Realizacja programowa algorytmów filtracji, estymacji i sterowania w PLC/PAC

Jarosław Tarnawski

Katedra Inżynierii Systemów Sterowania, Wydział Elektrotechniki i Automatyki, Politechnika Gdańska

Streszczenie: Sterowniki programowalne PLC (ang. *Programmable Logic Controller*) są główną przemysłową platformą implementacji algorytmów sterowania bezpośredniego. Standardowo producenci PLC dostarczają programistom jedynie podstawowe, najprostsze metody sterowania. Wraz z rozwojem sterowników PLC oraz ich następców PAC (ang. *Programmable Automation Controller*) pojawiły się zwiększone możliwości obliczeniowe i pamięciowe tych urządzeń oraz pełniejsza implementacja języków programowania określonych w normie IEC-61131-3. PLC i PAC mają obecnie moc obliczeniową i dostępną pamięć odpowiadającą komputerom osobistym PC sprzed kilku lat, można je programować również w językach wysokiego poziomu stosując zmienne zdefiniowane w postaci macierzowej. Uwzględniając pewne ograniczenia i specyfikę działania PLC oraz PAC można w tych urządzeniach zaimplementować wiele zdyskretyzowanych algorytmów sterowania, estymacji czy filtracji. Pomimo niewątpliwych potencjalnych korzyści wynikających ze stosowania zaawansowanych metod w warstwie sterowania bezpośredniego, temat ten w literaturze jest skromnie reprezentowany. W artykule prezentowane jest podejście do programowania algorytmów filtracji, estymacji i sterowania, opisanych równaniami różnicowymi. Przedstawiono metodykę budowy oprogramowania dla PLC/PAC. Dla zilustrowania procesu implementacji algorytmu z pogranicza filtracji i estymacji wykorzystano metodę najmniejszych kwadratów ze współczynnikiem zapominania RLS_FF.

Słowa kluczowe: sterowniki programowalne PLC/PAC, dyskretne algorytmy sterowania, strojenie i weryfikacja oprogramowania, estymacja RLS_FF

Sterowniki programowalne powstały w celu zastąpienia układów sterowania przełączającego, realizowanych za pomocą styczników i przekaźników [6]. Pierwotnie PLC obsługiwały wyłącznie sygnały dyskretne, następnie zostały wyposażone w przekaźniki czasowe i zegary umożliwiające włączanie i wyłączanie urządzeń nie tylko w zależności od stanu sygnałów pomiarowych, ale również z opóźnieniem czy o określonym czasie. Równolegle rozwijane były systemy klasy DCS zorientowane od początku na realizację sterowania ciągłego: algorytmy PID, MPC, sterowania rozmytego i wielu innych. Systemy klasy DCS były lokowane głównie w przemyśle chemicznym, energetyce, rafineryjnym itp. Ze względu na elastyczność, dużą konkurencję na rynku i pozycjonowanie przez producentów PLC można znaleźć w instalacjach przemysłowych różnej skali od najmniejszych do bardzo dużych. Wraz z rozwojem techniki mikroprocesorowej i komputerowej PLC zostały wyposażone w podstawowy aparat matematyczny, początkowo bazujący na liczbach całkowitych, a następnie zmiennoprzecinkowych. Pierwotnie pamięć PLC była widziana jako struktura wektorowa, obecnie może być również organizowana w postaci macierzy. Współczesne możliwości sprzętowe i programowe udostępnione przez producentów programistom PLC zbliżone są do tych dostępnych w komputerach klasy PC. Producenci, aby zaakcentować rozwój i zwiększone możliwości PLC, nową generację tych urządzeń nazwali PAC [5]. Różnice między systemami PLC/PAC (wraz z systemem SCADA) oraz DCS znacznie się zmniejszyły. Platforma PLC jest standardowo dość oszczędnie wyposażona w predefiniowane bloki i algorytmy stosowane w automatyce, ale predestynowana do zrealizowania programowego, również zaawansowanych algorytmów filtracji, estymacji i algorytmów sterowania, co przedstawiono w artykule dla urządzeń firmy GE Fanuc kontrolerów PAC – RX3i. Zaprezentowane metody i opracowany kod mają charakter przenośny, a po niewielkim dostosowaniu mogą zostać zastosowane w urządzeniach PLC/PAC innych firm.

Przez zaawansowane algorytmy sterowania estymacji i filtracji dla PLC określa się na potrzeby tego artykułu te metody, które są standardowo niedostępne na platformie PLC i są bardziej skomplikowane w realizacji niż regulatory dwu- i trójpołożeniowe, regulatory regułowe IF-THEN oraz regulatory PID. Dla łatwiejszej orientacji można wymienić wielowymiarowe sterowanie od stanu, metody identyfikacji, filtracja, estymacja parametryczna, liniowe MPC, adaptacyjne MRAC i ST. W dostępnej literaturze trudno odnaleźć przykłady bardziej rozbudowane niż regulatory PID, ewentualnie regulatory sklejane za pomocą logiki rozmytej.

1. Sposób działania PLC wymusza metodę programowania

Sterowniki programowalne to przemysłowe urządzenia sterujące zaprojektowane i zbudowane tak, aby spełniać postulat pracy w czasie rzeczywistym. Oznacza to, że programiści i użytkownicy tych urządzeń mają gwarancję, iż w skończonym, znanym *a priori* czasie informacja docierająca na wejścia PLC zostanie wczytana, przetworzona, a następnie zostanie wygenerowana odpowiedź wystawiona na wyjścia urządzenia. Pracę w reżimie spełniającym twarde wymagania czasu rzeczywistego zrealizowano za pomocą nieskończonej pętli działającej na poziomie systemu operacyjnego, nazywanej cyklem pracy sterownika. Można wskazać wspólne cechy dla wielu rządzeń tej klasy. Powszechnie stosowane etapy/fazy cyklu to: inicjalizacja cyklu, wczytanie wejść fizycznych do pamięci, wykonanie programu sterującego, przeniesienie stanu pamięci na wyjścia, diagnostyka. W skład cyklu przeważnie wchodzi również etap komunikacji z programatorem umożliwiającym wczytanie programu do PLC lub wymuszenie odpowiedniego trybu pracy tego urządzenia. Podgląd pamięci PLC może być również realizowany w tym trybie lub asynchronicznie – niezależnie od cyklu. Producenci dostarczają bardzo precyzyjnych informacji o czasie trwania poszczególnych faz cyklu. Czas ten zależy od liczy podłączonych modułów wejść/wyjść, długości i złożoności programu i wielu innych czynników. Obliczanie czasu trwania cyklu ma jednak charakter deterministyczny i może być przeprowadzone z dokładnością do milisekund, a niektóre składniki cyklu można określić z dokładnością do mikrosekund.

2. Nadzór nad czasem trwania cyklu

Ze względu na realizację postulatu pracy w czasie rzeczywistym producenci PLC wprowadzili ograniczenia od góry na maksymalny czas trwania cyklu. Rozwiązanie to jest podobne do stosowanych w mikrokontorlerach mechanizmów typu *watchdog* powodujących wykrycie przekroczenia maksymalnego dopuszczalnego czasu trwania cyklu. Przykładowo w PAC RX3i GE Fanuc (CPU310) maksymalny dopuszczalny czas trwania cyklu to 2550 ms. Przekroczenie maksymalnego czasu trwania cyklu może być spowodowane różnymi przyczynami, np. błędem logicznym w programie sterującym. Najczęściej są to pętle nieskończone spowodowane skokami do wcześniejszej części programu, niewłaściwie zapętlone podprogramy lub złe sformułowanie warunków na opuszczenie pętli. Zabezpieczenie *watchdog* razem z fazą autodiagnostyki obejmuje również wszelkiego rodzaju przekłamania w odczycie z pamięci programu i ogólnie nazywaną utratą integralności programu logicznego i systemu operacyjnego PLC. W wyniku przekroczenia dopuszczalnego maksymalnego czasu trwania cyklu (nie mylić z przekroczeniem czasu ustawionego jako stały czas trwania cyklu) sterownik przechodzi w tryb STOP, nie wykonuje programu sterującego i wymaga interwencji programisty/operatora. Opisane zabezpieczenie opracowano, aby zapewniać skończony czas reakcji systemu na bodźce wejściowe oraz zabezpieczenie przed błędami logicznymi i utratą integralności systemu, jednak ma ono także konsekwencje w postaci ograniczenia czasu przeznaczonego na realizację programu sterującego. Opisując to wprost, wszystkie wykonywane obliczenia oraz pozostałe etapy cyklu muszą trwać krócej niż maksymalny czas trwania cyklu. Nie ma to znaczenia dla prostych i średnio złożonych programów, ale przy realizacji programowej algorytmów wymagających dużej liczby obliczeń staje się to istotnym ograniczeniem. Przy bardzo złożonych algorytmach obliczeniowych istnieje możliwość podziału algorytmu na części i wykonania tych części w osobnych cyklach bez przekraczania maksymalnego czasu trwania pojedynczego cyklu. Jednak wtedy krok wykonania algorytmu nie jest realizowany w każdym cyklu, a po wykonaniu wszystkich etapów algorytmu, czyli co kilka cykli.

3. Wybór języka programowania

W normie IEC 61131-3 dotyczącej programowania sterowników PLC zdefiniowane są języki programowania graficzne i tekstowe [1]. Do grupy języków graficznych zalicza się język drabinkowy LD (ang. *Ladder Diagram*) oraz język bloków funkcyjnych FBD (ang. *Function Blok Diagram*). Języki graficzne zapewniają dużą przejrzystość kodu dla operacji na stykach i przekaźnikach (realizacji układów przełączających), jednak przy dużej liczbie operacji matematycznych ten zapis jest zbyt rozwlekły i nieczytelny. Języki tekstowe opisane w normie IEC 61131-3 to język listy instrukcji IL (ang. *Instruction Logic*) oraz język tekstu strukturalnego ST (ang. *Structured Text*). Różnica między nimi jest zasadnicza, IL jest językiem niskopoziomowym, a ST – wysokopoziomowym. Język IL jest zbliżony koncepcyjnie do assemblera – zapewnia bardzo dużą wydajność obliczeń jednak realizuje wyłącznie podstawowe operacje na danych (z wykorzystaniem akumulatora i stosu). Jego przydatność jest zatem ograniczona. Najkorzystniejszym językiem, z punku widzenia implementacji zaawansowanych algorytmów, jest język ST przypominający składnią mieszankę języków C, Pascal i Basic, oferujący pętle, skoki warunkowe, instrukcje wyboru oraz indeksowanie macierzy. Niektórzy producenci PLC dostarczają do swoich sterowników kompilatory języka C i ten język również byłby użyteczny, jednak ze względu na różnice w implementacji kompilatorów oraz brak tego języka w normie, trudno mówić o zaletach przenośności i uniwersalności takiego kodu [2]. Rozważania w artykule dotyczą języków IL i ST, jednak należy podkreślić, że producenci rzadko dostarczają oba te języki ze swoimi produktami. Przykładowo, dla PLC GE Fanuc serii 90-30 dostępny jest język IL, a nie ma ST, natomiast dla PAC GE Fanuc RX3i jest odwrotnie. Do nadzorowania przebiegu programu można stosować język SFC (ang. *Sequential Function Chart*) albo język LD z wywołaniem procedur.

4. Forma zapisu zaawansowanych algorytmów sterowania, estymacji filtracji

Zaawansowane algorytmy sterowania obiektami dynamicznymi są przeważnie definiowane równaniami różniczkowymi lub całkowymi. Ta forma nie jest wprost przydatna do implementacji w PLC. W wyniku dyskretyzacji równania różniczkowe można przekształcić w równania różnicowe, które mają postać dogodną do implementowania w urządzeniach cyfrowych, jak mikrokontrolery, komputery PC oraz PLC/PAC [4]. W literaturze dotyczącej dyskretnej teorii sterowania znajdziemy bogatą bazę algorytmów sterowania o różnej skali złożoności opisanych równaniami różnicowymi. Postać równań różnicowych mają dyskretne wersje obserwatorów, algorytmów estymacji parametrów modeli (np. metodą najmniejszych kwadratów, metodą gradientową) oraz algorytmy filtracji, np. dyskretny filtr Kalmana lub dyskretne częstotliwościowe filtry pasmowe. Cechą wspólną tych algorytmów jest ich rekurencyjny charakter, praca ze stałym okresem próbkowania, sposób realizacji za pomocą podstawowych operacji arytmetycznych oraz mechanizm zapamiętywania bieżących wartości sygnałów, i w kolejnych chwilach wykonania algorytmu ich przesuwania w pamięci (reprezentujące „starzenie" sygnału) po to, aby dysponować w kolejnych chwilach opóźnionymi wartościami sygnałów, koniecznymi do realizacji algorytmu.

Sterowniki PLC standardowo, w językach programowania zdefiniowanych w normie IEC-61131, nie obsługują macierzowych operacji algebraicznych (programiści mają do dyspozycji skalarne operacje +, –, /, *). Z punktu widzenia realizacji algorytmu sterowania w PLC istotne jest rozróżnienie skalarnego i macierzowego charakteru równania różnicowego implementowanego algorytmu.

5. Czynności wstępne w projektowaniu systemu sterowania z PLC/PAC

Projektowanie oprogramowania dla PLC wymaga czynności przygotowawczych jeszcze przed opracowaniem koncepcji programu sterującego. Proces ten można podzielić na fazy [3]:

– podstawowa: określenie właściwego kroku pracy algorytmu (krok próbkowania, aktualizacji) na podstawie dynamiki obiektu/procesu, dobór współpracujących urządzeń, określenie niezbędnych wielkości, przypisania im zmiennych, wybór typu zmiennych, oszacowanie wielkości pamięci, nadanie etykiet zmiennym oraz adresom logicznym w pamięci PLC, wybór właściwego PLC dysponującego odpowiednią mocą obliczeniową, układami wejść/wyjść, pamięcią, językami programowania, możliwościami komunikacyjnymi, uwzględniając aspekty ekonomiczne i potencjalne możliwości rozwoju, oszacowanie czasu trwania jednego cyklu programu sterującego, konfiguracja PLC (w tym wymuszenie pracy ze stałym krokiem),
– programowania: strukturyzacja – podział algorytmu i jego programowej realizacja na bloki logiczne (funkcje, procedury), wybór języka i metody programowania dla każdego z bloku logicznego,
– dodatkowe: zaprojektowanie metody diagnostyki programu i weryfikacji implementacji, zaprojektowanie współpracy z użytkownikiem, interfejs HMI, wytypowanie zmiennych do archiwizacji i współpracy z bazą danych, dbałość o czytelność programu – strukturyzacja, czytelność, nazwy zmiennych, opisy poszczególnych fragmentów kodu.

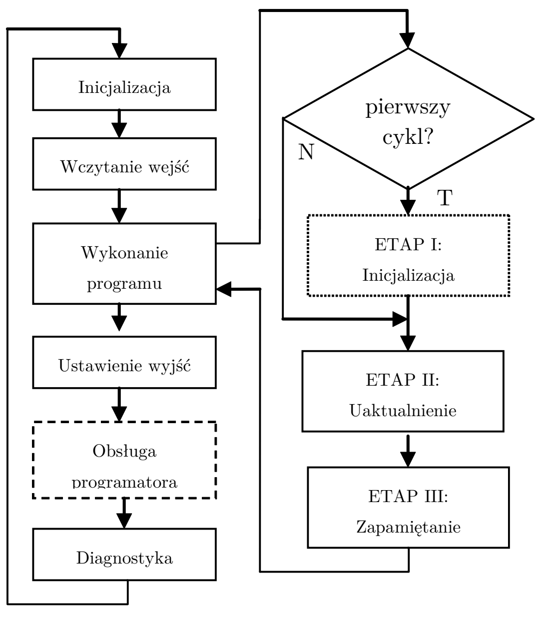
6. Metoda realizacji algorytmów danych przez równania różnicowe w PLC

Uwzględniając specyfikę działania sterowników programowalnych oraz postać algorytmów opisanych za pomocą równań różnicowych można zaproponować ogólną strukturę programu (rys. 1). Składa się ona z mechanizmu nadzoru nad krokiem wykonania algorytmu i trzech etapów: inicjalizacji zmiennych, uaktualniania zmiennych, zapamiętywania zmiennych.

Etap inicjalizacji zmiennych wykonywany jest tylko w pierwszym cyklu pracy sterownika i służy do nadania warunków początkowych równania różnicowego, tj. nadania wartości wszystkim zmiennym wykorzystywanym w programie opisanych jako przeszłe (np. $u(k–1)$, $y(k–3)$, $e(k–5)$). Oprócz zainicjowania zmiennych występujących w opisie równania, np. $e(k–5)$ niezbędne będą wartości kolejne, tj. od $e(k–4)$ do $e(k–1)$, bowiem te wartości w kolejnych krokach czasu będą reprezentowały stosowaną w programie $e(k–5)$. Występuje tu analogia do równań różniczkowych, gdzie komplet warunków początkowych dla równania n-tego stopnia stanowi zestaw pochodnych od pierwszej do $(n–1)$-ej w chwili $t = 0$.

W drugim etapie wykonania programu następuje uaktualnienie zmiennych zrealizowane przez operacje arytmetyczne definiujące algorytm. Najczęściej są to proste operacje dodawania, odejmowania, mnożenia i dzielenia. W przypadkach algorytmów opisanych skalarnie etap ten jest prosty do realizacji i wykonywany bardzo szybko. Jeżeli algorytmy dane są w postaci macierzowej – czas jego trwania i realizacji zależy od charakteru tych macierzy i ich wymiaru. W artykule zostanie osobno przedstawiony wątek dotyczący tego etapu programu w wersji macierzowej. Należy pamiętać, że ograniczeniami są maksymalne wymiary macierzy udostępniane przez PLC oraz to, że łączny czas trwania cyklu nie może przekroczyć maksymalnego czasu określonego przez mechanizm nadzoru. Oczywiście łączny czas cyklu, na który największy wpływ ma właśnie ten etap programu, nie może przekraczać założonego okresu próbkowania przyjętego w układzie sterowania.

Trzeci etap wykonywania programu, nazwany zapamiętywaniem zmiennych, ma na celu przechowanie obecnych wartości sygnałów w pamięci, w kolejnych chwilach będą one niezbędne jako wartości opóźnionych wielkości wykorzystywanych w programie. Etap ten rozpoczyna się



Rys. 1. Cykl PLC i struktura wykonania programu
Fig. 1. PLC cycle and structure of program execution

nadpisaniem zmiennych najstarszych (najbardziej opóźnionych) sygnałów tymi z poprzednich chwil czasowych, aż do zapisania wartości bieżących, które w następnym cyklu będą opóźnione o jeden okres próbkowania.

Dla ilustracji trzech etapów ogólnego schematu programu (nie jako przykłady zaawansowanych działań przeznaczonych dla PLC) przedstawiono (tab. 1) proces realizacji całkowania metodą prostokątów, różniczkowania numerycznego metodą ilorazu różnicowego wstecz oraz modelu obiektu inercyjnego drugiego rzędu dyskretyzowanego metodą ZOH.

W tej postaci programu pochodna sygnału $x(k)$ będzie poprawnie liczona od drugiego kroku.

Dla algorytmów korzystających z sygnału opóźnionego wyłącznie o jeden krok i tylko raz, w obliczeniach można połączyć etapy II i III. Przykładem jest tu operacja całkowania (tab. 1), w której wspólny etap może mieć postać I := I + e * T; czyli jest to jednocześnie forma uaktualnienia i zapamiętania danych. Zmienna I1 jest wtedy zbędna, a etap inicjalizacji zmiennych przyjmie postać I := 0; Pozwala to zaoszczędzić pamięć i liczbę operacji do wykonania – jest to forma strojenia kodu. Podany proces realizacji równań różnicowych w PLC ma charakter ogólny, uniwersalny, ale przy odpowiedniej analizie operacji, zmiennych i logiki można „zoptymalizować” kod.

7. Operacje macierzowe

Sterowniki programowalne nie mają standardowo zaimplementowanych operacji arytmetycznych na macierzach. Istnieją rozkazy przesyłania, obracania, przeszukiwania i porównywania macierzy, jednak są one nieprzydatne do realizacji algorytmów opisanych w artykule. W normie IEC 61131 określono takie struktury danych jak macierz, ale producenci umożliwiają pracę z macierzami w sposób skalarny, tj. odwołując się do poszczególnych komórek macierzy, a nie wykonując operacji na całej macierzy. Dodatkowo producenci wprowadzili ograniczenia na wymiary danych, np. GE Fanuc w serii RX3i dopuszcza maksymalną liczbę danych w macierzy do 9999. Obsługa operacji macierzowych – dodawanie, odejmowanie, mnożenie, transponowanie i odwracanie macierzy musi być zrealizowana programowo za pomocą operacji na skalarach, co jest utrudnieniem na etapie kodowania, ale również czasochłonne podczas wykonywania programu. Zrealizowanie operacji odwracania macierzy wymaga też przechowywania wyników cząstkowych, czyli dodatkowej pamięci z zasobów PLC. Model programowania PLC nie jest na tyle elastyczny, nie umożliwia dynamicznego (podczas działania programu) przydziału pamięci. Rozmiary i typy zmiennych muszą być zdefiniowane na etapie konfiguracji PLC. Utrudnia to, ale nie uniemożliwia realizacji zadań.

Tab. 1. Ilustracja realizacji programowej całkowania, różniczkowania i modelu inercji II-go rzędu w PLC
Tab. 1. Example of software implementation of integration, differentiation and II-order inertia model in the PLC

	całkowanie	**różniczkowanie**	**inercja II-go rzędu**
opis czas ciągły	$I = \int_{tp}^{tk} e(\tau)\,d\tau$	$D = \frac{dx(t)}{dt}$	$T_1T_2\frac{d^2y}{dt^2} + (T_1+T_2)\frac{dy}{dt} + y(t) = ku(t)$ $T_1 = 1,\ T_2 = 3,$
opis po dyskretyzacji T – okres próbkowania $k-1,\ k,\ k+1,\ \ldots$ kolejne dyskretne chwile czasu	$I(k) = I(k-1) + Te(k)$	$I(k) = I(k-1) + Te(k)$	dla $T = 1$ s $y(k) = 1{,}084\,y(k-1) - 0{,}2636\,y(k-2) +$ $+ 0{,}1091\,u(k-1) + 0{,}07004\,u(k-2)$
Lista i znaczenie zmiennych	I – I(k) I1–I(k–1)	x–x(k) x1–x(k-1)	y–y(k) y1–y(k–1) y2–y(k–2) u–u(k) u1–u(k–1) u2–u(k–2)
P R O G R A M w języku ST			

Etap I
inicjalizacja zmiennych
	I1 : = 0;	x1 : = 0;	y1 := 0;
			y2 := 0;
			u1 := 0;
			u2 := 0;

Etap II uaktualnienie zmiennych	I : = I1+e*T;	D : = (x–x1)/T;	y:=1.084*y1–0.2636*y2+0.1091*u1 +0.07004*u2;

Etap III
zapamiętanie zmiennych
	I1 : = I;	x1 := x;	y2: = y1;
			y1: = y;
			u2: = u1;
			u1: = u;

8. Kompromis między efektywnością czasową, rozmiarem i czytelnością programu

Najkorzystniejszym rozwiązaniem dla programisty jest język ST i zastosowanie dozwolonych w tym języku pętli i indeksowania macierzy. W ten sposób tworzone są zwarte w treści i przejrzyste programy niewielkie w rozmiarze, ale nie najszybsze ze względu na dużą liczbę skoków koniecznych do zrealizowania pętli. W zastosowaniach, w których istnieje zagrożenie przekroczenia maksymalnego dopuszczalnego czasu trwania cyklu można program w języku ST napisać bez pętli w sposób ciągły – instrukcja po instrukcji. Znacznie zwiększy się objętość programu, jednak zmniejszy się czas jego wykonywania. Przy realizacji mnożenia dwóch macierzy o wymiarach 15×15 za pomocą operacji skalarnych powstaje konieczność wykonania 3375 operacji mnożenia i tyle samo dodawania. Dla ilustracji tych rozważań wykonano testy mnożenia macierzy w różnych wersjach realizacji programistycznej. W macierzach znajdowały się liczby zmiennoprzecinkowe 32-bitowe. Testy wykonano dla dwóch jednostek centralnych GE Fanuc RX3i. Wyniki przedstawione w tab. 2 wyrażone są w milisekundach i bajtach.

Z danych zawartych w tab. 2 wynika, że postać kodu ma istotny wpływ na szybkość wykonania programu i jego objętość. W przypadku kodu pisanego bez pętli znacznie zwiększa się prędkość wykonania programu, ale kosztem jego objętości.

Ręczne wprowadzanie kodu bez pętli dla dużych macierzy jest zadaniem bezsensownym ze względu na ogromną liczbę operacji do rozpisania. Ciekawym rozwiązaniem jest automatyczne generowanie tego kodu. Program w języku IL i ST jest plikiem tekstowym. Można zatem korzystając z dowolnego języka programowania (na PC) napisać program korzystający z pętli, który wygeneruje tekst w formacie programu w danym języku nie zawierającym pętli. Przykład programu w formacie m-pliku języka MATLAB do generowania procedury mnożenia macierzy w języku ST przedstawiono na rys. 2, a jego wynik dla zmiennej rozmiar = 2 na rys. 3.

```
fid = fopen(,ProgramST.txt','w');
  for i = 1:rozmiar
    for k = 1:rozmiar
        fprintf(fid,strcat(,C[,,int2str(i-1),
        ,,',int2str(k-1),']:=0.0;\n'));
        for j = 1:rozmiar
        fprintf(fid,strcat(,C[,,int2str(i-1),
        ,,',int2str(k-1),']:=','C[,,int2str(i-1),
        ,,',int2str(k-1),']+','A[,,int2str(i-1),
        ,,',int2str(j-1),']*','B[,,int2str(j-1),
        ,,',int2str(k-1),']; \n , ));
        end;
    end;
  end;
fclose(fid);
```

Rys. 2. Program dla PC generujący program dla PLC w języku ST

Fig. 2. PC program for generating program for the PLC in ST language

Tab. 2. Szybkość wykonania i rozmiar w pamięci programu w różnych formach

Tab. 2. Speed of execution and the size of the memory in various program forms

	ST w pętlach	ST bez pętli
PAC RX3i CPU310	48 ms; 660 B	6,1 ms; 84 448 B
PAC RX3i CPE305	18,5 ms; 660 B	4,4 ms; 88 449 B

```
C[0,0]:=0.0;
C[0,0]:=C[0,0]+A[0,0]*B[0,0];
C[0,0]:=C[0,0]+A[0,1]*B[1,0];
C[0,1]:=0.0;
C[0,1]:=C[0,1]+A[0,0]*B[0,1];
C[0,1]:=C[0,1]+A[0,1]*B[1,1];
C[1,0]:=0.0;
C[1,0]:=C[1,0]+A[1,0]*B[0,0];
C[1,0]:=C[1,0]+A[1,1]*B[1,0];
C[1,1]:=0.0;
C[1,1]:=C[1,1]+A[1,0]*B[0,1];
C[1,1]:=C[1,1]+A[1,1]*B[1,1];
```

Rys. 3. Program w języku ST dla PLC

Fig. 3. PLC program in ST language

Ewidentną zaletą takiej realizacji programu jest skalowalność. Podobnie można generować również kod w języku IL. Ze względu na objętość kodu bez pętli nie zawsze możliwe jest opracowanie całego programu w ten sposób. Rozsądnym podejściem jest zrealizowanie tylko wybranych fragmentów kluczowych z punktu widzenia wydajności, a resztę kodu w sposób klasyczny (z zastosowaniem pętli).

9. Realizacja programowa algorytmu RLS ze współczynnikiem zapominania RLS_FF

Przykładem ilustrującym realizację algorytmu opisanego rekurencyjnym równaniem różnicowym może być estymacja metodą najmniejszych kwadratów ze współczynnikiem zapominania. Algorytm ten poddany nieznacznym modyfikacjom odpowiada filtrowi Kalmana, z jego pomocą można estymować parametry (jest estymatorem), a generowane estymaty można wykorzystać do budowy regulatora adaptacyjnego lub modelu obiektu. Przedstawiony algorytm estymacji RLS z współczynnikiem zapominania pochodzi z [7]. Jego rekurencyjna postać predestynuje go do realizacji w PLC i nie wymaga realizacji procedury odwracania macierzy, która absorbuje dużo czasu i pamięci. Kryterium dane przez minimalizację funkcji strat

$$\min V_t(\theta) = \sum_{i=1}^{t} \alpha^{t-1} \varepsilon^2(i) \tag{1}$$

Równania definiujące algorytm RLS z współczynnikiem zapominania (RLS_FF)

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)\varepsilon(t) \tag{2a}$$

$$\varepsilon(t) = y(t) - \varphi^T(t)\hat{\theta}(t-1) \tag{2b}$$

$$K(t) = P(t)\varphi(t) \tag{2c}$$

$$P(t) = \frac{1}{\alpha}\left[P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\alpha + \varphi^T(t)P(t-1)\varphi(t)}\right] \tag{2d}$$

gdzie: $\hat{\theta}(t)$ – wektor nieznanych, estymowanych parametrów, $\varepsilon(t)$ – błąd estymacji, różnica między wyjściem z obiektu a wyjściem modelu opartym na obliczonej estymacie, nazywany również błędem predykcji, $\varphi(t)$ – wektor wielkości mierzalnych (regresor), $y(t)$ – wyjście z obiektu, $K(t)$ – wektor wzmocnienia, α – współczynnik zapominania, parametr projektowy, $P(t)$ – macierz wzmocnień.

Niezbędne wartości początkowe to $P(0)$ – wartość macierzy wzmocnień w chwili początkowej, $\hat{\theta}(0)$ – estymaty parametrów w chwili początkowej.

Gdy mamy jakąkolwiek wiedzę *a priori* o parametrach, to możemy wprowadzić te wartości do $\hat{\theta}(0)$, w przeciwnym przypadku zwykle $\hat{\theta}(0) = 0$. Natomiast $P(0)$ zwykle przyjmuje wartość $P(0) = \rho I$, gdzie ρ jest również parametrem projektowym (za [7] zwykle odpowiednio „dużą liczbą"). Parametr projektowy α określa, jak szybko „zapominane" (brane pod uwagę z coraz mniejszą wagą) są wcześniejsze pomiary. Gdy α pomiary nie są w ogóle zapominane i wszystkie mają tę samą wagę. Zwykle wartości są bliskie 1, np. z zakresu (0,95–0,99).

Algorytm RLS_FF zwykle ma charakter macierzowy, a wymiar macierzy zależy od parametryzacji modelu czyli wymiarów wektora $\hat{\theta}(t)$. Został on umownie oznaczony jako nV. W tab. 3 podano zmienne stosowane przy realizacji programu.

Do zrealizowania operacji macierzowych definiujących algorytm RLS_FF za pomocą operacji skalarnych konieczne było wydzielenie elementarnych operacji i uporządkowanie ich w odpowiedniej kolejności w formie etapów do realizacji w formie programu. W tab. 4 przedstawiono listę operacji E1–E12 umożliwiających zrealizowanie macierzowego algorytmu RLS_FF w PLC/PAC.

Niemal wszystkie etapy przedstawione w tab. 4 (za wyjątkiem E12) są operacjami macierzowymi lub wektorowymi – ich zapis nie jest zapisem programu, a jedynie niezbędnych operacji do wykonania; np. realizacja etapu pierwszego, czyli mnożenia macierzy P(k–1) przez wektor regresora zrealizowany w języku ST (rys 4).

Tab. 3. Lista zmiennych w realizacji algorytmu RLS_FF (M macierz 2-wymiarowa, W wektor, S skalar, R rzeczywista, I całkowita)

Tab. 3. Variable list of RLS_FF algorithm (M Matrix 2 dimension, W vector, S scalar, R real, I integer)

Nazwa zmiennej	Opis	Org. w pamięci	Typ	Wymiar
nV	Wymiar wektorów theta i fi	S	int	1
P0	Wartość początkowa macierzy P	M	real	1, nV
theta0	Wartość początkowa wektora parametrów estymowanych	W	real	1, nV
theta	Wektor parametrów estymowanych w chwili k	W	real	1, nV
theta_1	Wektor parametrów estymowanych w chwili k–1	W	real	1, nV
thetaP	Wektor korekty do parametrów estymowanych	W	real	1, nV
K	Wektor wzmocnienia	W	real	nV, 1
eps	Błąd estymacji (predykcji)	S	real	1
y	Wyjście z obiektu	S	real	1
fiT	Regresor transponowany	W	real	1, nV
fi	Regresor	W	real	nV, 1
P	Macierz wzmocnień w chwili k	M	real	nV, nV
P_1	Macierz wzmocnień w chwili k–1	M	real	nV, nV
L1, L2, L3, L4, L5	Wektory pomocnicze przy obliczaniu licznika P	W, M, M, M, M	real	nV, 1; macierze nV, nV
M1, M2, M3	Wektory pomocnicze przy obliczaniu mianownika P	W, S, S	real	1, nV; 1; 1
ym	Wyjście z modelu na podstawie estymat	S	real	1
alfa	Współczynnik zapominania	S	real	1
i, j, k	Zmienne indeksujące macierze	S	int	1;1;1

Tab. 4. Etapy realizacji programowej algorytmu RLS_FF
Tab. 4. Stages of implemtation process of RLS_FF algorithm

Etap I inicjalizacja zmiennych	P_1 = P0 theta_1 = theta0
Etap II uaktualnienie zmiennych	E1: L1=P_1*fi E2: L2=L1*fiT E3: L3=L2*P_1 EL4: M1=fiT*P_1 E5: M2=M1*fi E6: M3=alfa+M2 E7: L4=L3/M3 E8: L5=P_1-L4 E9: P=L5/alfa E10: K=P*fi E11: ym=fiT*theta1 E12: eps=y-ym E13: thetaP=K*eps E14: theta=theta_1+thetaP
Etap III zapamiętanie zmiennych	P_1 = P theta_1 = theta

```
for i := 1 to nV do
  for j := 1 to nV do
      L1[i,1] := L1[i,1] + P_1(i,j) * fi(j);
  end_for
end_for
```

Rys. 4. Kod programu dla etapu E1
Fig. 4. The code for E1 stage

Wielkość kodu całego programu przekracza możliwości zamieszczenia go w artykule. Cały kod oraz inne algorytmy estymacji opracowane w języku ST dla PLC są dostępne na www.eia.pg.gda.pl/kss/estymacja.zip. Opracowane kody zostały sprawdzone pod względem poprawności realizacji i były wykorzystane zarówno na potrzeby estymacji, jak i sterowania adaptacyjnego i adaptacyjno-predykcyjnego.

Wydzielenie etapów i zaproponowanie zmiennych dla programu było przeprowadzone tak, aby zapewnić maksymalną czytelność procesu implementacji. Z pewnością istnieje możliwość strojenia kodu pod względem oszczędności pamięci i szybkości działania programu.

10. Testowanie i strojenie opracowanego oprogramowania dla PLC

Istotnym etapem tworzenia oprogramowania jest faza testów. Testowanie oprogramowania tworzonego dla komputerów w większości przypadków może być prowadzone w urządzeniu końcowym pracującym w warunkach docelowych. Testowanie programów opracowanych dla PLC nie zawsze jest możliwe w takich warunkach. Wymagałoby dostępu do obiektu sterowania i infrastruktury technicznej związanej z systemem sterowania. O ile jest to możliwe dla nowopowstających układów, tak w przypadku modernizacji działającej instalacji oczekuje się załadowania do sterownika sprawdzonego programu i minimalnego czasu przestoju. W związku z tym konieczna jest inna forma testowania oprogramowania, której ostatnim etapem jest uruchomienie na obiekcie. Testowanie oprogramowania bez dostępu do obiektu jest możliwe, ale jego wiarygodność zależy od tego, jak dokładnie można odwzorować poza obiektem warunki, które panują w pętli sterowania. Przeważnie wiąże się to z opracowaniem modelu i skorzystania ze środowisk symulacyjnych umożliwiających spreparowanie warunków wiernie oddających te na obiekcie, a również scenariuszy symulacji pracy w różnych warunkach, w tym nietypowych i awaryjnych. Po osiągnięciu poprawności semantycznej i logicznej, czyli stanu, w którym program realizuje założone działania można rozpocząć strojenie kodu – optymalizację programu. Efektem jest poprawa czytelności oraz funkcjonowania programu, np. szybkości wykonywania lub oszczędnego wykorzystania pamięci.

11. Weryfikacja

Po przetestowaniu kodu dla PLC w sposób symulacyjny, zarówno w pętli otwartej, jak i zamkniętej w wersji programowej i sprzętowej niezbędnym etapem jest weryfikacja poprawności pracy w urządzeniu docelowym podłączonym do obiektu rzeczywistego. Jeżeli modele obiektów były opracowane z dużą dokładnością i wiernością, to testy z obiektem rzeczywistym powinny okazać się w dużej części potwierdzeniem tych osiągniętych symulacyjnie. Ze względu na nieuniknione uproszczenia, pominięcie pewnych aspektów pracy obiektu rzeczywistego, jeszcze na tym etapie mogą ujawnić się usterki w oprogramowaniu. Testy z rzeczywistym obiektem sterowania są ostatnim i najważniejszym etapem weryfikacji oprogramowania. Najczęściej nie można wprowadzić rzeczywistego obiektu w dowolny stan, by przetestować go bez uruchamiania całego procesu technologicznego, zatem może wymagać to obecności programistów, nadzoru technologów i operatorów przy pierwszych pełnych przebiegach systemu sterowania z możliwością przejęcia sterowania ręcznego.

12. Dokumentacja, instrukcje serwisowe i utrzymaniowe

Opracowanie dokumentacji programu jest często lekceważone przez programistów i traktowane jako zło konieczne wymagane przez klienta. Tymczasem w rozbudowanych programach z zaawansowanymi metodami sterowania warto ten etap potraktować poważnie i dokumentację sporządzić w taki sposób, aby ewentualna modernizacja kodu nie oznaczała konieczności żmudnego wdrażania się w opracowany kod, czy wręcz pisania go zupełnie od nowa. Dobrze opracowane instrukcje i komunikaty utrzymaniowe ułatwią obsłudze systemu poprawne interwencje w nietypowych lub awaryjnych sytuacjach. Natomiast instrukcje serwisowe ułatwią sprawną diagnostykę uprawnionemu serwisowi konserwującemu system sterowania lub usuwającemu awarie.

13. Podsumowanie

W wyniku zebranych doświadczeń z przemysłu, pracy dydaktycznej i zaprogramowania wielu algorytmów filtracji, estymacji i sterowania w PLC/PAC podjęto próbę sformułowania oceny obszarów przydatności i nieprzydatności PLC/PAC. Urządzenia te nie są właściwym wyborem dla: sterowania obiektami, procesami, przetwarzaniem sygnałów bardzo szybkich, tj. takich które wymagają interwencji częściej niż w milisekundach, do bardzo złożonych, czasochłonnych obliczeń np. optymalizacji, do obliczeń wymagających ogromnej pamięci, do przetwarzania danych wymagających łącznie czasu ponad maksymalny czas dopuszczalny przez układ *watchdog*, do obliczeń wymagających/wykorzystujących zewnętrzne biblioteki programistyczne. Jednak PLC/PAC nadaje się do realizacji większości algorytmów filtracji, estymacji, sterowania zdefiniowanych przez równania różnicowe. Wskazano ograniczenia PLC natury pamięciowej, ograniczenia na maksymalny czas wykonania cyklu i ograniczenia natury programistycznej. Wymienione ograniczenia eliminują wykorzystanie PLC/PAC dla bardzo rozbudowanych obiektów posiadających bardzo dużą liczbę parametrów i zmiennych procesowych, których zapisanie i przetworzenie w czasie jednego cyklu przekroczy opisane ograniczenia. Wtedy jednak przeważanie istnieje możliwość dekompozycji i realizacji zadań nie przez pojedynczy PLC tylko rozdzielenia ich pomiędzy wiele urządzeń tego typu. W oparciu o opisaną w artykule strukturę i metodologię budowy programu z sukcesem zaimplementowano w PLC/PAC algorytmy filtracji pasmowej, estymacji parametrycznej metodą najmniejszych kwadratów i metodą gradientową, algorytmy liniowego sterowania predykcyjnego, algorytmy sterowania adaptacyjnego zarówno w wersji samostrojącej ST, jak i wersji nadążania za trajektorią MRAC. Zatem obecny stan zaawansowania PLC i PAC stwarza możliwości zaimplementowania algorytmów lokowanych dotąd w wyższych warstwach modelu komputerowego systemu sterowania – w warstwie nadrzędnej/nadzorczej oraz niektórych zadań z warstwy nazywanej umownie optymalizacyjną. Implementacja zaawansowanych metod sterowania w PLC pozwala znacznie podnieść funkcjonalność PLC (standardowo wyposażonych tylko w najprostsze algorytmy), przeorientować zadania w poszczególnych warstwach modelu komputerowego systemu sterowania i zbliżyć się ku znacznie tańszym rozwiązaniom opartym o PLC/PAC do systemów klasy DCS.

Bibliografia

1. PN-EN 61131-3:2004 Sterowniki programowalne – część 3:Języki programowania, PKN, 2004.
2. Kacprzak S., *Programowanie sterowników PLC zgodnie z normą IEC61131-3 w praktyce*, Wydawnictwo BTC, 2011.
3. Broel-Plater B., *Układy wykorzystujące sterowniki PLC. Projektowanie algorytmów sterowania*, PWN, 2009.
4. Kwaśniewski J., *Sterowniki PLC w praktyce inżynierskiej*, Wydawnictwo BTC, 2008.
5. Pietrusewicz K., Dworak P., *Programowalne sterowniki automatyki PAC*, Wydawnictwo Nakom, 2007.
6. Legierski T., Kasprzyk J., Wyrwał J., Hajda J., *Programowanie sterowników PLC*, Wydawnictwo pracowni komputerowej Jacka Skalmierskiego, 1998.
7. Soderstrom T., Stoica P., *Identyfikacja systemów*, PWN, 1997. ■

The software implementation of filtering, estimation and control algorithms in PLC / PAC

Abstract: PLCs (Programmable Logic Controllers) are the main industrial platform for the implementation of direct control algorithms. PLC producers provide to programmers only basic, simple control methods. With the development of PLC and their successors (Programmable Automation Controller – PAC) appeared increased CPU and memory capabilities of the equipment and fuller implementation of programming languages defined in the standard IEC-61131-3. PLCs and PACs now have the computing power and memory of the personal computer PC a few years ago, they can also be programmed in high level languages using the variables in the form of a matrix. Taking into account the limitations and specifics of the PLC and PAC in these devices it is possible to implement almost any discrete control, estimation and filtering algorithm. This paper presents an approach to programming filtering, estimation and control algorithms defined by differential equations. The methodology of software development for PLC / PAC is presented. For presentation of implementation process of filtering and estimation algorithm least squares with forgetting factor (RLS_FF) is used.

Keywords: Programmable Logic Controllers, discrete control algorithms, tuning and verification of software, estimation RLS_FF

dr inż. Jarosław Tarnawski

Adiunkt w Katedrze Inżynierii Systemów Sterowania, Wydział Elektrotechniki i Automatyki Politechniki Gdańskiej. Obszary zainteresowań badawczych: synteza komputerowych systemów sterowania, przemysłowe sieci informatyczne, sterowanie obiektami z opóźnieniami, systemy środowiskowe.

e-mail: j.tarnawski@eia.pg.pl