Dr inż. Paweł Skruch

Dr inż. Marek Długosz

Prof. dr hab. inż. Wojciech Mitkowski

Katedra Automatyki i Inżynierii Biomedycznej
Akademia Górniczo-Hutnicza
Al. A. Mickiewicza 30/B1, 30-059 Kraków, Polska
E-mail: pawel.skruch@agh.edu.pl, mdlugosz@agh.edu.pl, wojciech.mitkowski@agh.edu.pl

Matematyczne metody testowania mikroprocesorowych regulatorów PID umożliwiające zwiększenie ich niezawodności

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

Streszczenie: 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.

1. Wprowadzenie

Regulator proporcjonalno-całkująco-różniczkujący (PID) jest najbardziej rozpowszechnionym i najczęściej stosowanym typem regulatora w przemyśle. Regulator PID jest stosowany od blisko stu lat w różnych formach: jako urządzenie czysto mechaniczne, pneumatyczne, elektromechaniczne, a ostatnio także jako urządzenie cyfrowe. Współczesny cyfrowy regulator PID jest specjalizowanym układem elektronicznym wyposażonym w oprogramowanie realizujące określoną funkcję sterowania. Realizacja sterowania odbywa się z reguły na niestandardowej platformie sprzętowej, która jest bardzo często skonstruowana i skonfigurowana na potrzeby danego przypadku. Regulatory PID są też często systemami krytycznymi ze względu na bezpieczeństwo. Tego typu systemy określa się mianem systemów wbudowanych [29, 30]. Systemy wbudowane są obecne we wszystkich dziedzinach począwszy od skomplikowanych urządzeń badawczych, poprzez aparaturę pomiarową, sterowniki stosowane w motoryzacji [4, 29] i lotnictwie aż do popularnego sprzętu gospodarstwa domowego. Ze względu na obszary zastosowań, regulator PID musi charakteryzować się dużą niezawodnością określoną np. poprzez prawdopodobieństwo wystąpienia awarii. Zagwarantowanie niezawodności elektronicznego regulatora PID polega między innymi na wykrywaniu i usuwaniu błędów, które mogą prowadzić do awarii systemu. Proces sprawdzania układu w celu zweryfikowania czy spełnia on wyspecyfikowane wymagania określa się mianem testowania. Testowanie to również działanie zmierzające do wykrywania błędów w układzie [12, 14].

Do najczęściej spotykanych błędów, które mogą przyczynić się do nieprawidłowej pracy urządzeń sterujących wyposażonych w oprogramowanie należy zaliczyć błędy funkcjonalne związane z niewłaściwą implementacją, błędy arytmetyczne i związane z zastosowaniem arytmetyki stałoprzecinkowej, błędy związane z komunikacją oraz z zarządzaniem zadaniami, brak odporności systemu na zakłócenia i pracę poza zakresem zmienności sygnałów wejściowych.

Istnieje wiele udokumentowanych przypadków, które pokazują, że konsekwencje wynikające z błędów w oprogramowaniu w systemach sterujących mogą być bardzo poważne. Jednym z takich przypadków jest tragedia w koszarach w Dhahranie w Arabii Saudyjskiej, w które 25 lutego 1991 roku uderzyła iracka rakieta typu SCUD zabijając 28 amerykańskich żołnierzy. Ten wypadek był spowodowany błędem w systemie obrony przeciwlotniczej PATRIOT [24]. Do obliczania celu pocisku był wykorzystywany zegar, który liczył czas w zmiennej całkowitej. Zmienna ta w celu dalszych obliczeń była rzutowana na liczbę zmiennoprzecinkową. Liczby zmiennoprzecinkowe mają taką własność, że małe liczby są zapamiętywane z dużą dokładnością, a wielkie liczby z niewielką dokładnością. Bateria rakiet PATRIOT pracowała nieprzerwalnie przez 100 godzin co spowodowało, że błąd zegara wynosił około 1/3 sekundy. Błąd ten przy prędkościach pocisków balistycznych spowodował błąd obliczenia celu pocisku o około 600 metrów. Inny przypadek jest związany z maszyną do radioterapii Therac-25, która została wyprodukowana przez kanadyjską firmę Atomic Energy i francuską CGR. Między 1985 a 1987 rokiem odnotowano przynajmniej 6 przypadków, w których pacjentom podano za dużą dawkę promieniowania w efekcie czego 5 pacjentów zmarło w wyniku choroby popromiennej [18]. Wypadki te były spowodowane w błędami w oprogramowaniu sterującym pracą maszyny. Niewątpliwie najbardziej spektakularny efekt błędu komputerowego w historii miał miejsce w czasie lotu testowego europejskiej rakiety Ariane 5. Po 37 sekundach lotu rakieta zeszła z kursu i została zniszczona przez system autodestrukcji [19]. Błąd był spowodowany brakiem zabezpieczenia w oprogramowaniu podczas rzutowania 64-bitowej zmiennej zmiennoprzecinkowej na 16-bitową zmienną całkowitą. Jako, że był to lot bezzałogowy nie było ofiar śmiertelnych, ale katastrofa spowodowała straty szacowane na 370 milinów dolarów.

Testy dzieli się generalnie na dwa typy: „białej skrzynki” i „czarnej skrzynki”. Testy czarnoskrzynkowe to takie, w których tworzenie przypadków testowych opiera się na założeniach funkcjonalnych jakie powinien spełniać układ zgodnie z wymaganiami. Testy białoskrzynkowe to takie, w których tworzenie przypadków testowych opiera się na budowie wewnętrznej programu, który realizuje określone funkcje układu.

Techniki czarnoskrzynkowe zwane są również technikami opartymi na specyfikacji [2, 3]. Specyfikacja dla układów automatyki to przede wszystkim modele matematyczne. Przypadki testowe powinny być zatem wytwarzane systematycznie na podstawie modeli [29, 30]. Do najważniejszych technik czarnoskrzynkowych zalicza się analizę wartości brzegowych, podział na klasy równoważności, testowanie w oparciu o tablicę decyzyjną, testowanie przejść pomiędzy stanami, testowanie w oparciu o przypadki użycia (zob. np. [3, 22]).

Technika projektowania przypadków testowych w oparciu o analizę wartości brzegowych polega na wykorzystaniu wartości brzegowych funkcji sterujących do stymulacji systemu. U podstaw tej metody leży przypuszczenie, że na brzegu przestrzeni sterowania system może zachowywać się nieprawidłowo. Podział na klasy równoważności jest techniką polegającą na podziale przestrzeni sterowania lub przestrzeni wyjścia na podzbiory, dla których zakłada się na podstawie specyfikacji, że zachowanie układu będzie takie samo lub podobne. Podzbiory te nazywa się klasami równoważności. Można je wyznaczyć w oparciu o wartości wejścia i wyjścia układu, parametry układu oraz jego własności. Zbudowanie tak zwanej tablicy decyzyjnej, czyli tabelarycznego ujęcia wszystkich warunków, jakie mogą

zdarzyć się w systemie wraz z działaniami, jakie należy podjąć, pomaga w testowaniu układów sterowanych zdarzeniami i opisywanych warunkami logicznymi. Testowanie przejść pomiędzy stanami sprawdza się w układach, które opisuje się za pomocą tak zwanych maszyn stanów. Taka forma opisu jest bardzo często stosowana w układach wbudowanych i automatyce. Typowym przykładem takiego opisu są wszelkiego rodzaju automaty, czyli iteracyjne modele zachowania się systemu oparte o tablicę dyskretnych przejść między stanami. Testy mogą być też tworzone w oparciu o przypadki użycia. Przypadek użycia jest sekwencją prostych kroków określających interakcję pomiędzy wymuszeniem w systemie a samym systemem.

Techniki projektowania przypadków testowych w oparciu o analizę programu realizującego określony algorytm lub określoną funkcję regulacji określa się mianem technik białoskrzynkowych. Podstawą takich technik jest poprawne zidentyfikowanie struktury oprogramowania lub systemu. Przykładowo, taką strukturą jest kod, tzn. instrukcje, decyzje i rozgałęzienia, może nią być także drzewo wywołań, czyli diagram, w którym moduły wołają inne moduły, strukturą może być także struktura menu, struktura panelu sterującego, itp. Konstruując przypadki testowe bierze się pod uwagę takie wartości wejściowe, aby uzyskać wykonanie odpowiednich instrukcji w kodzie (testowanie instrukcji), decyzji (testowanie decyzyjne), warunków, itp.

Należy podkreślić, że większość z przedstawionych technik i metod testowania nie zbyt dobrze się sprawdza w przypadku układów, w których dynamika odgrywa istotną rolę i której nie można zaniedbać [20, 27]. Jeżeli chcemy przetestować czy zbudowany układ charakteryzuję się określoną dynamiką musimy posłużyć się już bardziej rozbudowanym aparatem matematycznym i metodami (zob. np. [6, 17]), które pozwalają na sprawdzenie własności dynamicznych takich jak czas narastania, szybkość zmierzania do stanu ustalonego, itd. Niestety na chwilę obecną jest niewiele opracowań, które dotyczą testowania systemów dynamicznych. Przez systemy dynamiczne rozumiemy tutaj układy opisane równaniami różniczkowymi zwyczajnymi lub cząstkowymi, lub też układy opisane równaniami różnicowymi. Pewne rezultaty dotyczące testowania systemów hybrydowych, które można próbować zastosować do systemów czysto dynamicznych są przedmiotem prac [5, 11, 15, 28]. Podobne zagadnienia będą również dotyczyć, ostatnio zyskujących na popularności, układów niecałkowitego rzędu (zob. np. [21]).

Struktura niniejszej pracy jest następująca. W rozdziale 2 został przedstawiony matematyczny opis regulatora PID. Opis ten stanowi jednocześnie definicję wymagań funkcjonalnych dla konstruowanego fizycznego urządzenia. Koncepcja testowania oparta na modelu systemu jako wzorcowi zachowania jest tematem rozdziału 3. Kolejne rozdziały pracy opisują za pomocą odpowiedniego aparatu matematycznego najważniejsze elementy występujące w procesie testowania, tj. notację testów (rozdział 4), sposób określania wyników testów (rozdział 5) i pokrycie testowe (rozdział 6). Algorytmy wyboru przypadków testowych zostały opisane w rozdziałach 7 i 8. Przedstawione algorytmy oparte są o tak zwane testy zgodności i testy negatywne. Rozdział 9 zawiera przykład ilustrujący poruszane w pracy zagadnienia. Rozdział 10 zawiera podsumowanie.

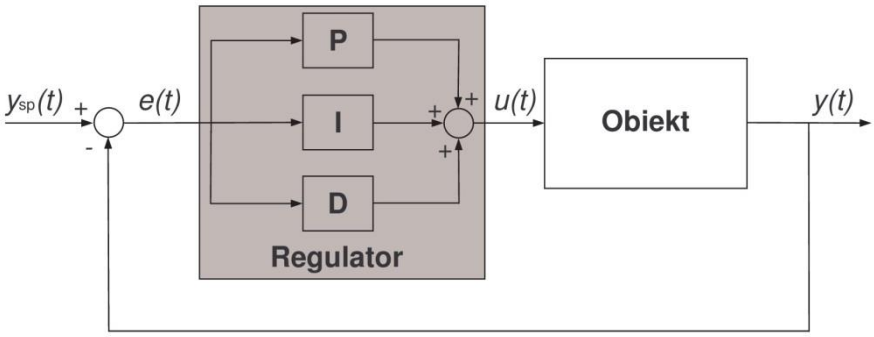
2. Matematyczny opis regulatora PID

Mikroprocesorowy regulator PID jest systemem elektronicznym wyposażonym w oprogramowanie realizujące określoną funkcję sterowania. Regulator PID pracuje w pętli sprzężenia zwrotnego (rys. 1) i działa w taki sposób, aby zredukować uchyb $e(t)$ poprzez odpowiednie dostosowanie sygnału sterującego $u(t)$ podawanego na wejście regulowanego obiektu. Uchyb jest obliczany jako różnica pomiędzy wartością zmiennej wyjściowej obiektu $y(t)$ a pożądaną wartością zadaną $y_{\mathrm{sp}}(t)$. Sygnał sterujący jest wynikiem wyliczenia

zawierającego trzy oddzielne człony: proporcjonalny „P”, całkujący „I” oraz różniczkujący „D”, to znaczy

$$u(t) = K\left(e(t) + \frac{1}{T_\mathrm{i}}\int_0^t e(\tau)\,\mathrm{d}\tau + T_\mathrm{d}\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right), \tag{1}$$

gdzie K oznacza wzmocnienie członu proporcjonalnego, T_i jest czasem całkowania dla członu całkującego, T_d oznacza czas różniczkowania dla członu różniczkującego.



Rys. 1. Schemat blokowy zamkniętego układu sterownia z regulatorem PID

Wprowadzając nowe oznaczenia $w_1(t) = \int_0^t e(\tau)\,\mathrm{d}\tau$ oraz $w_2(t) = \dot{w}_1(t)$ równanie (1) będzie można zapisać w postaci

$$u(t) = Kw_2(t) + \frac{K}{T_\mathrm{i}}w_1(t) + KT_\mathrm{d}\dot{w}_2(t)\,, \tag{2}$$

lub równoważnie w notacji macierzowej

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

przy czym $\boldsymbol{w}(t) = [w_1(t) \quad w_2(t)]^\mathrm{T} \in W \subset \mathbb{R}^2$,

$$\boldsymbol{E} = \begin{bmatrix} 0 & 1 \\ -(T_\mathrm{i}T_\mathrm{d})^{-1} & -{T_\mathrm{d}}^{-1} \end{bmatrix},\ \ \boldsymbol{F} = \begin{bmatrix} 0 \\ (KT_\mathrm{d})^{-1} \end{bmatrix}. \tag{4}$$

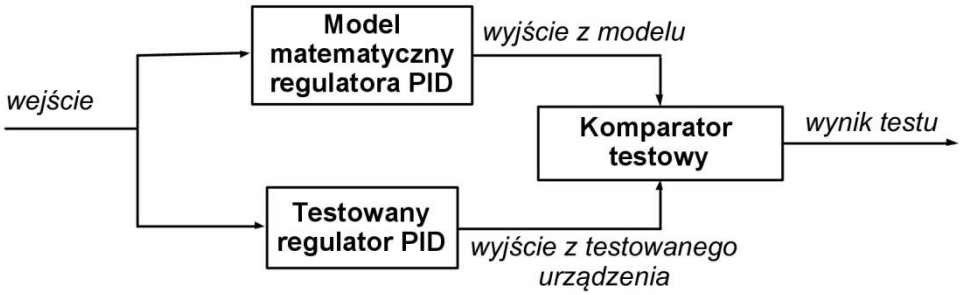
Ograniczenia fizyczne oraz ograniczenia wynikające z zasobów systemowych komputera prowadzą do założenia, że zbiór W musi być ograniczony. Oznacza to, że zbiór W może być zawarty w pewnym okręgu o skończonym promieniu.

3. Koncepcja testowania w oparciu o model systemu jako wzorzec zachowania

Równania (1) i (3) opisują model matematyczny regulatora PID i jednocześnie określają podstawowe wymagania funkcjonalne dla konstruowanego urządzenia. Dalsze czynności inżynierskie polegają na zaprojektowaniu i zbudowaniu fizycznego urządzania, które będzie implementować funkcję regulacji określoną przez równania (1) i (3). W kolejnej fazie, zbudowany układ jest poddawany testowaniu w celu wykrycia błędów w systemie. Proces testowania można przeprowadzić w oparciu o model systemu (1) lub (3), który stanowi w tym wypadku wzorzec zachowania.

Koncepcja testowania przedstawiona na rys. 2 wykorzystuje model w mechanizmie wyznaczania przewidywanych wyników (‘*wyjście z modelu’*) do porównania z wynikami

podawanymi przez testowany system (‘*wyjście z testowanego urządzenia’*) w reakcji na jednakowe wymuszenie (‘*wejście’*) [1]. Sygnał wymuszający podawany na model jest sygnałem wirtualnym, a sygnał podawany na testowany system jest sygnałem fizycznym (np. napięciowym, prądowym, rezystancyjnym) – logicznie oba te sygnały są równoważne. Porównywanie wyników jest realizowane przez urządzenie zwane komparatorem testowym. Rezultat tego porównania reprezentuje funkcja, która przyjmuje wartości binarne, tj. „*wynik testu* = 0” (tzw. test pozytywny) jeżeli wyjście z testowego urządzenia pokrywa się z wyjściem z modelu w zakresie przyjętej tolerancji, i „*wynik testu* = 1” (tzw. test negatywny) w przeciwnym przypadku.



Rys. 2. Koncepcja testowania oparta o model systemu jako wzorzec zachowania

4. Notacja testów

Głównym elementem w procesie testowania jest tworzenie tak zwanych przypadków testowych. Przypadek testowy to zbiór wejść, warunków wykonania oraz oczekiwanych wyników utworzony aby zweryfikować określone wymagania [13].

Dla wymagań określonych przez model (1) przypadek testowy można opisać wykorzystując następującą notację matematyczną

$$\mathrm{T}_{\mathrm{case}}^{(j)} = \left\{T^{(j)}, e^{(j)}(\cdot), u^{(j)}(\cdot)\right\}, \tag{5}$$

gdzie $\mathrm{T}_{\mathrm{case}}^{(j)}$, $j = 1, 2, \dots, N$, $N \geq 1$ jest oznaczeniem przypadku testowego, $e^{(j)}: [0, T^{(j)}] \to \mathbb{R}$ oznacza funkcję wejścia, $u^{(j)}: [0, T^{(j)}] \to \mathbb{R}$ oznacza oczekiwaną funkcję wyjścia, $T^{(j)}$ jest czasem, w którym wykonuje się dany przypadek testowy. Notacja (5) oraz model (1) będą później, w rozdziale 8 pracy, wykorzystane w metodzie wyboru przypadków testowych opartej na testowaniu negatywnym.

Dla wymagań określonych przez model (3) przypadek testowy można opisać za pomocą następującej notacji

$$\mathrm{T}_{\mathrm{case}}^{(j)} = \left\{T^{(j)}, u^{(j)}(\cdot), \boldsymbol{w}^{(j)}(\cdot)\right\}, \tag{6}$$

gdzie $u^{(j)}: [0, T^{(j)}] \to \mathbb{R}$ oznacza funkcję wejścia w odróżnieniu od notacji (5), a $\boldsymbol{w}^{(j)}: [0, T^{(j)}] \to \mathbb{R}^2$ jest oczekiwaną funkcją wyjścia. Notacja (6) oraz model (3) będą wykorzystane w rozdziale 7 w metodzie wyboru przypadków testowych opartej na testowaniu zgodności.

Zbiór składający się z jednego lub więcej przypadków testowych będziemy oznaczać jako $\mathrm{T}_{\mathrm{suite}}$, przy czym

$$\mathrm{T}_{\mathrm{suite}} = \left\{\mathrm{T}_{\mathrm{case}}^{(1)}, \mathrm{T}_{\mathrm{case}}^{(2)}, \dots, \mathrm{T}_{\mathrm{case}}^{(N)}\right\}. \tag{7}$$

5. Komparator testowy

Komparator testowy to narzędzie, które dla danego przypadku testowego porównuje rzeczywiste rezultaty wygenerowane przez testowane urządzenie z oczekiwanymi rezultatami [14]. Oczekiwane rezultaty w koncepcji przedstawionej na rys. 2 są uzyskiwane z modelu systemu określonego przez równania (1) lub (3).

Przykładowa realizacja komparatora testowego dla wymagań określonych przez model (1) oraz dla notacji (5) może wyglądać następująco

$$z\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right)=\begin{cases}0 & \text{gdy } \forall_{t\in[0,T^{(j)}]}\left|u^{(j)}(t)-u_{\mathrm{s}}^{(j)}(t)\right|<\varepsilon\left|u^{(j)}(t)\right|,\\ 1 & \text{w przeciwnym przypadku,}\end{cases} \tag{8}$$

gdzie z oznacza wynik testu, tj. $z=0$ dla testu pozytywnego, $z=1$ dla testu negatywnego, $\varepsilon>0$ jest przyjętą wartością tolerancji, $u_{\mathrm{s}}^{(j)}(\cdot)$ jest funkcją wyjściową uzyskaną z testowanego urządzenia.

Dla modelu (3) oraz notacji (6) komparator testowy może działać według następującego schematu

$$z\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right)=\begin{cases}0 & \text{gdy } \forall_{t\in[0,T^{(j)}]}\left\|\boldsymbol{w}^{(j)}(t)-\boldsymbol{w}_{\mathrm{s}}^{(j)}(t)\right\|<\varepsilon\left\|\boldsymbol{w}^{(j)}(t)\right\|,\\ 1 & \text{w przeciwnym przypadku,}\end{cases} \tag{9}$$

przy czym $\boldsymbol{w}_{\mathrm{s}}^{(j)}(\cdot)$ jest funkcją wyjściową wygenerowaną przez testowane urządzenie.

6. Pokrycie testowe

Niewątpliwie bardzo ważnym elementem w procesie testowania jest określenie tak zwanego pokrycia strukturalnego, czyli miary wskazującej jaki odsetek wybranych elementów jest „pokryty” (wykonany, osiągnięty) podczas wykonywania zestawu przypadków testowych [13]. Określenie dobrych technik pomiaru pokrycia strukturalnego w przypadku układów dynamicznych, a do takich należy niewątpliwie regulator PID, jest zadaniem niełatwym. Naturalnym wyborem wydaje się pomiar pokrycia stanów sytemu. W przypadku systemów dynamicznych przestrzeń stanów jest jednak zbiorem o nieskończonej liczbie elementów i zbudowanie przypadków testowych, które przechodzą przez wszystkie stany systemu może okazać się niemożliwe do realizacji.

W dalszej części pracy zostanie opisany sposób wyliczania pokrycia testowego dla ciągłych systemów dynamicznych, który został zaczerpnięty z pracy [25]. Opisana tam metoda odnosi się do modelu (3). Pokrycie testowe $C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}})$ dla danego zestawu przypadków testowych $\mathrm{T}_{\mathrm{suite}}$ definiuje się w następujący sposób

$$C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}})=\frac{\left|\bigcup_{j=1}^{j=N} V_{\boldsymbol{h}}\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right)\right|}{|W_{\boldsymbol{h}}|}, \tag{10}$$

gdzie

$$W_{\boldsymbol{h}}=\{\boldsymbol{i}=[i_1,\ i_2]^{\mathrm{T}}\in\mathbb{Z}^2\colon\ \exists_{\boldsymbol{w}\in W}\boldsymbol{w}\in G_{\boldsymbol{h}}(\boldsymbol{i})\} \tag{11}$$

jest zbiorem uzyskanym z przestrzeni stanów W poprzez jej podział na prostokątne elementy

$$G_{\boldsymbol{h}}(\boldsymbol{i})=\left\{\boldsymbol{w}\in\mathbb{R}^2\colon \boldsymbol{w}=[w_1,\ w_2]^{\mathrm{T}},\ \left\lfloor\frac{w_k}{h_k}\right\rfloor=i_k,\ k=1,2\right\} \tag{12}$$

o wielkości określonej przez wektor $\boldsymbol{h} = [h_1,\ h_2]^{\mathrm{T}}$, $h_1, h_2 > 0$. $\left\lfloor \frac{w_k}{h_k} \right\rfloor$ we wzorze (12) oznacza największą liczbę całkowitą nie większą niż $\frac{w_k}{h_k}$. Wówczas zbiór określony jako

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

będzie podzbiorem $W_{\boldsymbol{h}}$ zawierającym elementy, które są pokrywane przez dany przypadek testowy $\mathrm{T}_{\mathrm{case}}^{(j)}$. Podobnie, zbiór

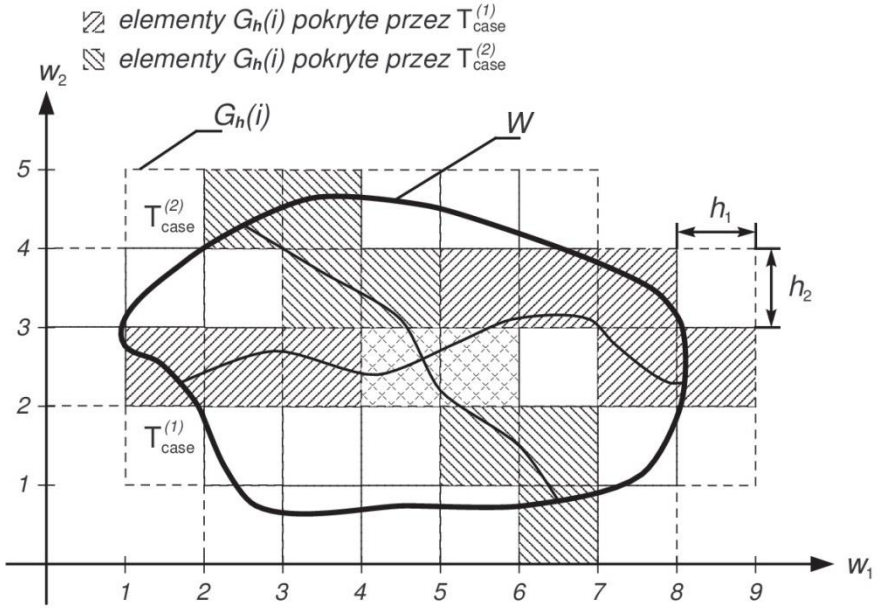
$$V_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) = \cup_{j=1}^{j=N} V_{\boldsymbol{h}}\left(\mathrm{T}_{\mathrm{case}}^{(j)}\right) \tag{14}$$

będzie obejmować te elementy ze zbioru $W_{\boldsymbol{h}}$, które są pokrywane przez cały zestaw przypadków testowych $\mathrm{T}_{\mathrm{suite}}$.

Rys. 3 zawiera graficzną ilustrację obliczania pokrycia testowego za pomocą zaprezentowanej metody. Przestrzeń stanu W (obszar zawarty wewnątrz zamkniętej krzywej zaznaczonej na rysunku pogrubioną linią) został podzielony na 45 prostokątnych elementów $G_{\boldsymbol{h}}(\boldsymbol{i}) = [i_1, i_1 + 1) \times [i_2, i_2 + 1)$ o wymiarach $1\ \times 1$

$$W_{\boldsymbol{h}} = \{\boldsymbol{i} = [i_1,\ i_2]^{\mathrm{T}} \in \mathbb{Z}^2,\ i_1 = 0,\ 1, \dots, 8,\ i_2 = 0, 1, \dots, 4\}\backslash\{[0,\ 0]^{\mathrm{T}},\ [1,\ 0]^{\mathrm{T}},\ [8,\ 0]^{\mathrm{T}},\ [0,\ 1]^{\mathrm{T}},\ [0,\ 4]^{\mathrm{T}},\ [7,\ 5]^{\mathrm{T}},\ [8,\ 4]^{\mathrm{T}}\}. \tag{15}$$

Rysunek zawiera dodatkowo wykres trajektorii $\boldsymbol{w}$ przynależących do dwóch przykładowych przypadków testowych $\mathrm{T}_{\mathrm{case}}^{(1)}$ i $\mathrm{T}_{\mathrm{case}}^{(2)}$. Z wykresu łatwo można odczytać, że pierwszy przypadek testowy pokrywa 10 elementów ze zbioru $W_{\boldsymbol{h}}$, zaś drugi przypadek testowy – 9 elementów. Zestaw przypadków testowych $\mathrm{T}_{\mathrm{suite}} = \left\{\mathrm{T}_{\mathrm{case}}^{(1)}, \mathrm{T}_{\mathrm{case}}^{(2)}\right\}$ pokrywa w sumie 17 elementów ze zbioru $W_{\boldsymbol{h}}$ co implikuje pokrycie testowe $C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) = \frac{17}{45} \approx 0.45\ (45\ \%)$.



Rys. 3. Ilustracja koncepcji obliczania pokrycia testowego dla przestrzeni stanów $\boldsymbol{W}$

7. Metoda wyboru przypadków testowych oparta na testowaniu zgodności

W tym rozdziale zostanie zaprezentowany algorytm wyboru przypadków testowych oparty na tak zwanym testowaniu zgodności. Testowanie zgodności [14] (w literaturze można również spotkać określenia testowanie poprawności [16] czy też testowanie funkcjonalne [13]) to podejście polegające na weryfikacji zgodności implementacji z odpowiednią specyfikacją (w tym przypadku z modelem matematycznym). Przedstawiony algorytm wykorzystuje model systemu w postaci (3) oraz pokrycie testowe liczone według schematu z rozdziału 6.

Algorytm 1

1°: Określ granulację $\boldsymbol{h} = [h_1,\ h_2]^{\mathrm{T}},\ h_1, h_2 > 0,$ wymaganą wartość pokrycia testowego $\delta \in (0, 1]$ oraz czas wykonywania testów $T > 0$.
2°: Podstaw $\mathrm{T}_{\mathrm{suite}} := \emptyset,\ V_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) := \emptyset,\ C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) := 0,\ j := 1$.
3°: Jeśli $C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}}) < \delta$ przejdź do kroku 4°, w przeciwnym przypadku zakończ obliczenia.
4°: Znajdź $\boldsymbol{w}_a \in G_{\boldsymbol{h}}(\boldsymbol{i}_a)$ gdzie $\boldsymbol{i}_a \in W_{\boldsymbol{h}} \backslash V_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}})$.
5°: Określ funkcję wejścia $u(\cdot)$ przeprowadzającą system z zerowego punktu początkowego do punktu $\boldsymbol{w}_a$.
6°: Oblicz trajektorię systemu $\boldsymbol{w}(t) = \int_0^T e^{(t-\tau)\boldsymbol{E}}\boldsymbol{F}u(\tau)\mathrm{d}\tau$ poprzez rozwiązanie równania (3).
7°: Podstaw $T^{(j)} := T,\ u^{(j)}(\cdot) := u(\cdot),\ \boldsymbol{w}^{(j)}(\cdot) := \boldsymbol{w}(\cdot)$ i utwórz przypadek testowy $\mathrm{T}_{\mathrm{case}}^{(j)} = \left\{T^{(j)}, u^{(j)}(\cdot), \boldsymbol{w}^{(j)}(\cdot)\right\}$.
8°: Podstaw $\mathrm{T}_{\mathrm{suite}} := \mathrm{T}_{\mathrm{suite}} \cup \mathrm{T}_{\mathrm{case}}^{(j)}$.
9°: Oblicz $V_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}})$ oraz $C_{\boldsymbol{h}}(\mathrm{T}_{\mathrm{suite}})$.
10°: Podstaw $j := j + 1$ i przejdź do kroku 3°.

Uwaga 1. Wielkość granulacji $\boldsymbol{h} = [h_1,\ h_2]^{\mathrm{T}}$ można określić za pomocą następującej formuły

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

W przypadku systemów krytycznych ze względu na bezpieczeństwo można rozważać mniejsze wartości h_i. Należy jednak pamiętać, że zmniejszenie granulacji podziału przestrzeni stanu W będzie wiązało się z większym nakładem obliczeniowym.

Uwaga 2. System określony przez równanie (3) jest sterowalny gdyż $\mathrm{rank}\,[\boldsymbol{E}\ \boldsymbol{EF}] = 2$ (zob. warunek sterowalności np. w pracy [20]). Oznacza to, że istnieje sterowanie, które przeprowadzi system w skończonym czasie T z zerowego punktu początkowego do dowolnego punktu $\boldsymbol{w}_a$. Jednym z takich sterowań jest sterowanie minimalno-energetyczne [zob. 20], które można wyliczyć w sposób analityczny.

8. Metoda wyboru przypadków testowych oparta na testowaniu negatywnym

Testowanie negatywne to takie, którego celem jest pokazanie, że system nie działa. W tym rozdziale zagadnienie testowania negatywnego zostanie sformułowane jako problem optymalizacyjny. Przypadki testowe będą konstruowane w oparciu o model (1) podczas procedury optymalizacyjnej o charakterze iteracyjnym. Takie podejście pozwali na znaczne

zredukowanie ilości wymaganych testów oraz pozwoli skoncentrować się na sytuacjach, które mogą prowadzić do awarii całego systemu.

Zagadnienie testowania negatywnego można sformułować jako poszukiwania przypadku testowego $\mathrm{T_{case}} = \{T, e(\cdot), u(\cdot)\}$ będącego wynikiem następującego problemu optymalizacyjnego

$$\max_{e\in E_{\mathrm{ad}}} J(e) = \max_{e\in E_{\mathrm{ad}}} \int_0^T \big(u(t) - u_{\mathrm{s}}(t)\big)^2 \,\mathrm{d}t\,, \tag{17}$$

gdzie E_{ad} oznacza zbiór rozwiązań dopuszczalnych. Zbiór E_{ad} może być określony przez ograniczenia fizyczne i ograniczone nałożone przez zasoby systemowe komputera.

Algorytm 2

1°: Określ zbiór rozwiązań dopuszczalnych E_{ad} oraz czas wykonywania testów $T > 0$.
2°: Rozwiąż problem optymalizacyjny (16) w celu uzyskania rozwiązania optymalnego $e^* \in E_{\mathrm{ad}}$ w sensie zdefiniowanego wskaźnika jakości.
3°: Oblicz oczekiwaną funkcję wyjściową $u^*(\cdot)$ za pomocą równania (1) dla wejścia $e^*(\cdot)$.
4°: Utwórz przypadek testowy $\mathrm{T_{case}} = \{T, e^*(\cdot), u^*(\cdot)\}$.

9. Eksperymenty badawcze

Eksperymenty badawcze polegały na sprawdzeniu czy przedstawione w pracy algorytmy są w stanie wykryć usterki w rzeczywistym systemie. Usterki te w postaci nieprawidłowych wartości parametrów regulatora PID zostały celowo wprowadzone do implementacji regulatora na platformie mikroprocesorowej. W celu lepszej ilustracji uzyskanych wyników wprowadzono wartości parametrów różniących się od właściwych nastaw o 20%. Błędy te mogą być skutkiem zastosowania arytmetyki stałoprzecinkowej, mogą one też wynikać z błędów podczas procedury identyfikacyjnej jak być bezpośrednim następstwem błędu programisty. Wprowadzenie niewłaściwych wartości parametrów do układu regulacji może skutkować tym, że system nie będzie dochodzić do stanu ustalonego w zakładanym czasie, w układzie wystąpią większe przeregulowania, a w najbardziej niekorzystnym przypadku system zamknięty nie będzie stabilny. Dobrą jakość regulacji osiąga się bowiem dopiero przy właściwym doborze nastaw co jest szczególnie istotne w zagadnieniach sterowania optymalnego [7] mających zastosowanie np. w silnikach elektrycznych [8] i spalinowych [26].

Rozważmy model regulatora PID określony przez równanie (1) z następującymi parametrami

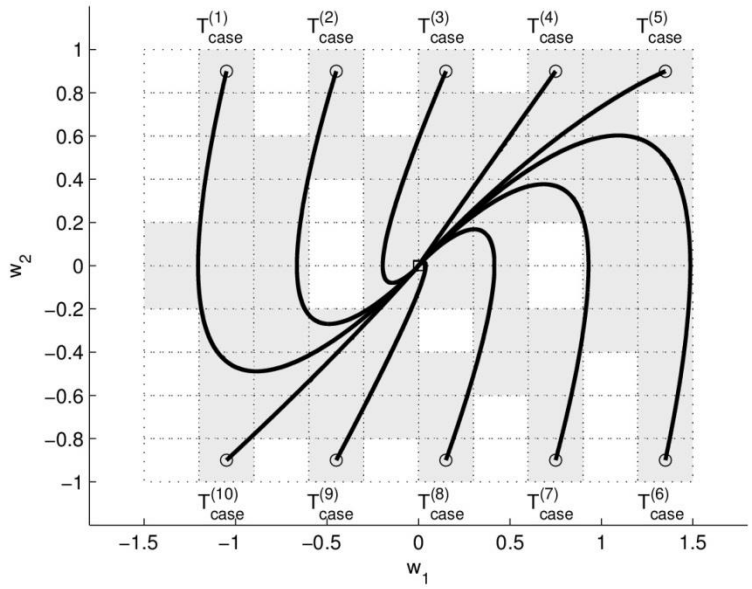
$$K = 3.60,\ \ T_{\mathrm{i}} = 1.81,\ \ T_{\mathrm{d}} = 0.45\,. \tag{18}$$

W oparciu o ten model budujemy mikroprocesorowy regulator PID implementujący funkcję regulacji daną wzorem (1), ale z błędnymi parametrami, to znaczy

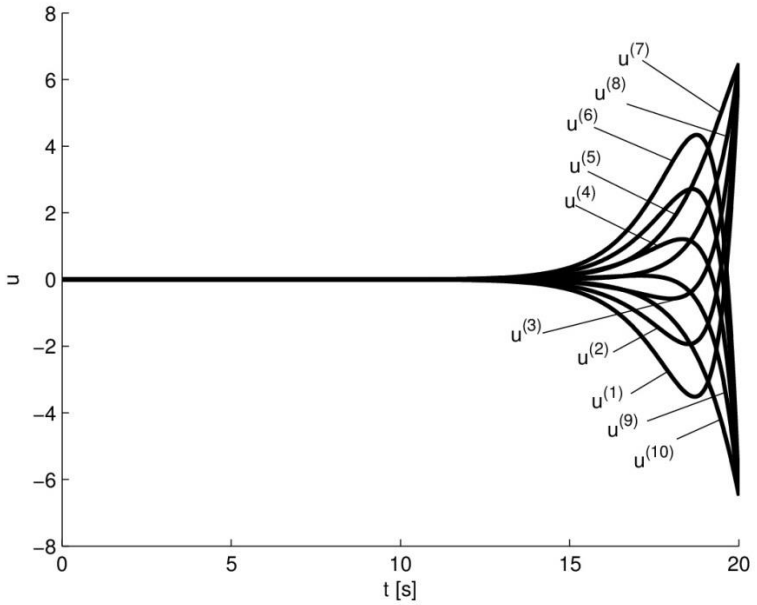
$$\widetilde{K} = 2.88,\ \ \widetilde{T_{\mathrm{i}}} = 2.17,\ \ \widetilde{T_{\mathrm{d}}} = 0.36\,. \tag{19}$$

Następnie, korzystamy z algorytmu 1 przyjmując następujące założenia: $\boldsymbol{h} = [0.3,\ 0.2]^{\mathrm{T}}$, $\delta = 0.7,\ \ T = 20$ [s], $|w_1(t)| \le 1.5,\ \ |w_2(t)| \le 1.0$. W rezultacie dostajemy zestaw 10

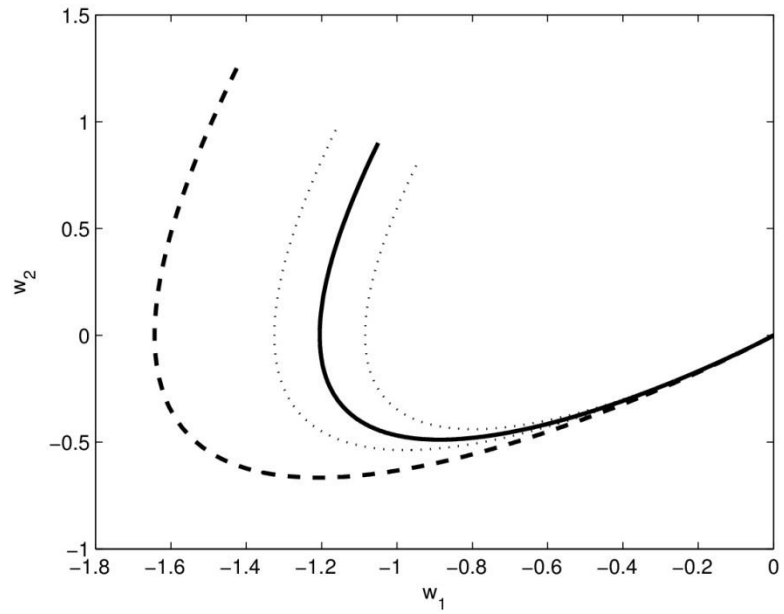
przypadków testowych, który zapewnia pokrycie testowe na poziomie co najmniej 0.7. Elementy składowe wygenerowanych przypadków testowych w postaci (6) są graficznie zilustrowane na rys. 4 i 5. Porównanie rzeczywistej trajektorii uzyskanej z testowanego urządzenia z oczekiwaną trajektorią jest przedstawione na rys. 6. Dla przyjętej wartości tolerancji $\varepsilon = 0.1$ wynik z wykonania pierwszego przypadku testowego jest negatywny, co potwierdza istnienie błędu w systemie.



Rys. 4. Trajektorie systemu $\boldsymbol{w}^{(j)}$, $j = 1, 2, \ldots, 10$ oraz elementy (szare prostokąty) zbioru $W_{\boldsymbol{h}}$ pokryte przez przypadki testowe $\mathrm{T}_{\mathrm{case}}^{(j)}$. Trajektorie startują w punktach oznaczonych przez □ i kończą się w ○



Rys. 5. Funkcje wejściowe dla przypadków testowych $\mathrm{T}_{\mathrm{case}}^{(j)}$, $j = 1, 2, \ldots, 10$



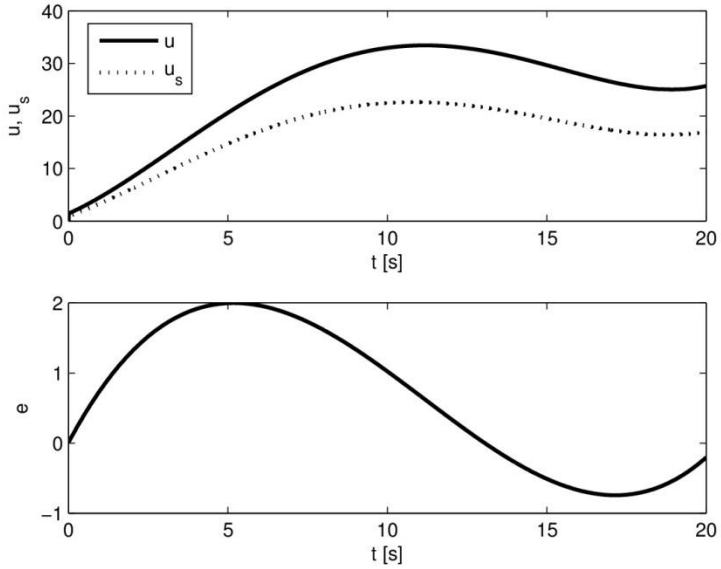
Rys. 6. Porównanie rzeczywistej trajektorii $\boldsymbol{w}_s^{(1)}$ uzyskanej z testowanego regulatora PID (linia przerywana) z oczekiwaną trajektorią $\boldsymbol{w}^{(1)}$ (linia ciągła). Granice 10% tolerancji są zaznaczone na rysunku linią kropkowaną.

Procedura optymalizacyjna dla algorytmu 2 została przeprowadzona na następującym zbiorze dopuszczalnych funkcji wejściowych

$$E_{\mathrm{ad}} = \{e \in \mathrm{PC}([0,T],\mathbb{R}):\ e(t) = \alpha_0 t^3 + \alpha_1 t^2 + \alpha_2 t + \alpha_3,\ \alpha_i \epsilon \mathbb{R},\ |e(t)| \leq 2,\ i = 0,1,2,3\}. \tag{20}$$

Wyznaczanie ekstremum dla zdefiniowanego wskaźnika jakości (zob. wzór (17)) zostało przeprowadzone w oparciu o sympleksową metodę spadku określaną metodą Neldera-Meada [23], w efekcie której otrzymano następujące rozwiązanie

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



Rys. 7. Elementy składowe przypadku testowego wygenerowanego w oparciu o algorytm 2

Elementy składowe wygenerowanego przypadku testowego w postaci (5) są graficznie zilustrowane na rys. 7. Rysunek ten zawiera również dla porównania rzeczywistą trajektorię uzyskaną z testowanego urządzenia.

Główną zaletą metody testowania negatywnego opartej o algorytm 2 jest znaczna redukcja przypadków testowych, których poszukiwanie odbywa się z wykorzystaniem procedury optymalizacyjnej. Algorytm skupia się tylko na takich przebiegach wejściowych do układu regulacji, które prowadzą do błędnych zdarzeń w systemie. W konsekwencji można w znacznym stopniu ograniczyć czas i koszty związane z testowaniem układu. Ponieważ nakład pracy związany z testowaniem i weryfikacją układu to według szacunków [2] od 30 do 90% całościowego nakładu pracy w projekcie, korzyści płynące ze zmniejszeniem tego czynnika nawet w stopniu minimalnym mogą być bardzo opłacalne. Należy także podkreślić, że algorytm 2 przeprowadza poszukiwanie przypadków testowych przy jednoczesnym wykorzystaniu rzeczywistego układu i jego matematycznej reprezentacji. Zatem aby rozpocząć proces testowania potrzebny jest zarówno model i układ rzeczywisty. Trudności w implementacji mogą również wynikać z faktu, że w procedurze optymalizacyjnej poszukujemy najlepszego rozwiązania w zbiorze funkcji, a nie w zbiorze wartości.

10. Podsumowanie

W pracy zostały zaprezentowane różne metody, za pomocą których można weryfikować mikroprocesorowe regulatory PID w celu zapewniania wymaganej jakości systemu, zgodności z normami bezpieczeństwa, a przede wszystkich w celu wyeliminowania błędów jeszcze na etapie projektowania systemu. Eliminacja błędów we wczesnych fazach powstawania produktu pozwala zwiększyć niezawodność systemu i zmniejszyć ryzyko powstania awarii na etapie eksploatacji. Wszystkie elementy procesu testowania (tj. koncepcja, notacja testów, pokrycie testowe, określanie wyniku testu, wybór przypadków testowych) zostały sformułowane i opisane w pracy za pomocą odpowiedniej notacji matematycznej. Kluczową rolę w zaprezentowanej koncepcji odgrywa model matematyczny systemu, który jest traktowany jako wzorzec zachowania. W ten sposób możliwe było opracowanie metod testowania dla systemów, w których dynamika odgrywa istotną rolę i do których nie można zastosować klasycznych technik testowania.

Przedstawione w pracy matematyczne metody weryfikacji regulatorów PID można łatwo uogólnić na inne przypadki mikroprocesorowych układów regulacji. Regulatory z kompensatorem dynamicznym [27], regulatory silników elektrycznych [7, 8] i spalinowych, specjalizowane układy regulacji zbudowane w oparciu o sieci neuronowe [9] i logikę rozmytą [10] mogą być weryfikowane i testowane przy wykorzystaniu opisanych algorytmów.

Literatura

1. Adrion W, Brandstad J, Cherniabsky J. Validation, verification and testing of computer software. Computing Survey 1982; 14(2): 159-192.

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. Ocena wpływu stanów dynamicznych silnika spalinowego na jego własności użytkowe. Eksploatacja i Niezawodnosc – Maintenance and Reliability 2015; 17(1): 35-41.

5. Dang T. Model-based testing of hybrid systems. In: Model-Based Testing for Embedded Systems. Boca Raton: CRC Press 2011; 383-423.

6. Dang T, Nahhal T. Coverage-guided test generation for continuous and hybrid systems. Formal Methods in System Design 2009; 34(2): 183-213.

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.

10. Długosz R, Pedrycz W. Łukasiewicz fuzzy logic networks and their ultra low power hardware implementation. Neurocomputing 2010; 73(7-9): 1222-1234.

11. Esposito J. Automated test trajectory for hybrid systems. Proceedings of the 35th Southeastern Symposium on System Theory 2003; 441-444.

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.

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.

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.

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.

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.

26. Skruch P. An educational tool for teaching vehicle electronic system architecture. International Journal of Electrical Engineering Education 2011; 48(2): 174-183.

27. Skruch P: Feedback stabilization of a class of nonlinear second-order systems. Nonlinear Dynamics 2010; 59(4): 681-692.

28. Tabuada P. Verification and Control of Hybrid Systems. Dordrech: Springer, 2009.

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.