SCIENCE AND TECHNOLOGY

Paweł SKRUCH
Marek DŁUGOSZ
Wojciech MITKOWSKI

MATHEMATICAL METHODS FOR VERIFICATION OF MICROPROCESSOR-BASED PID CONTROLLERS FOR IMPROVING THEIR RELIABILITY

MATEMATYCZNE METODY TESTOWANIA MIKROPROCESOROWYCH REGULATORÓW PID UMOŻLIWIAJĄCE ZWIĘKSZENIE ICH NIEZAWODNOŚCI*

Proportional-Integral-Derivative (PID) control is the most common control algorithm used in industry. The extensive use of electronics and software has resulted in the situation where the digital PID controller using a microprocessor as well as its software implementation replaces existing pneumatic, mechanical and electromechanical solutions. The reliability of the software system is assured by detection and removal of errors that can lead to failures. The paper presents mathematical methods for verification and testing of microprocessor-based PID controllers that can be used to increase the reliability of the system. The presented methodology explores the concept of testing with a model as an oracle.

Keywords: *controller, PID, testing, reliability.*

Regulator PID (regulator proporcjonalno-całkująco-różniczkujący) jest najbardziej rozpowszechnionym i najczęściej stosowanym typem regulatora w przemyśle. Intensywny rozwój elektroniki i informatyki spowodował, że cyfrowe regulatory PID budowane na bazie mikroprocesora z odpowiednim oprogramowaniem zastąpiły dotychczasowe rozwiązania pneumatyczne, mechaniczne i elektromechaniczne. Zagwarantowanie niezawodności układu elektronicznego z oprogramowaniem polega między innymi na wykrywaniu i usuwaniu błędów, które mogą prowadzić do awarii. W pracy przedstawiono matematyczne metody weryfikacji mikroprocesorowych regulatorów PID mające na celu wykrycie błędów w systemie i w konsekwencji zwiększenie jego niezawodności poprzez zmniejszenie prawdopodobieństwa wystąpienia awarii. Metody testowania opierają się na tak zwanym podejściu modelowym, to znaczy, wykorzystują model systemu jako wzorzec zachowania.

Słowa kluczowe: *regulator, PID, testowanie, niezawodność.*

1. Introduction

Proportional-Integral-Derivative (PID) control is the most common control algorithm used in industry. It has been in use for over a century in various forms: as a purely mechanical device, as a pneumatic device and as an electronic device.

Modern digital PID controller is a system that can be considered as a combination of computer hardware and software designed to perform a dedicated control function. The control is implemented on a custom hardware platform, which is often designed and configured for the particular application. Such systems are called embedded systems [29, 30]. Embedded systems may be observed in common devices employed in everyday living (e.g., coffee machines, washing machines, cell phones) as well as in sophisticated engineering systems (e.g. cars [4, 29], planes, spacecrafts).

PID controllers are also often safety critical systems. Due to the area of application, the PID controller must have high reliability as unexpected failures can be fatal. Ensuring the reliability of embedded software systems based on the detection and removal of errors that can lead to system failure. The process to verify that the system meets the specified requirements is referred to as testing. Testing is also the process of trying to discover every conceivable fault or weakness in a work product [12, 14].

The most common errors that can lead to improper operation of control devices equipped with the software include functional errors in the code, arithmetic errors associated with the use of fixed-point arithmetic, communication and task management errors, lack of robustness to different types of disturbances and work outside the scope of the variability of input signals.

There are several facts that show clearly possible consequences of poorly tested systems. On February 25, 1991, an Iraqi Scud hit the barracks in Dhahran in Saudi Arabia, killing 28 soldiers from the US Army. This accident was caused by software error in the system's clock [24]. The PATRIOT missile battery has been in operation for 100 hours, by which time the system's internal clock had drifted by one third of a second. For a target moving as fast as Scud, this was equivalent to a position error of 600 meters. Another example is connected with Therac-25 radiation therapy machine that was produced by Atomic Energy of Canada Limited and CGR of France. The machine was involved with at least six known accidents between 1985 and 1987, in which patients were given massive overdoses of radiation, which were in some cases on the order of hundreds of grays [18]. At least five patients died of the overdoses. These accidents were caused by errors in software control application. One of the most infamous computer bugs in history was found during flight 501 that took place on June 4, 1996. This was the first, and unsuccessful, test flight of the European Ariane 5 expendable lunch system. Due to an error in the software design (inadequate protection from integer overflow), the rocket veered off its flight path 37 seconds after launch and was destroyed by its automated self-destruct system [19]. As it was an unmanned flight, there were no victims, but the breakup caused the loss of four Cluster mission spacecraft, resulting in a loss of more than $370 million.

There are two basis classes of software testing: black box testing and white box testing. Black box testing is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. White box testing is testing that takes into account the internal mechanism of a system or component.

Black box testing is also called functional testing [2, 3] or specification-based testing. The specification for control systems can be very often presented in the form of models. Test cases should be then generated systematically out of the models [29, 30]. The most popular black box testing techniques include boundary value analysis, equivalence partitioning, decision table testing, state transition testing and use case testing (see e.g. [3, 22]).

Boundary values analysis is a testing technique in which tests are designed to include boundary values of input functions to stimulate the system. The idea comes from the boundary, which is the area where testing is likely to yield defects. Equivalence partitioning is a technique that divides the input data into groups that are expected to exhibit similar behavior, so they can are likely to be processed in the same way. The groups are called equivalence partitions (or classes) and can be also identified for outputs, interval values and parameters. Decision tables are a good way to capture system requirements that contain logical conditions. State transition testing is much used within the embedded software industry and technical automation where the system behavior can be represented using state diagrams. Tests can be also specified from use cases or business scenarios. A use case is a sequence of steps that describe the interactions between an actor (a user of the system) and the system.

White box testing (also called structure-based testing) is based on an identified structure of the software or system. The structure can be considered as the code itself (i.e., statements, decisions or branches), a call tree (a diagram in which modules call other modules), a menu structure, business process or web page structure. Test cases designed with the help of white box testing techniques take into account such input values to cover relevant instruction in the code (instruction testing), decisions (decision testing), conditions, etc.

It should be emphasized that most of the presented techniques and methods are seldom applicable in testing software systems where the dynamics cannot be neglected [20, 27]. The dynamical systems are modeled by difference or differential equations and have usually infinitely many states. There is a need for another approach that will han-

dle continuous aspects of the system (see e.g. [6, 17]). This is because of testing dynamics aspects of such systems requires tests that utilize time continuous input signals and time continuous output signals (even when the system is digitally processed). The process of selecting just a few of the many possible scenarios to be tested is a difficult and challenging task and currently is most often based on qualitative best engineering judgment. Some results [5, 11, 15, 28] developed for hybrid systems can be also applicable to dynamical systems and to fractional-order systems (see e.g. [21]) which recently are of interest to many scientists and engineers.

The paper is organized as follows. In the next section, modeling concepts of the functionality realized by the PID controller are introduced. These concepts are explored further in the next sections and are the base for creating test artifacts such as: test oracle (Section 3), notation of tests (Section 4), implementation of a test comparator (Section 5), test coverage (Section 6) and test generation (Sections 7 and 8). Experimental results are given in Section 9. Conclusions are in Section 10.

2. Mathematical description of the PID controller

An embedded PID controller is a system that can be considered as a combination of computer hardware and software designed to perform a dedicated control function. The PID controller (Fig. 1) works in a closed-loop system and attempts to minimize the error e(t) by adjusting the control input $u(t)$. The error is calculated as the difference between a measured process output $y(t)$ and a desired set point $y_{sp}(t)$. The control signal is a result of the following calculation

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\mathrm{d}\tau + T_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}\right), \qquad (1)$$

where K is proportional gain, T_i is integral time, T_d is derivative time. The control signal is thus a sum of three terms: the P-term (which is proportional to the error), the I-term (which is proportional to the integral of the error), and D-term (which is proportional to the derivative of the error).
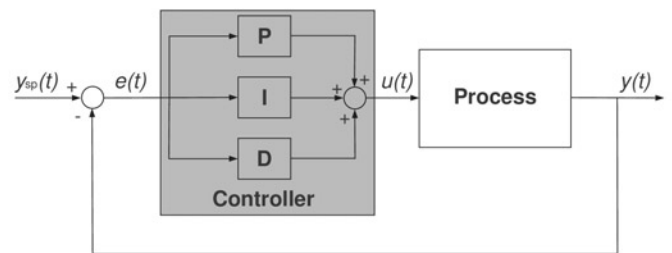

Fig. 1. Block diagram of the closed-loop system with the PID controller

Introduce the notation $w_1(t) = \int_0^t e(\tau)\mathrm{d}\tau$ and $w_2(t) = \dot{w}_1(t)$ the equation (1) can be written as:

$$u(t) = Kw_2(t) + \frac{K}{T_i}w_1(t) + KT_d\dot{w}_2(t), \qquad (2)$$

or, equivalently, in matrix notation as:

$$\dot{\boldsymbol{w}}(t) = \boldsymbol{E}\boldsymbol{w}(t) + \boldsymbol{F}u(t), \boldsymbol{w}(0) = \boldsymbol{0}, \qquad (3)$$

where $\boldsymbol{w}(t) = \begin{bmatrix} w_1(t) & w_2(t) \end{bmatrix}^{\mathrm{T}} \in W \subset \mathbb{R}^2$

$$E = \begin{bmatrix} 0 & 1 \\ -(T_i T_d)^{-1} & -T_d^{-1} \end{bmatrix}, \quad F = \begin{bmatrix} 0 \\ (KT_d)^{-1} \end{bmatrix}. \quad (4)$$

The physical and implementation constraints imposed by computer system resources lead to the assumption that the space W is bounded. The assumption means that the space W is contained in a circle of finite radius.

3. Concept of testing with a model as an oracle

The formulas (1) and (3) specify mathematically the system's behavior in clear and unambiguous form. It can be used in computer simulations in an early phase of development to validate the system concept, calibrate parameters and optimize the system performance. In the next phase, the physical system is designed (i.e., hardware and software) that shall meet the specified requirements in the form of the equations (1), (3). Testing process shall be considered as the last phase in the development process that allows verifying that the physical system behavior is identical to that observed during computer simulations. If the tests fail then the system needs to be redesigned. The physical system that is being tested for the correct operation is often referred as system under test (SUT).

The term test oracle describes a source to determine expected results to compare with the actual result of the SUT [1]. The role of such source in the model-based approach is often played by the model (see Fig. 2). The approach stipulates that the same input is applied to both the SUT and to the model. The input signal is physical in case of the SUT (e.g., voltage, current or resistance) and virtual in case of the model; from logical point of view both signals are equivalent. The judgment whether the result of a test is in conformance with the model is delegated to a test comparator. The test comparator is usually a tool that compares the actual output produced by the SUT with the expected output produced by the model.
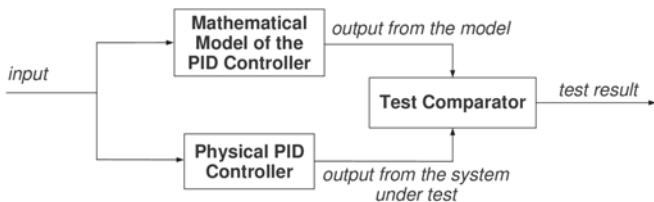

Fig. 2. Concept of testing with a model as an oracle

4. Notation of tests

One of the fundamental tasks of software testing is the creation of test cases. A test case can be considered as a set of inputs, execution preconditions and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement [13].

Adapting this definition to the SUT modeling concepts (1), a single test case $T_{case}^{(j)}$ can be defined as:

$$T_{case}^{(j)} = \left\{ T^{(j)}, e^{(j)}(\cdot), u^{(j)}(\cdot) \right\} \quad (5)$$

where $j=1,2,\ldots,N$, $N \geq 1$ is a label to indicate different test cases, $e^{(j)}: [0,T^{(j)}] \rightarrow \mathbb{R}$ is an input function, $u^{(j)}: [0,T^{(j)}] \rightarrow \mathbb{R}$ is an expected output function, $T^{(j)}$ stands for test execution time. Notation (5) and the model (1) play the key role in the test selection method presented in Section 8.

When the system model is described by the equation (3), then a single test case $T_{case}^{(j)}$ can be presented in the form

$$T_{case}^{(j)} = \left\{ T^{(j)}, u^{(j)}(\cdot), w^{(j)}(\cdot) \right\} \quad (6)$$

where $u^{(j)}: [0,T^{(j)}] \rightarrow \mathbb{R}$ is an input function unlike the notation (5) and $w^{(j)}: [0,T^{(j)}] \rightarrow \mathbb{R}^2$ is an expected output function. Notation (6) with the model (3) are the base for the test selection method presented in Section 7.

A collection of one or more test cases forms a test suite T_{suite}, where:

$$T_{suite} = \left\{ T_{case}^{(1)}, T_{case}^{(2)}, \ldots, T_{case}^{(N)} \right\}. \quad (7)$$

5. Implementation of a test comparator

The test comparator can be considered as a tool that implements a mechanism for determining whether a test passes or fails [14]. In the concept illustrated on Fig. 2, this tool compares the actual output produced by the SUT with the expected output produced by the model (1) or (3).

A possible practical realization of the comparison function for a given test case (5) is presented below:

$$z\left(T_{case}^{(j)}\right) = \begin{cases} 0 \text{ if } \forall_{t \in [0,T^{(j)}]} \left| u^{(j)}(t) - u_s^{(j)}(t) \right| < \varepsilon \left| u^{(j)}(t) \right|, \\ 1 \text{ otherwise,} \end{cases} \quad (8)$$

where z denotes the test result, that is, $z=0$ when the test passes, $z=1$ when the test fails, $\varepsilon > 0$ is the tolerance range, $u_s^{(j)}(\cdot)$ stands for the output produced by the SUT.

In the similar way, the comparison function can be defined for notation (6):

$$z\left(T_{case}^{(j)}\right) = \begin{cases} 0 \text{ if } \forall_{t \in [0,T^{(j)}]} \| w^{(j)}(t) - w_s^{(j)}(t) \| < \varepsilon \| w^{(j)}(t) \|, \\ 1 \text{ otherwise,} \end{cases} \quad (9)$$

where $w_s^{(j)}(\cdot)$ is the output produced by the SUT.

6. Calculation of test coverage

The degree to which a given test suite T_{suite} addresses all specified requirements for a given system is determined by a test coverage measure [13]. The most obvious quantification of the system's behavior exercised by the test suite is computed by dividing the number of the system states explored by the test suite by the cardinality of the entire state space. However, the formula has limited usefulness for dynamical systems (and PID controller belongs to this class of systems) because the state space for such systems contains usually infinite number of states.

In following part of this section it is presented a method for calculation of test coverage that was taken from the paper [25]. The method described therein has been adapted to the model (3). The test coverage $C_h(T_{suite})$ of the test suite T_{suite} can be defined as follows:

$$C_h(T_{suite}) = \frac{\left| \bigcup_{j=1}^{j=N} V_h\left(T_{case}^{(j)}\right) \right|}{|W_h|} \quad (10)$$

where:

$$W_h = \left\{ \boldsymbol{i} = [i_1, i_2]^{\mathrm{T}} \in \mathbb{Z}^2 : \exists_{\boldsymbol{w} \in W} \boldsymbol{w} \in G_h(\boldsymbol{i}) \right\} \qquad (11)$$

is the transformed state space created from the system state space W,

$$G_h(\boldsymbol{i}) = \left\{ \boldsymbol{w} \in \mathbb{R}^2 : \boldsymbol{w} = [w_1, w_2]^{\mathrm{T}}, \frac{w_k}{h_k} = i_k, k = 1, 2 \right\}$$

denotes a partition with the size $\boldsymbol{h} = [h_1, h_2]^{\mathrm{T}}, h_1, h_2 > 0$, $\left\lceil \frac{w_k}{h_k} \right\rceil$ is the

largest integer greater than $\frac{w_k}{h_k}$,

$$V_{\boldsymbol{h}}\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right) = \left\{ \boldsymbol{i} \in W_{\boldsymbol{h}} : \exists_{t \in [0, T^{(j)}]} \boldsymbol{w}^{(j)} \in G_{\boldsymbol{h}}(\boldsymbol{i}) \right\} \qquad (13)$$

is a set of states of the transformed state space covered by the test case $\mathrm{T}_{\mathrm{case}}^{(j)}$. It should be noticed that the sum

$$V_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) = \bigcup_{j=1}^{j=N} V_{\boldsymbol{h}}\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right) \qquad (14)$$

will contain the information about the states covered by the test suite $\mathrm{T}_{\mathrm{suite}}$.

In the example presented on Fig. 3, bounded two-dimensional internal state space W (the area embraced by the bold solid line) has been transformed to the space

$$W_{\boldsymbol{h}} = \left\{ \boldsymbol{i} = [i_1, i_2]^{\mathrm{T}} \in \mathbb{Z}^2, i_1 = 0, 1, \ldots, 8, i_2 = 0, 1, \ldots, 4 \right\} \setminus \\ \left\{ [0,0]^{\mathrm{T}}, [1,0]^{\mathrm{T}}, [8,0]^{\mathrm{T}}, [0,1]^{\mathrm{T}}, [0,4]^{\mathrm{T}}, [7,5]^{\mathrm{T}}, [8,4]^{\mathrm{T}} \right\}. \qquad (15)$$

that consists of 45 elements $G_{\boldsymbol{h}}(\boldsymbol{i}) = [i_1, i_1 + 1) \times [i_2, i_2 + 1)$ with the size of 1×1. Fig. 3 contains also system trajectories related to two exampled test cases: $\mathrm{T}_{\mathrm{case}}^{(1)}$ and $\mathrm{T}_{\mathrm{case}}^{(2)}$. 10 grid boxes are visited by the system trajectory belonging to the first test case, 9 – to the second test case. The test suite $\mathrm{T}_{\mathrm{suite}} = \left\{ \mathrm{T}_{\mathrm{case}}^{(1)}, \mathrm{T}_{\mathrm{case}}^{(2)} \right\}$ consisting of these test cases covered in total 17 grid boxes what implies the test coverage at level $C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) = \frac{17}{45} \approx 0.45(45\%)$.

7. A test selection method for conformance testing

In this section, an algorithm for generating test cases is presented. The algorithm uses the modeling concept (3) of the SUT to generate test cases and calculate test coverage according to the method presented in the previous section. It explores transformed state space by using input signals that steer the system from an initial state to a final state. The selection and completeness of test cases is quantified by the coverage metric (10). The main idea of the presented strategy is to check that the functional specification in the form of the equation (3) is correctly implemented, which is variously referred to in the literature as conformance testing [14], correctness testing [16] or functional testing [13].
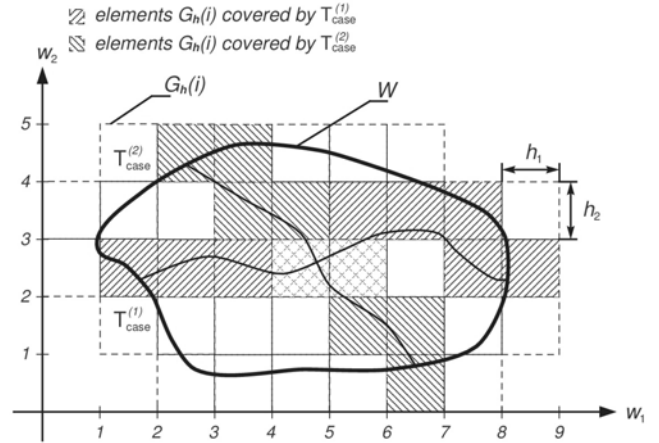

Fig. 3. Illustration of the test coverage for the state space W

Algorithm 1

1°: Set the parameters: $\boldsymbol{h} = [h_1, h_2]^{\mathrm{T}}, h_1, h_2 > 0, \delta \in (0,1], T > 0$

2°: $\mathrm{T}_{\mathrm{suite}} := \varnothing$, $V_h(\mathrm{T}_{\mathrm{suite}}) := \varnothing$, $C_h(\mathrm{T}_{\mathrm{suite}}) := 0$, $j := 1$

3°: **while** $C_h(\mathrm{T}_{\mathrm{suite}}) < \delta$ do

4°: Find $\boldsymbol{w}_a \in G_h(i_a)$ where $i_a \in W_h \setminus V_h(\mathrm{T}_{\mathrm{suite}})$

5°: Calculate the control function $u(\cdot)$ that steers the system from the zero initial state to the final state $\boldsymbol{w}_a$

6°: Calculate the trajectory $\boldsymbol{w}(t) = \int_0^T e^{(t-\tau)\boldsymbol{E}} \boldsymbol{F} u(\tau) \mathrm{d}\tau$ by solving the equation (3)

7°: $\mathrm{T}_{\mathrm{case}}^{(j)} = \left\{ T^{(j)}, u^{(j)}(\cdot), \boldsymbol{w}^{(j)}(\cdot) \right\}$ where $T^{(j)} \cdot T, u^{(j)}(\cdot) := u(\cdot), \boldsymbol{w}^{(j)}(\cdot) := \boldsymbol{w}(\cdot)$

8°: $\mathrm{T}_{\mathrm{suite}} := \mathrm{T}_{\mathrm{suite}} \cup \mathrm{T}_{\mathrm{case}}^{(j)}$

9°: Calculate $V_h(\mathrm{T}_{\mathrm{suite}})$ and $C_h(\mathrm{T}_{\mathrm{suite}})$

10°: $j := j + 1$

11°: **end while**

Remark 1. The size $\boldsymbol{h} = [h_1, h_2]^{\mathrm{T}}$ of the partition can be chosen according to the formula

$$h_i \frac{\max_{t \geq 0} |w_i(t)|}{10}, i = 1, 2. \qquad (16)$$

For safety critical systems there would be recommended to decrease the granulation of the partitions h_i. However, it should be clear that too small granulation significantly increases the number of test cases and overall testing effort.

Remark 2. The system (3) is controllable as the rank of the controllability matrix is equal to the size of the system, that is, rank$[\boldsymbol{E} \ \boldsymbol{EF}] = 2$ (see e.g. [20]). This means that there exist generally many different controls which steer the system from the zero initial state to the final state $\boldsymbol{w}_a$ at time $T > 0$. For example, minimum energy control [20] is probably the easiest computable control steering the system to a desired state under the assumption that the constraints posed on the system are not violated.

8. A test selection method for negative testing

In this section, the test selection problem is formulated as an optimization problem. Representative test cases are constructed during optimization procedure using the model (1). The test selection is combined with the test execution and these two activities are conducted at the same time. The main advantage of the approach is focus on

error prone situations that leads to drastically reduced number of representative test cases.

The problem is to find a test case $T_{\text{case}} = \{T, e(\cdot), u(\cdot)\}$ which is a result of the optimization procedure

$$\max_{e \in E_{\text{ad}}} J(e) = \max_{e \in E_{\text{ad}}} \int_0^T (u(t) - u_s(t))^2 \, dt \qquad (17)$$

where E_{ad} stands for the set of admissible error functions. The set E_{ad} can be correlated with physical and implementation constraints imposed by computer system resources.

Algorithm 2

1°: Set the parameters: E_{ad} and $T > 0$

2°: Run the optimization procedure for the problem (16) to obtain the solution $e^* \in E_{\text{ad}}$

3°: Calculate the control signal $u^*(\cdot)$ using the equation (1) for the error signal $e^*(\cdot)$

4°: $T_{\text{case}} := \{T, e^*(\cdot), u^*(\cdot)\}$.

9. Experimental results

In order to evaluate the efficiency and usability of the presented algorithms as well as their ability to find faults they were applied to the real system. The faults in the form of incorrect parameters of the PID controller have been deliberately introduced to the system implementation. For better illustration of the results the parameters have been modified by 20% from the correct values. In practice, these faults can be caused by the use of fixed-point arithmetic; they can also result from errors in the identification procedure and can be a direct consequence of programmer error. Introduction of incorrect parameters values to the control system can result in different time to reach the steady state than expected, larger overshoot in the system and in the worst case in instability of the closed-loop system. Good control quality depends strongly on the correct settings what is especially important in optimal control problems [7] applicable, for example, for electric motors [8] and internal combustion engines [26].

Consider the model (1) of the PID controller with the following parameters

$$K = 3.60, \quad T_i = 1.81, \quad T_d = 0.45. \qquad (18)$$

Next, the functionality described by the equation (1) has been implemented in software, which runs in a microprocessor on the embedded hardware platform, however with incorrect values of the parameters, that is

$$\tilde{K} = 2.88, \quad \tilde{T}_i = 2.17, \quad \tilde{T}_d = 0.36. \qquad (19)$$

The entire system has been tested with the help of the algorithm 1 which has been implemented and executed for the following input parameters: $\boldsymbol{h} = [0.3, 0.2]^T$, $\delta = 0.7$, $T = 20$ [s], $|w_1(t)| \leq 1.5$, $|w_2(t)| \leq 1.0$ (system implementation constraints). The test suite that guarantees the coverage level higher that δ consits of 10 test cases. Elements of the generated test cases of the form of (6) are graphically presented in Figs. 4 and 5. Comparison of the actual trajectory obtained from the SUT with the expected trajectory is shown in Fig. 6. The output from the SUT for the first test case is not within the tolerance range $\varepsilon = 0.1$ relative to the expected output, therefore the test case is qualified as *fail*. This proofs existence of the fault in the system.

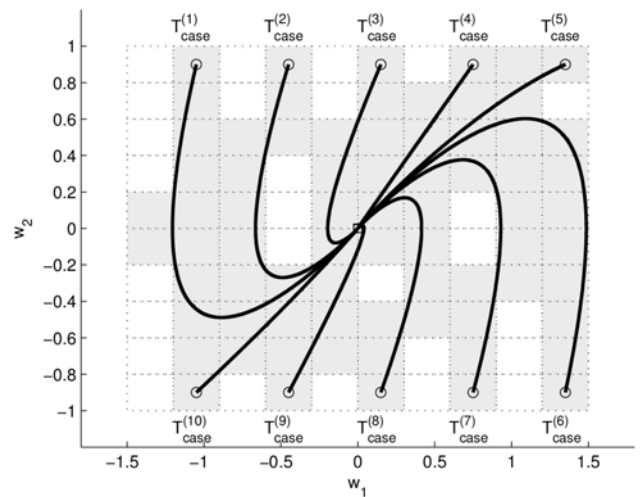Consider the following set of admissible error functions:

Fig. 4. Trajectories $w^{(j)}$, $j = 1, 2, ..., 10$ and elements (gray rectangles) of the transformed state space W_h covered by the test cases $T_{\text{case}}^{(j)}$. The model trajectories start in □ and end in ○
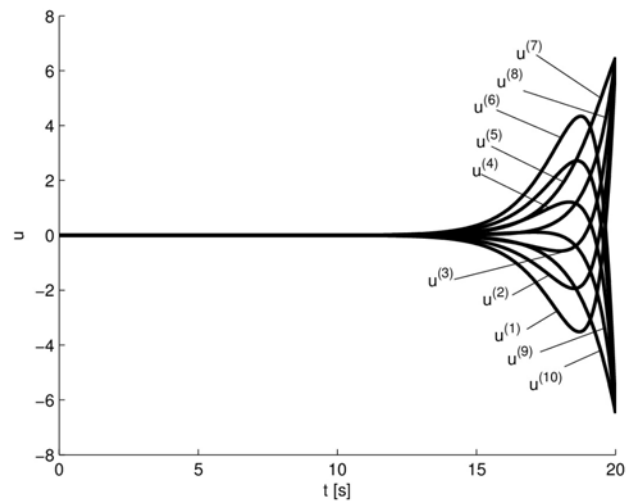

Fig. 5. Input functions for the test cases $T_{\text{case}}^{(j)}$, $j = 1, 2, ..., 10$
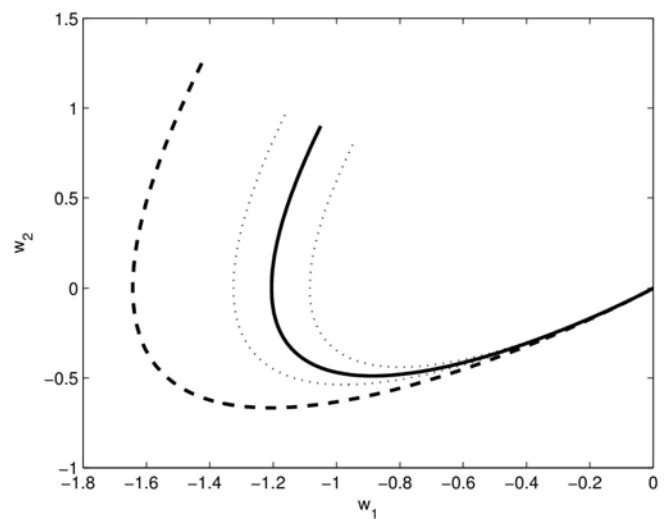

Fig. 6. The comparison of the output function $w_s^{(1)}$ produced by the tested PID system (dotted line) with the expected output function $w^{(1)}$ (solid line) produced by the model. The limits of tolerance of 10% are marked on the drawing by a thin dotted line
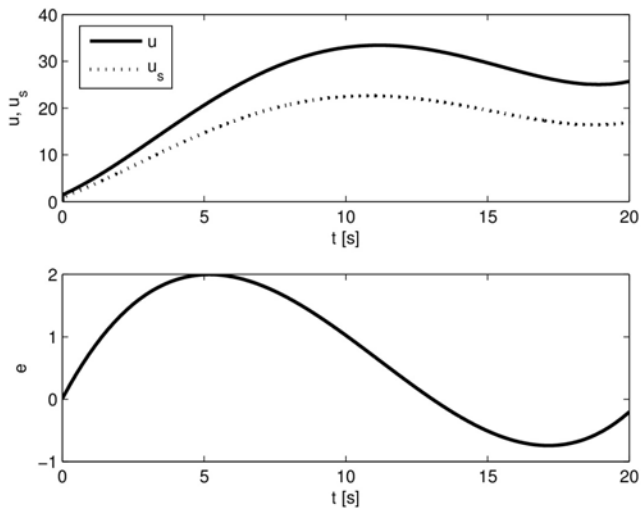
Fig.7. The elements of the test case generated with the help of the algorithm 2

$$E_{\mathrm{ad}} = \left\{ e \in \mathrm{PC}\left([0,T],\mathbb{R}\right) : e(t) = \alpha_0 t^3 + \alpha_1 t^2 + \alpha_2 t + \alpha_3, \alpha_i \epsilon \,\mathbb{R}, \left|e(t)\right| \le 2, i = 0,1,2,3 \right\}$$

$$(20)$$

that can be used in the optimization procedure (algorithm 2) to find such test cases that maximize the difference (17) between the outputs produced by the tested system and its model within the time T. The implementation of the algorithm with the Nelder-Mead simplex (direct method) [23] leads to the following local optimal solution:

$$e(t) = 0.0032 t^3 - 0.1072 t^2 + 0.8534 t + 0.0089 . \qquad (21)$$

Elements of the generated test cases of the form of (5) are graphically presented in Fig. 7. The figure includes also for comparison purposes the actual trajectory obtained from the tested system.

The main advantage of the testing method based on the algorithm 2 is a significant reduction of test cases, which the search is done us-

ing the optimization procedure. The algorithm focuses on error prone situations. As a result, the time and cost associated with the testing of the system can be significantly reduced. Since the effort put into testing is, according to estimates [2], from 30 up to 90 percentage of the overall effort in the projects, the benefits coming from even a very small reduction of this factor can be very profitable. It should be also noted that the algorithm 2 performs the search for test cases while using the physical system and its mathematical representation. Thus, to start the process of testing both the model and the real system are required for. Moreover, the formulation described in the algorithm 2 takes the form of a functional optimization problem, which may appear difficult to solve as it requires transformation to a value optimization problem.

10. Conclusions

The paper has presented two different methods for testing embedded PID controllers to provide required quality of the system, assure compliance with safety standards and eliminate errors at the stage of system design. Elimination of errors in the early stages of product development can increase system reliability and reduce the risk of failures during the operational phase. All elements of the testing process (i.e., concept of testing, notation of test cases, implementation of a test comparator, test coverage, selection of test cases) have been formulated and described in using the appropriate mathematical notation. The key role in the presented approach plays the mathematical model that represents intended behavior of the designed system. In this way it was possible to develop methods for testing systems where the dynamics plays an important role and where classical testing techniques cannot be applied to.

The presented approach can be easily generalized to other microprocessor-based control systems. Controllers with dynamic compensator [27], electric motor controllers [7, 8], controllers of internal combustion engines, neural networks controllers [9] and fuzzy logic controllers [10] are examples of the systems that can be verified using the algorithms described in this paper.

References

1. Adrion W, Brandstad J, Cherniabsky J. Validation, verification and testing of computer software. Computing Survey 1982; 14(2): 159-192, http://dx.doi.org/10.1145/356876.356879.
2. Beizer B. Software Testing Techniques, 2nd ed. Boston: Van Nostrand Reinhold, 1990.
3. Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems. New York: John Willey & Sons, 1995.
4. Chłopek Z, Biedrzycki J, Lasocki J, Wójcik P. Assessment of the impact of dynamic states of an internal combustion engine on its operational properties. Eksploatacja i Niezawodnosc – Maintenance and Reliability 2015; 17(1): 35-41, http://dx.doi.org/10.17531/ein.2015.1.5.
5. Dang T. Model-based testing of hybrid systems. In: Model-Based Testing for Embedded Systems. Boca Raton: CRC Press 2011; 383-423, http://dx.doi.org/10.1201/b11321-15.
6. Dang T, Nahhal T. Coverage-guided test generation for continuous and hybrid systems. Formal Methods in System Design 2009; 34(2): 183-213, http://dx.doi.org/10.1007/s10703-009-0066-0.
7. Długosz M. Problemy optymalizacyjne układów napędowych robotyki. Przeglad Elektrotechniczny – Electrical Review 2011; 87(9a): 238-242.
8. Długosz M, Lerch T. Komputerowa identyfikacja parametrów silnika prądu stałego. Przeglad Elektrotechniczny – Electrical Review 2010; 85(2): 34-38.
9. Długosz R, Kolasa W, Pedrycz M, Szulc M. Parallel programmable asynchronous neighborhood mechanism for Kohonen SOM implemented in CMOS technology. IEEE Transactions on Neural Networks 2011; 22(12): 2091-2104, http://dx.doi.org/10.1109/TNN.2011.2169809.
10. Długosz R, Pedrycz W. Łukasiewicz fuzzy logic networks and their ultra low power hardware implementation. Neurocomputing 2010; 73(7-9): 1222-1234, http://dx.doi.org/10.1016/j.neucom.2009.11.027.
11. Esposito J. Automated test trajectory for hybrid systems. Proceedings of the 35th Southeastern Symposium on System Theory 2003; 441-444, http://dx.doi.org/10.1109/SSST.2003.1194609.
12. IEEE Std 1012-2004. IEEE standard for software verification and validation, 2004.
13. IEEE Std 61012-1990. IEEE standard glossary of software engineering terminology, 1990.
14. ISTQB International Software Testing Qualification Board. Standard glossary of terms used in software testing, version 2.1, 2010.

15. Julius A, Fainekos G, Anand M, Lee I, Pappas G. Robust test generation and coverage for hybrid systems. Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control (HSCC), Pisa 2007; 329-342, http://dx.doi.org/10.1007/978-3-540-71493-4_27.
16. Kaner C, Faulk J, Nguyen H. Testing Computer Software, 2nd ed. New York: John Willey & Sons, 1995.
17. LaValle S, Kuffner J. Rapidly-exploring random trees: progress and prospects. In: Algorithmic and Computational Robotics: New Directions 2001; 293-308.
18. Leveson N, Turner S. An investigation of the Therac-25 accidents. IEEE Computer 1993; 27(7): 18-41; http://dx.doi.org/10.1109/MC.1993.274940.
19. Lions J. ARIANE 5. Flight 501 failure. Ariane 501 Inquiry Board Report, Paris, 1996.
20. Mitkowski W. Stabilizacja systemów dynamicznych. Warszawa: WNT, 1991.
21. Mitkowski W, Skruch P. Fractional-order models of the supercapacitors in the form of RC lader networks. Bulleting of the Polish Academy of Sciences, Technical Sciences 2013; 61(3): 581-587, http://dx.doi.org/10.2478/bpasts-2013-0059.
22. Myers G. The Art of Software Testing, 2nd ed. New York: John Willey & Sons, 2004.
23. Nelder J, Mead R. A simplex method for function minimization. The Computer Journal 1965; 7(4): 308-313, http://dx.doi.org/10.1093/comjnl/7.4.308.
24. Skeel R. Roundoff error and the Patriot missile. Society for Industrial and Applied Mathematics (SIAM) News 1992; 25(4): 11.
25. Skruch P. A coverage metric to evaluate tests for continuous-time dynamic systems. Central European Journal of Engineering 2011; 1(2): 174-180, http://dx.doi.org/10.2478/s13531-011-0015-8.
26. Skruch P. An educational tool for teaching vehicle electronic system architecture. International Journal of Electrical Engineering Education 2011; 48(2): 174-183, http://dx.doi.org/10.7227/IJEEE.48.2.5.
27. Skruch P: Feedback stabilization of a class of nonlinear second-order systems. Nonlinear Dynamics 2010; 59(4): 681-692, http://dx.doi.org/10.1007/s11071-009-9570-4.
28. Tabuada P. Verification and Control of Hybrid Systems. Dordrech: Springer, 2009, http://dx.doi.org/10.1007/978-1-4419-0224-5.
29. Zander-Nowicka J. Model-based testing of real-time embedded systems in the automotive domain. PhD thesis. Berlin: Fraunhofer IRB Verlag, 2009.
30. Zander J, Schieferdecker I, Mosterman P. (Eds) Model-Based Testing for Embedded Systems. Boca Raton: CRC Press, 2012.

Paweł SKRUCH
Marek DŁUGOSZ
Wojciech MITKOWSKI
Department of Automatics and Biomedical Engineering
AGH University of Science and Technology
Al. A. Mickiewicza 30/B1, 30-059 Kraków, Poland
E-mails: pawel.skruch@agh.edu.pl, mdlugosz@agh.edu.pl,
wojciech.mitkowski@agh.edu.pl