, distribution over applications' memory). There is a common consensus in the literature that such fault model well mimics Single Event Upset (SEU) effects. The experience also shows that such disturbances give the most interesting results for the fault susceptibility analysis [8]. Deeper discussion upon the fault injection policy can be found in [6, 8, 16].

An important issue (frequently neglected in the literature) is qualification of experimental results. In case of a typical calculation-oriented application correctness of its result is usually easy. It is much more complicated for other classes of applications, especially related to real-time systems [16]. Control algorithms

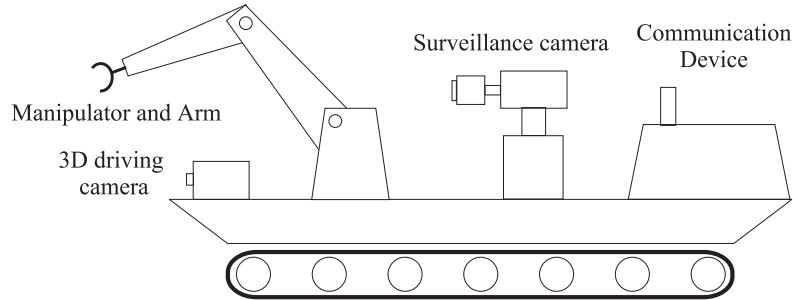


Fig. 3. Remotely controlled robot for nuclear plants

require complex analysis of the controlled process behaviour. In FITS the oracle procedure (in an external library) can be specified to examine the tested application outputs in accordance with an application's specificity [8, 16, 17].

3.1 Experiment set-up

The process under consideration is a remotely controlled robot vehicle for nuclear plants [4] shown in Fig. 3. The vehicle must act reliably in an hazardous and unsafe environment. Application of such vehicles can significantly reduce radiation exposure to personnel and improve maintenance programme performance. Discrete-time model of the process is (the sampling time is 0.5 min)

$$y(k) = b_9 u(k-9) - b_{10} u(k-10) - a_1 y(k-1) - a_2 y(k-2) \quad (10)$$

where $b_9 = 0.022276$, $b_{10} = 0.019823$, $a_1 = -1.683638$, $a_2 = 0.704688$. Input u of the controlled process (output of the control algorithm in Fig. 1 and 2) is the voltage applied. The vehicle model (*Process* box in Fig. 1 and 2) limits the voltage range to $[-5; 5]$. Output y of the process is the velocity of the vehicle. Values of y at each sampling instant are considered to be the result of the whole application and are subject to correctness analysis. The input of the whole application (y^{ref}) is the required vehicle's velocity, a single step from 0 to 1 occurring at $k = 10$ is considered during the tests. The DMC algorithm uses the step-response model obtained from the model (10) (the horizon of the process dynamics $D = 100$) whereas the GPC algorithm uses directly the model (10). In both algorithms $N = 20$, $N_u = 5$ and $\lambda_p = 1$.

The whole experiment is conducted by FITS automatically. At the end of the experiment synthetic (aggregated) results for each fault location are given. In general, 4 classes of test results are distinguished:

- C: correct vehicle behaviour ($ISE < 20$),
- INC: incorrect (unacceptable) vehicle behaviour ($ISE \geq 20$),
- S: test terminated by the system due to un-handled exception,
- T: timed-out test.

where the standard factor ISE (Integrated Sum of Squared Errors) is proposed as a measure of result (y) correctness. The reference ISE value (obtained during GR) is 12.79 (due to delayed vehicle response). System exceptions (S) are generated by hardware mechanisms embedded in contemporary COTS (commercial off-the-shelf) systems, e.g. memory access violation detection.

Analysis of fault effects requires detail information upon the faults injected and the application behaviour. FITS can provide details about every test (simulated fault injection) that allow manual replay the whole test execution. Moreover, all the events and user messages occurring during the test are recorded. The tested application is instrumented to save its outputs (here simulation results, i.e. a set of control signals in subsequent sampling instants) into separate files for each test (file names managed by FITS). This gives a possibility for deeper analysis (post-experiment) of fault effects in the correlation with the injected fault and observed behaviour for each single test.

4 Experimental Results

The cost of the DMC algorithm implementation is 173 bytes of binary code (50 machine instructions). At the same time, the GPC algorithm implementation takes 212 bytes (59 instructions). The main difference is the number of executed instructions, i.e. a single simulation execution of the DMC and GPC algorithms takes 121261 and 12192 machine instructions, respectively. Such a result is not surprising as the control law used in the DMC algorithm has 100 parameters whereas the GPC control law has only 14 parameters.

Fig. 4 depicts summarised results of experimental evaluation of DMC and GPC algorithms (results categories are described in Section 3). The main difference can be seen in the case of faults located in the data area used by the considered algorithms. As the DMC algorithm uses much more parameters (100) than the GPC one (14), it is more insensitive to single disturbances.

It is worth noting, that both algorithms can be further hardened by introducing exception handling. In the experiments carried out, no exception handling is present. Hence, approximately 50% of faults affecting the static image of the code, executed instruction stream and CPU registers resulted in unhandled exceptions (most of them relate to memory access violations). Previous experience shows [6, 8–10, 16, 17] that such behaviour can be efficiently improved.

A great number of tests have been performed (more than 25000 per algorithm) and their results compared. Because of limited space only two time-plots are presented here. Fig. 5 compares simulation results of the golden run (solid lines) and two example incorrect vehicle behaviour (dashed lines). In the first case (top figures), fault injected into static code of the DMC algorithm implementation results in slow response and big control errors (ISE=31.44). It leads to unacceptable output response (slow and with big control errors). Moreover, the manipulated variable (u) oscillations appear (the top left plot). In the second case (bottom figures) the responses from a different experiment with the DMC algorithm are shown (fault also injected into the static code). This time violation

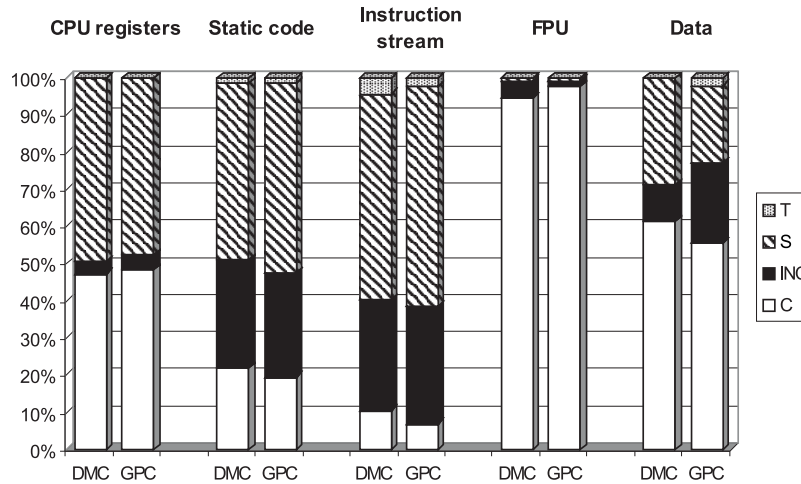


Fig. 4. Summary of the experimental results

of the manipulated variable constraint is observed (as shown in the bottom left graph). It would result in an oscillatory movement of the robot forward and back repeatedly.

Fig. 6 shows the distribution of ISE values observed (faults located in the static code) for tests considered as unacceptable (INC). It is worth noting that both the DMC and GPC algorithms have similar distribution for low ISE values (< 100). Moreover, more than 66% of the INC tests have very high ISE values (≥ 100), which means that the control system is unstable. Obviously, other correctness measures should be also considered and need further development and investigation.

5 Conclusions

The paper presents a novel approach to evaluation and comparison of dependability of software implementation of two most popular MPC algorithms, i.e. DMC and GPC. For this purpose software implemented fault injector is used. Explicit formulation of these algorithms is considered in which the control laws are calculated off-line. The experiments show that the performance of both algorithms is different in the case of some faults (in data) as well as their faults susceptibility. It is worth noting that robustness of software implementation can be improved by some basic software fault detection/tolerance techniques. The new measures of process behaviour are also considered to be developed. Those topics will be covered in the future research.

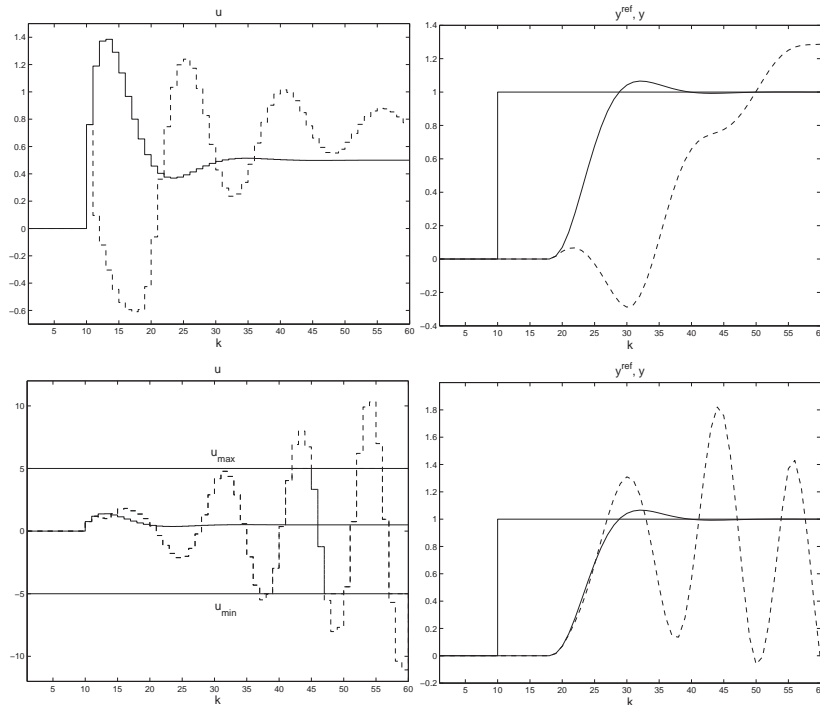


Fig. 5. DMC algorithm simulation results: **top:** slow response and big control errors (ISE=31.44); **bottom:** manipulated variable u constraint violation (ISE=23.94); solid line – golden run, dotted line – incorrect vehicle behaviour

Acknowledgement. This work was supported by a research grant from the dean of the Faculty of Electronics and Information Technology, Warsaw University of Technology.

References

1. Benso A., Prinetto P.: *Fault injection techniques and tools for embedded systems reliability evaluation*, Kluwer Academic Publishers. (2003).
2. Clarke D. W., Mohtadi C., Tuffs P. S.: *Generalized predictive control—I*, The basic algorithm. *Automatica*. **23** (1987) 137–148.
3. Cutler R., Ramaker B.: *Dynamic matrix control—a computer control algorithm*, AIChE National Meeting, Houston. (1979).
4. Dorf C. D.: *Modern control systems*, Addison-Wesley. Reading. (1995).
5. Gawkowski P., Sosnowski J.: *Experiences with software implemented fault injection*, Int. Conf. on Architecture of Computing Systems. VDE Verlag GMBH. (2007) 73–80.
6. Gawkowski P., Sosnowski J., Radko B.: *Analyzing the effectiveness of fault hardening procedures*, 11th IEEE Int. On-Line Testing Symposium. (2005) 14–19.

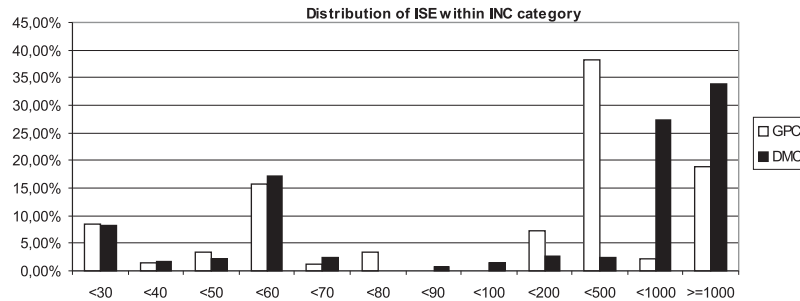


Fig. 6. Distribution of ISE in incorrect category

7. Gawkowski P., Sosnowski J.: *Analysing system susceptibility to faults with simulation tools*, Annales UMCS Informatica AI. **4** (2006) 123–134.
8. Gawkowski P., Sosnowski J.: *Dependability evaluation with fault injection experiments*, IEICE Transactions on Information & System. **E86-D** (2003) 2642–2649.
9. Gawkowski P., Sosnowski J.: *Experimental Validation Of Fault Detection And Fault Tolerance Mechanisms*, 7th IEEE Int. Workshop on High Level Design Validation And Test. Cannes (2002).
10. Gawkowski P., Sosnowski J.: *Analyzing fault effects in fault insertion experiments*, Proc. of On-line Testing Workshop. IEEE Computer Society Press (2001) 21–24.
11. Ławryńczuk M., Marusak M., Tatjewski P.: *Multilayer and integrated structures for predictive control and economic optimisation*, IFAC Symposium on Large Scale Systems, Gdańsk (2007), accepted for publication.
12. Maciejowski J. M.: *Predictive control with constraints*, Prentice Hall. Harlow. (2002).
13. Morari M., Lee J. H.: *Model predictive control: past, present and future*, Computers and Chemical Engineering. **23** (1999) 667–682.
14. Qin S. J., Badgwell T. A.: *A survey of industrial model predictive control technology*, Control Engineering Practice. **11** (2003) 733–764.
15. Rossiter J. A.: *Model-based predictive control*, CRC Press, Boca Raton. (2003).
16. Sosnowski J., Gawkowski P., Lesiak A.: *Fault injection stress strategies in dependability analysis*, Control and Cybernetics. **33** (2005) 679–699.
17. Sosnowski J., Lesiak A., Gawkowski P., Włodawiec P.: *Software implemented fault inserters*, IFAC Workshop on Programmable Devices and Systems. Ostrava (2003) 293–298.
18. Sosnowski J., Gawkowski P., Lesiak A.: *Fault injection stress strategies. 4th IEEE LATW 2003 Workshop*, (2003) 258–263.
19. Tatjewski P.: *Advanced control of industrial processes, Structures and algorithms*, Springer. London. (2007).
20. Tatjewski P., Ławryńczuk M., Marusak M.: *Linking nonlinear steady-state and target set-point optimisation for model predictive control*, IEE Int. Control Conference ICC 2006, Glasgow (2006), CD-ROM.