

**Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska**

**Projektowanie układów sterowania
(projekt grupowy)**

**Sprawozdanie z projektu i ćwiczenia laboratoryjnego
nr 4, zadanie nr 15**

Michał Pióro, Radosław Ślepowroński, Jan Szymczak

Warszawa, 2024

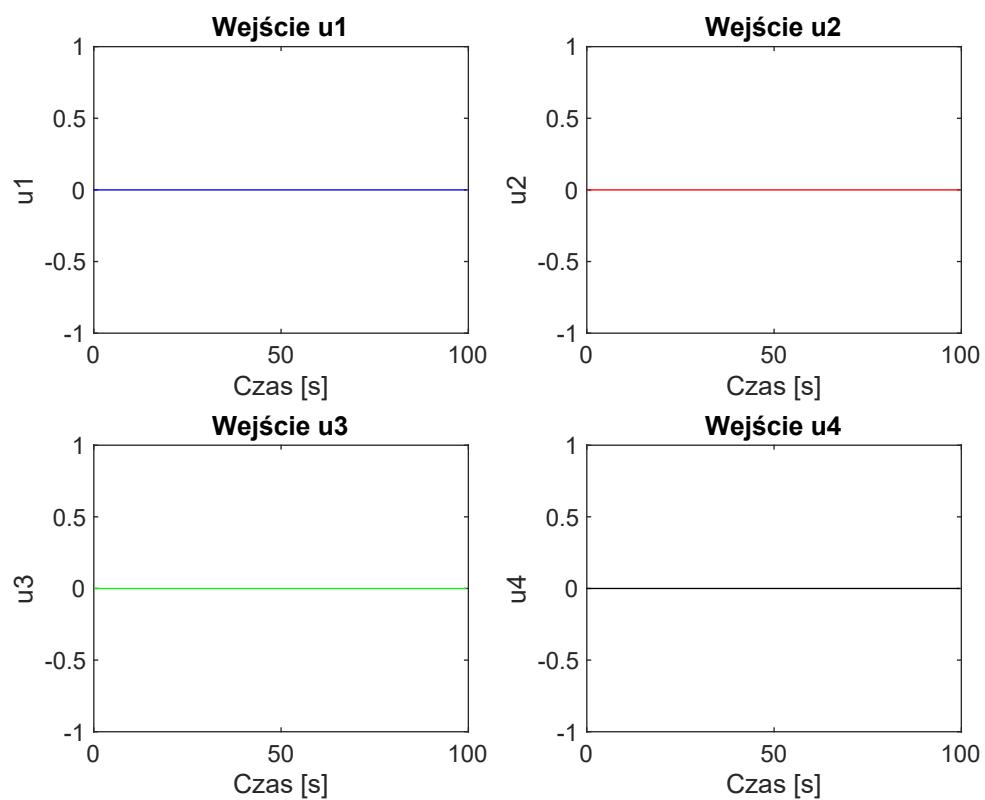
Spis treści

1. Projekt	2
1.1. Poprawność wartości U_{pp} , Y_{pp}	2
1.2. Odpowiedzi skokowe	3
1.3. Algorytmy regulacji	4
1.3.1. Algorytm PID	4
1.3.2. Oszczędny algorytm DMC	6
1.3.3. Klasyczny algorytm DMC	9
1.4. Dobór nastaw metodą eksperymentalną	10
1.4.1. Algorytm PID	10
1.4.2. Algorytm DMC	19
1.5. Dobór nastaw metodą optymalizacyjną	23
1.5.1. Algorytm PID	23
1.5.2. Algorytm DMC	30
1.6. Porównanie regulatorów DMC	34
2. Laboratorium - stanowisko grzewczo-chłodzące	38
2.1. Komunikacja ze stanowiskiem i organizacja projektu w GxWorks3	38
2.2. Wyznaczenie punktu pracy	40
2.3. Mechanizm zabezpieczający przed uszkodzeniem stanowiska	40
2.4. Automaty stanów	42
2.5. Dwupętlowy regulator PID	44
2.5.1. Implementacja	44
2.5.2. Strojenie	46
2.6. Regulator DMC 2×2 w wersji analitycznej	50
2.6.1. Odpowiedzi skokowe	50
2.6.2. Implementacja	51
2.6.3. Strojenie	54
2.7. Panel operatora	57
3. Laboratorium - stanowisko INTECO TCRANE	58
3.1. Struktura kodu	58
3.2. Bazowanie	59
3.3. Zabezpieczenie zakresu ruchu elementów ruchomych	60
3.4. Charakterystyka statyczna	60
3.5. Regulacja położenia wózka	61
3.5.1. Automat stanów wartości zadanej	61
3.5.2. Wykorzystanie własnej implementacji algorytmu PID	61
3.5.3. Wykorzystanie wbudowanego algorytmu PID	62
3.6. Panel operatora	63

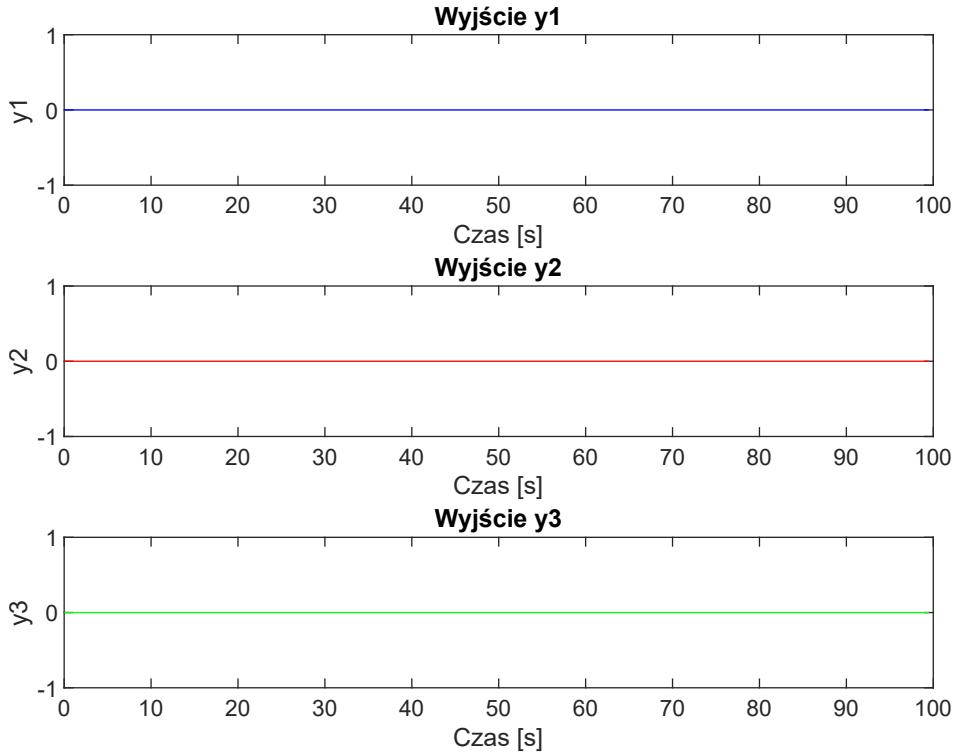
1. Projekt

1.1. Poprawność wartości U_{pp} , Y_{pp}

Aby zbadać poprawność wartości w punkcie pracy, przeprowadzono symulację, w której na wejście obiektu podawane są stałe wartości $u_1 = u_2 = u_3 = u_4 = 0$. Tak prezentują się wyniki symulacji:



Rys. 1.1. Sprawdzenie poprawności punktu pracy dla wejść

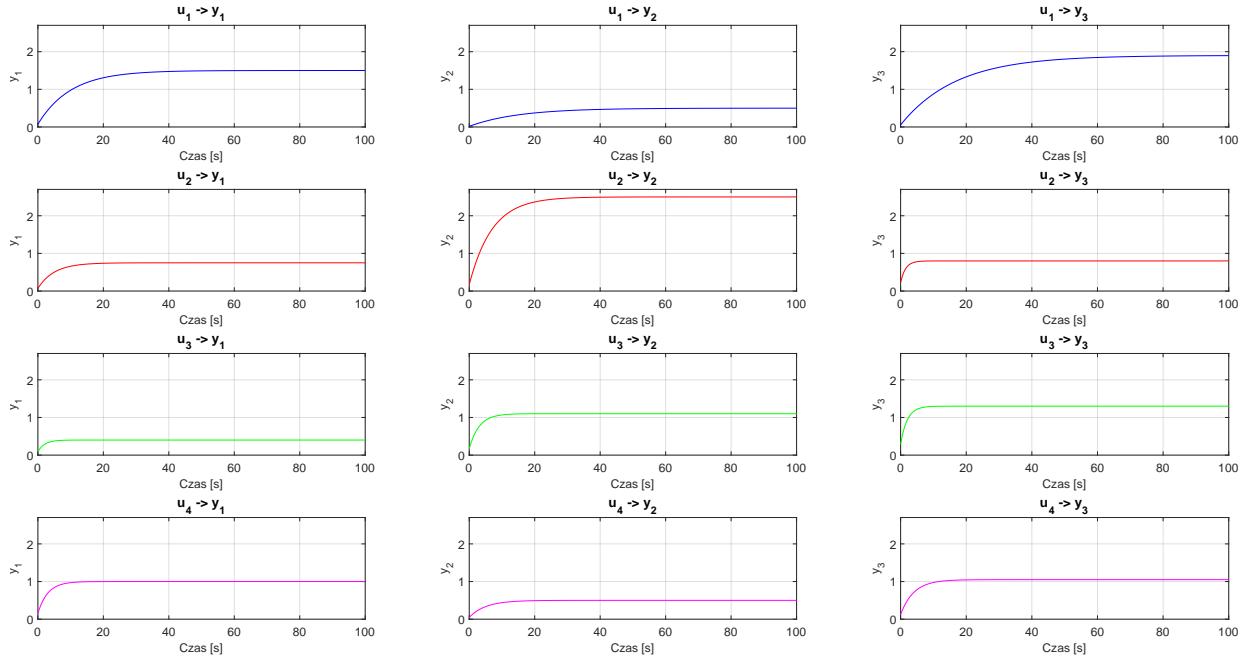


Rys. 1.2. Sprawdzenie poprawności punktu pracy dla wyjść

Jak widać, obiekt jest stabilny i podane wartości sygnałów faktycznie stanowią punkt pracy tego obiektu, ponieważ nie nastąpiły żadne zmiany sygnałów w prezentowanej symulacji.

1.2. Odpowiedzi skokowe

Przeprowadzono symulację odpowiedzi skokowych dla wszystkich 12 torów procesu:



Rys. 1.3. Odpowiedzi skokowe wszystkich 12 torów procesu

Większość torów charakteryzuje się innym wzmacnieniem oraz czasem, po którym następuje stabilizacja wyjścia. W tym momencie na podstawie zaprezentowanych odpowiedzi skokowych można określić, na które wyjścia powinny wpływać poszczególne wejścia w (teoretycznie) najlepszym regulatorze PID badanym w kolejnych sekcjach projektu. Dla wyjścia y_3 widać, że największe wzmacnienie występuje w torze związanym z wejściem u_1 , a więc to uchyb e_3 powinien oddziaływać na te wejście. Idąc dalej, uchyb e_2 będzie wpływać na u_2 z uwagi na największe wzmacnienie w tym torze, a uchyb e_1 na wejście u_4 - w tym przypadku nieco większe wzmacnienie występuje w torze u_1 , ale te wejście, z uwagi na większą różnicę we wzmacnieniu względem innych torów, zostanie "powiązane" z uchybem e_3 jak powyżej napisano. Wejście u_3 zatem pozostanie zerowane.

1.3. Algorytmy regulacji

1.3.1. Algorytm PID

Wielowymiarowy algorytm regulacji PID został zaimplementowany przy pomocy 3 autorskich funkcji, z których pierwsza `PID_MIMO_offline` 1.3 pozwala na wykonanie niezbędnych obliczeń offline. Jej parametry wejściowe zostały przedstawione w tabeli 1.1, a wyjściowe w tabeli 1.2.

Następną funkcją jest `PID_MIMO_U`, która pozwala na wykonanie obliczeń online regulatora PID wyznaczając wartości sterowań dla każdego jednowymiarowego regulatora PID użytego w wielowymiarowym PID. Jej parametry wejściowe zostały przedstawione w tabeli 1.3, a wyjściowe w tabeli 1.4. Warto zauważyć zastosowanie argumentu `U_nums` pozwalającego na określenie, którym wejściem będzie sterował jednowymiarowy regulator PID przypisany do danego wyjścia procesu. Regulatorzy są przyporządkowane do wyjść z numerami sobie odpowiadającymi co oznacza, że np. parametry PID-a dla pierwszego wyjścia są w wektorze podawane jako pierwsze elementy.

Ostatnią funkcją jest `p3_PID_MIMO` pozwalającą na zasymulowanie działania wielowymiarowego regulatora PID dla badanego wielowymiarowego procesu. Jej parametry wejściowe zostały przedstawione w tabeli 1.5, a wyjściowe w tabeli 1.6.

Parametr	Opis
K	wektor wzmacnień K regulatorów PID
Ti	wektor stałych całkowania T_i regulatorów PID
Td	wektor stałych różniczkowania T_d regulatorów PID
Tp	czas próbkowania

Tab. 1.1. Opis parametrów wejściowych `PID_MIMO_offline`

Parametr	Opis
r2	wektor parametrów r_2 regulatorów dyskretnych PID
r1	wektor parametrów r_1 regulatorów dyskretnych PID
r0	wektor parametrów r_0 regulatorów dyskretnych PID

Tab. 1.2. Opis parametrów wyjściowych `PID_MIMO_offline`

Listing 1.1. Obliczenia offline PID

```
function [r2, r1, r0] = PID_MIMO_offline(K, Ti, Td, Tp)

for i=1:max(size(K))
    r2(i) = K(i)*Td(i)/Tp;
    r1(i) = K(i)*(Tp/(2*Ti(i)) - (2*Td(i))/Tp - 1);
end
```

```
r0(i) = K(i)*(1 + Tp/(2*Ti(i)) + Td(i)/Tp);
end
```

Parametr	Opis
k	numer próbki czasowej
Y	wektor wartości wyjścia regulowanego procesu w chwili k
Y_zad	wektor wartości zadanych wyjścia regulowanego procesu w chwili k
U	macierz wartości sterowania regulowanego procesu
E	macierz wartości uchybów regulowanego procesu
R0	wektor parametrów r_0 regulatorów dyskretnych PID
R1	wektor parametrów r_1 regulatorów dyskretnych PID
R2	wektor parametrów r_2 regulatorów dyskretnych PID
U_nums	wektor przyporządkowania sterowań regulatorów PID wyjście procesu

Tab. 1.3. Opis parametrów wejściowych PID_MIMO_U

Parametr	Opis
U	macierz wartości sterowania regulowanego procesu
E	macierz wartości uchybów regulowanego procesu

Tab. 1.4. Opis parametrów wyjściowych PID_MIMO_U

Listing 1.2. Obliczenia online PID

```
function [U,E]=PID_MIMO_U(k,Y,Y_zad,U,E,R0,R1,R2,U_nums)
E(:,k)=Y_zad-Y;

U(:,k)=U(:,k-1);
for i=1:3
    U_n=U_nums(i);
    U(U_n,k)=R2(i)*E(i,k-2);
    U(U_n,k)=U(U_n,k)+R1(i)*E(i,k-1);
    U(U_n,k)=U(U_n,k)+R0(i)*E(i,k);
    U(U_n,k)=U(U_n,k)+U(U_n,k-1);
end
end
```

Parametr	Opis
kk	ilość iteracji symulacji
Tp	czas próbkowania
Y_zad	macierz zadanych wartości wyjściowych regulowanego procesu
K	wektor wzmacnień K regulatorów PID
Ti	wektor stałych całkowania T_i regulatorów PID
Td	wektor stałych różniczkowania T_d regulatorów PID
U_nums	wektor przyporządkowania sterowań regulatorów PID wyjście procesu

Tab. 1.5. Opis parametrów wejściowych p3_PID_MIMO

Parametr	Opis
Y	macierz wartości wyjściowych regułowanego procesu
U	macierz wartości sterowania regułowanego procesu
E	wektor wartości błędów kwadratowych wielowymiarowego regulatora PID

Tab. 1.6. Opis parametrów wyjściowych p3.PID_MIMO

Listing 1.3. Symulacja PID

```

function [Y,U,E] = p3_PID_MIMO(kk,Tp,Y_zad,K,Ti,Td,U_nums)
[R2,R1,R0]=PID_MIMO_offline(K,Ti,Td,Tp);
ny=size(Y_zad,1);
U=zeros(4, kk+4);
Y=zeros(3, kk+4);
Y_zad=[repmat(Y_zad(:,1), 1, 4), Y_zad];
E_PID=zeros(ny,kk);
E=0;
for k=5:kk+4
    [Y(1,k),Y(2,k),Y(3,k)] = symulacja_objektu15y_p4( ...
        U(1, k-1), U(1, k-2), U(1, k-3), U(1, k-4), ...
        U(2, k-1), U(2, k-2), U(2, k-3), U(2, k-4), ...
        U(3, k-1), U(3, k-2), U(3, k-3), U(3, k-4), ...
        U(4, k-1), U(4, k-2), U(4, k-3), U(4, k-4), ...
        Y(1, k-1), Y(1, k-2), Y(1, k-3), Y(1, k-4), ...
        Y(2, k-1), Y(2, k-2), Y(2, k-3), Y(2, k-4), ...
        Y(3, k-1), Y(3, k-2), Y(3, k-3), Y(3, k-4));
[U,E_PID]=PID_MIMO_U(k,Y(:,k),Y_zad(:,k),U,E_PID,R0,R1,R2,U_nums);

% Obliczanie błędu kwadratowego
e=Y_zad(:,k)-Y(:,k);
E=E+e'*e;
end
U=U(:, 5:end);
Y=Y(:, 5:end);
end

```

1.3.2. Oszczędny algorytm DMC

Implementację oszczędnego wielowymiarowego regulatora DMC rozpoczęto od zaimplementowania funkcji pozwalających na obliczanie macierzy M 1.4 oraz M_P 1.5. Poza odpowiednimi parametrami regulatora DMC, każda z tych funkcji przyjmuje tensor S, zawierający odpowiednio przechowywane odpowiedzi skokowe każdego wyjścia dla każdego wyjścia.

Następnie stworzono funkcję DMC_MIMO_offline 1.6 pozwalającą na obliczenie macierzy K_e oraz K_u wykorzystywanych do obliczania wartości sterowania przez oszczędną wersję regulatora DMC. Argumenty wejściowe tej funkcji to, poza wcześniej wspomnianym tensorem S, standardeowe parametry wielowymiarowego regulatora DMC w standardowej postaci.

Kolejną z funkcji jest DMC_MIMO_dU 1.7 wyznaczającą wektor zmian sterowań dU wielowymiarowego regulatora DMC. Przyjmowane przez nią parametry przedstawiono w tabeli 1.7.

Ostatnią funkcją jest p3_DMC_MIMO pozwalającą na zasymulowanie działania wielowymiarowego regulatora DMC dla badanego wielowymiarowego procesu. Jej parametry wejściowe zostały przedstawione w tabeli 1.8, a wyjściowe w tabeli 1.9

Listing 1.4. Obliczenia macierzy M

```

function [M] = DMC_MIMO_M(S, N, Nu)
[ny, nu, ~] = size(S);
M = zeros(ny * N, nu * Nu); % Inicjalizacja macierzy M

% Budowa macierzy M
for i = 1:N
    for j = 1:Nu
        if i >= j
            % Wyciągnięcie odpowiedniego fragmentu odpowiedzi skokowej
            Sp = S(:, :, i-j+1);
        else
            continue;
        end
        % Wstawienie bloku do macierzy M
        M((i-1)*ny+1:i*ny, (j-1)*nu+1:j*nu) = Sp;
    end
end
end

```

Listing 1.5. Obliczenia macierzy M_p

```

function [Mp] = DMC_MIMO_Mp(S, D, N)
[ny, nu, ~] = size(S);
Mp = zeros(ny * N, nu * (D - 1));

% Budowa macierzy Mp
for i = 1:N
    for j = 1:(D - 1)
        % Wyznaczenie różnicy odpowiedzi skokowych
        Sp_diff = S(:, :, i + j) - S(:, :, j);
        % Wstawienie bloku różnic do Mp
        Mp((i-1)*ny+1:i*ny, (j-1)*nu+1:j*nu) = Sp_diff;
    end
end
end

```

Listing 1.6. Obliczenia offline oszczędnego DMC

```

function [Ke, Ku] = DMC_MIMO_offline(S, D, N, Nu, lambda, psi)
[ny, nu, ~] = size(S);
M=DMC_MIMO_M(S,N,Nu);
Mp=DMC_MIMO_Mp(S,D,N);

Psi = kron(eye(N), diag(psi));
Lambda = kron(eye(Nu), diag(lambda));

K=(M'*Psi*M+Lambda)\M'*Psi;
K1=K(1:nu,:);
Ke=0;
for i = 1:N
    Ke = Ke + K1(:,1+(i-1)*ny:i*ny);

```

```

end
Ku=K1*Mp ;
end

```

Parametr	Opis
k	numer próbki czasowej
y	wektor wartości wyjścia regulowanego procesu w chwili k
y_zad	wektor wartości zadanych wyjścia regulowanego procesu w chwili k
Ke	macierz wzmocnień uchybów
Ku	macierz wzmocnień sterowań
dUp	macierz przeszłych przyrostów sterowań regulatora DMC

Tab. 1.7. Opis parametrów wyjściowych DMC_MIMO_dU

Listing 1.7. Obliczenia online oszczędnego DM

```

function [dU]=DMC_MIMO_dU(y,y_zad,Ke,Ku,dUp)
dU=Ke*(y_zad-y);
nu=size(Ke,1);
for i=1:size(Ku,2)/nu
    dU=dU-Ku(:,1+(i-1)*nu:i*nu)*dUp(:,i);
end
end

```

Parametr	Opis
kk	ilość iteracji symulacji
Y_zad	macierz zadanych wartości wyjściowych regulowanego procesu
D	horyzont dynamiki
N	horyzont predykcji
Nu	horyzont sterowań
lambda	wektor kar za zmienność sterowań
psi	wektor priorytetów wyjścia procesu

Tab. 1.8. Opis parametrów wejściowych p3_DMC_MIMO

Parametr	Opis
Y	macierz wartości wyjściowych regulowanego procesu
U	macierz wartości sterowania regulowanego procesu
E	wektor wartości błędów kwadratowego wielowymiarowego regulatora PID

Tab. 1.9. Opis parametrów wyjściowych p3_DMC_MIMO

Listing 1.8. Symulacja oszczędnego DMC

```

function [Y,U,E] = p3_DMC_MIMO(kk,Y_zad,D,N,Nu,lambda,psi)
S=p2_odpowiedzi_skokowe(D+N+Nu);
[Ke,Ku]=DMC_MIMO_offline(S,D,N,Nu,lambda,psi);

U = zeros(4, kk+4);
Y = zeros(3, kk+4);
Y_zad = [repmat(Y_zad(:,1), 1, 4), Y_zad];
dUp=zeros(4,D-1);

```

```

E=0;
for k=5:kk+4
    [Y(1,k),Y(2,k),Y(3,k)] = symulacja_obiektu15y_p4( ...
        U(1, k-1), U(1, k-2), U(1, k-3), U(1, k-4), ...
        U(2, k-1), U(2, k-2), U(2, k-3), U(2, k-4), ...
        U(3, k-1), U(3, k-2), U(3, k-3), U(3, k-4), ...
        U(4, k-1), U(4, k-2), U(4, k-3), U(4, k-4), ...
        Y(1, k-1), Y(1, k-2), Y(1, k-3), Y(1, k-4), ...
        Y(2, k-1), Y(2, k-2), Y(2, k-3), Y(2, k-4), ...
        Y(3, k-1), Y(3, k-2), Y(3, k-3), Y(3, k-4));
    dU=DMC_MIMO_dU(Y(:,k),Y_zad(:,k),Ke,Ku,dUp);
    U(:,k)=U(:,k-1)+dU;
    dUp(:,2:end)=dUp(:,1:end-1);
    dUp(:,1)=dU;
    % Obliczanie błędu kwadratowego
    e = Y_zad(:,k) - Y(:,k);
    E = E + e' * e;
end
U = U(:, 5:end);
Y = Y(:, 5:end);
end

```

1.3.3. Klasyczny algorytm DMC

W przypadku klasycznego regulatora DMC wykorzystano wcześniej zaimplementowane funkcje do obliczania macierzy M 1.4 oraz M_p 1.5. Korzystając z nich zaimplementowano funkcję `DMC_MIMO_full_y_offline` 1.9 pozwalającą na zrealizowanie niezbędnych obliczeń offline klasycznego wielowymiarowego regulatora DMC zwracającą macierz K oraz M_p , pozwalające na późniejsze wyznaczanie wartości sterowań procesu. Funkcja ta, tak jak i poprzednio dla oszczędnego DMC 1.6 przyjmuje tensor odpowiedzi skokowych S oraz standardowe parametry wielowymiarowego regulatora DMC.

Następną funkcją jest `DMC_MIMO_full_y_dU` 1.10, która pozwala na obliczenie wektora przyrostów sterowań dla chwili k , korzystając z wzorów klasycznego wielowymiarowego regulatora DMC w związku z czym jako parametr do obliczeń przyjmuje macierz K oraz M_p wyznaczone w obliczeniach offline.

Ostatnią z funkcji jest `p6_DMC_MIMO_full_y_zad` 1.11 pozwalająca na symulację klasycznego wielowymiarowego regulatora DMC dla badanego procesu. Funkcja ta przyjmuje i zwraca te same parametry co funkcja `p3_DMC_MIMO`.

TODO

Listing 1.9. Obliczenia offline klasycznego DMC

```

function [K, Mp] = DMC_MIMO_full_y_offline(S, D, N, Nu, lambda, psi)
M=DMC_MIMO_M(S,N,Nu);
Mp=DMC_MIMO_Mp(S,D,N);

Psi = kron(eye(N), diag(psi));
Lambda = kron(eye(Nu), diag(lambda));

K=(M'*Psi*M+Lambda)\M'*Psi;
end

```

Listing 1.10. Obliczenia online klasycznego DMC

```
function [dU]=DMC_MIMO_full_y_dU(y,y_zad,K,Mp,dUp)
    ny=size(y,1);
    nu=size(dUp,1);
    N=size(K,2)/ny;
    dU=K*(repmat(y_zad, N, 1)-repmat(y, N, 1)-Mp*dUp(:));
    dU=dU(1:nu);
end
```

Listing 1.11. Symulacja klasycznego DMC

```
function [Y,U,E] = p6_DMC_MIMO_full_y_zad(kk,Y_zad,D,N,Nu,lambda,psi)
S=p2_odpowiedzi_skokowe(D+N+Nu);
[K,Mp]=DMC_MIMO_full_y_offline(S, D, N, Nu, lambda, psi);

U = zeros(4, kk+4);
Y = zeros(3, kk+4);
Y_zad = [repmat(Y_zad(:,1), 1, 4), Y_zad];
dUp=zeros(4,D-1);
E=0;
for k=5:kk+4
    [Y(1,k),Y(2,k),Y(3,k)] = symulacja_objektu15y_p4( ...
        U(1, k-1), U(1, k-2), U(1, k-3), U(1, k-4), ...
        U(2, k-1), U(2, k-2), U(2, k-3), U(2, k-4), ...
        U(3, k-1), U(3, k-2), U(3, k-3), U(3, k-4), ...
        U(4, k-1), U(4, k-2), U(4, k-3), U(4, k-4), ...
        Y(1, k-1), Y(1, k-2), Y(1, k-3), Y(1, k-4), ...
        Y(2, k-1), Y(2, k-2), Y(2, k-3), Y(2, k-4), ...
        Y(3, k-1), Y(3, k-2), Y(3, k-3), Y(3, k-4));
    dU=DMC_MIMO_full_y_dU(Y(:,k),Y_zad(:,k),K,Mp,dUp);
    U(:,k)=U(:,k-1)+dU;
    dUp(:,2:end)=dUp(:,1:end-1);
    dUp(:,1)=dU;

    % Obliczanie błędu kwadratowego
    e = Y_zad(:,k) - Y(:,k);
    E = E + e' * e;
end
U = U(:, 5:end);
Y = Y(:, 5:end);
end
```

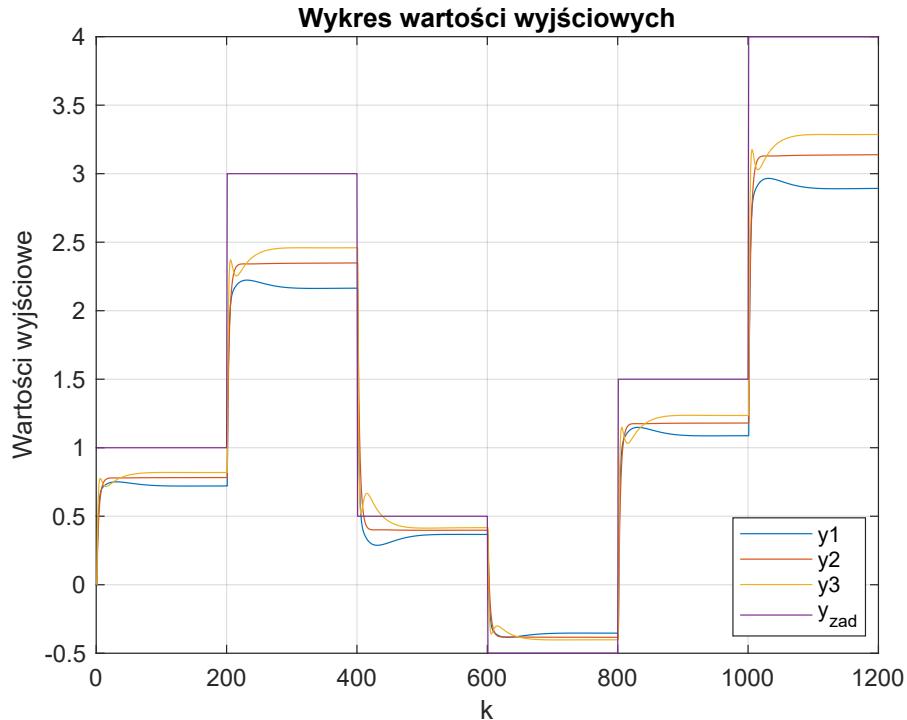
1.4. Dobór nastaw metodą eksperimentalną

1.4.1. Algorytm PID

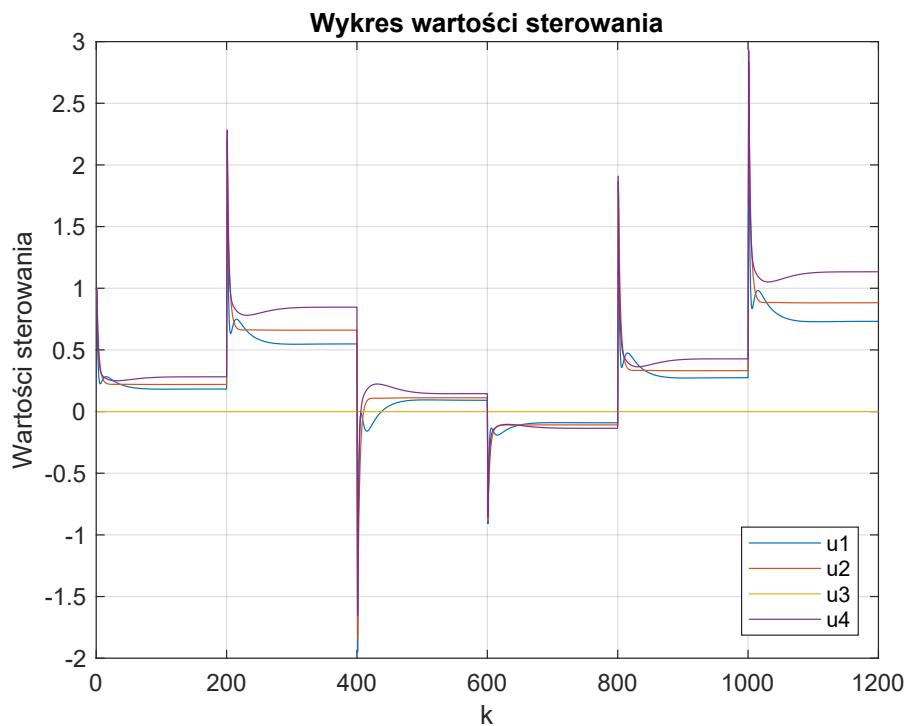
Zgodnie z opisem odpowiedzi skokowych procesu (1.3) zdecydowano się na następujące po-wiązania uchybów z wejściami: $e_1 \rightarrow u_4$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_1$, wejście u_3 jest zerowane. Przyjęto następującą trajektorię wartości zadanych, identyczną dla każdego wyjścia (dlatego poniżej na wykresach jest prezentowana tylko jedna trajektoria, aby pozbyć się zbędnych wpisów w legendzie i zwiększyć czytelność wykresów):

```
yzad(1:200)=1; yzad(201:400)=3; yzad(401:600)=0.5; yzad(601:800)=-0.5;
yzad(801:1000)=1.5; yzad(1001:kk)=4;
```

Proces strojenia rozpoczęto od sprawdzenia poprawności implementacji, a więc przygotowano prosty regulator P dla każdego wyjścia, o wartości wzmacnienia wynoszącym 1.

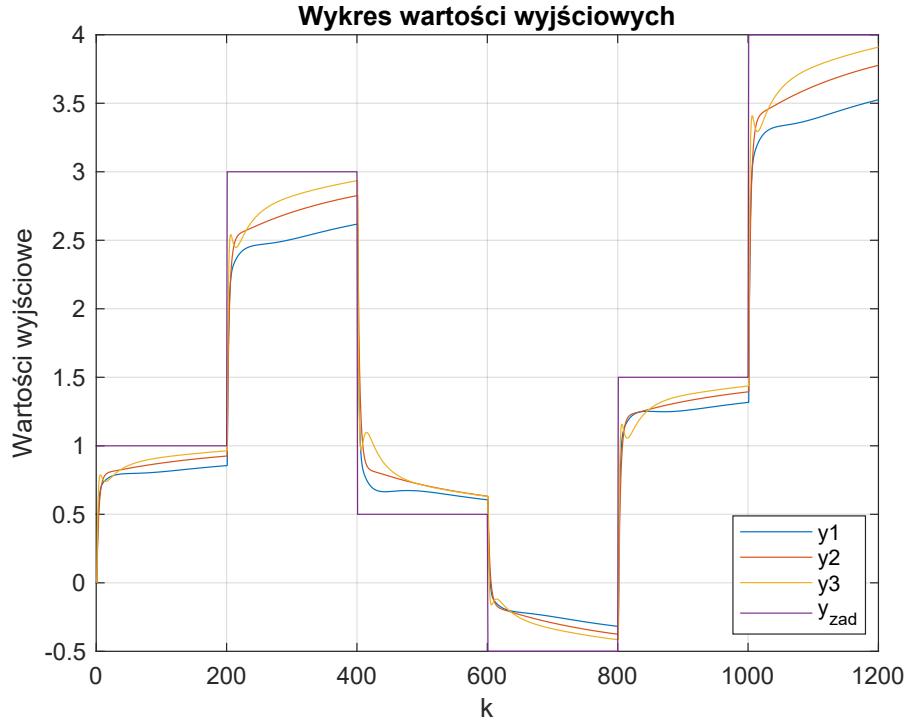


Rys. 1.4. Wyjścia procesu dla regulatorów P i $K = 1$

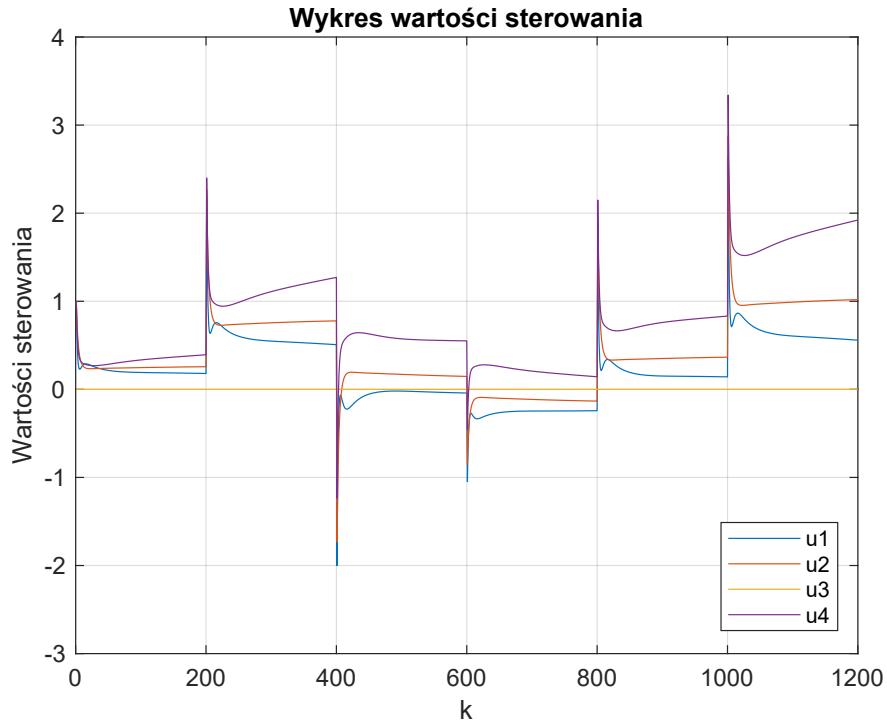


Rys. 1.5. Wejścia procesu dla regulatorów P i $K = 1$

Implementacja wydaje się być poprawna - regulator reaguje odpowiednio na zmiany wartości zadanej. Oczywiście w tym momencie występuje uchyb ustalony, dlatego następnym krokiem było wprowadzenie całkowania, na początku równego dla każdego regulatora $T_i = 80$:



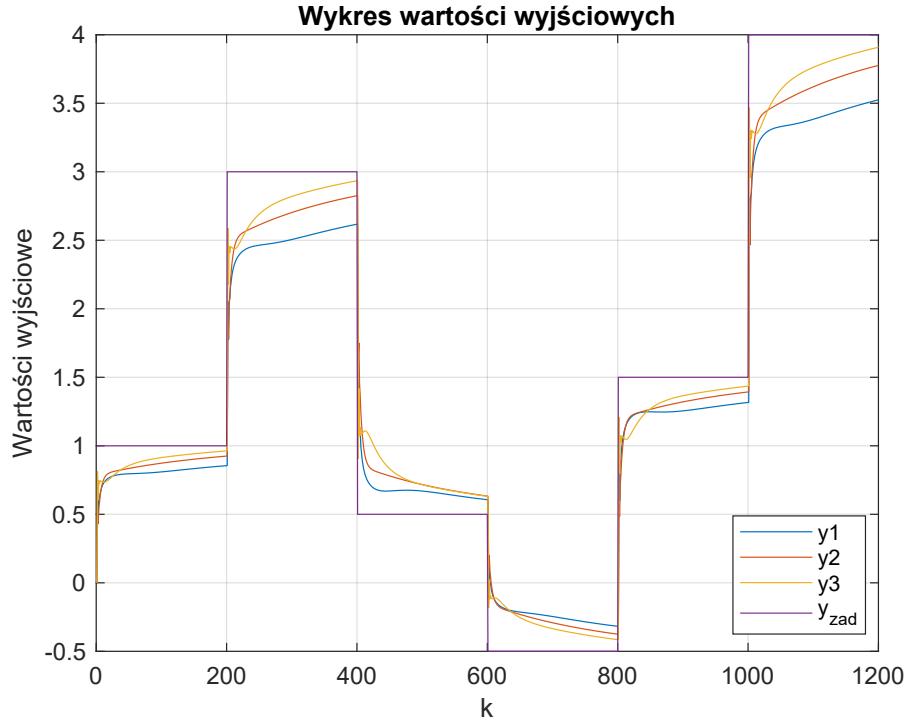
Rys. 1.6. Wyjścia procesu dla regulatorów PI: $K = 1$, $T_i = 80$



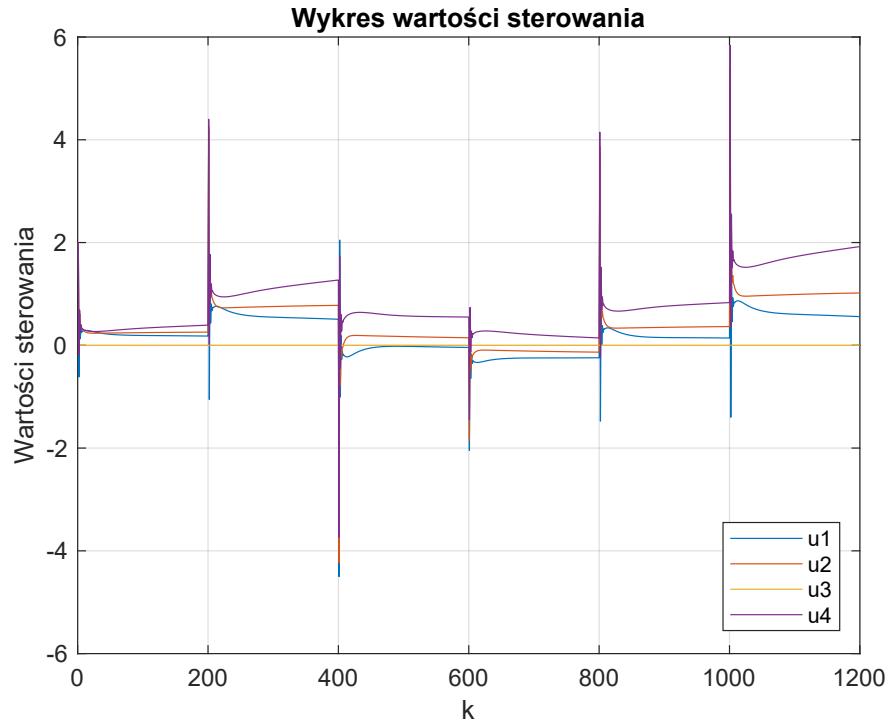
Rys. 1.7. Wejścia procesu dla regulatorów PI: $K = 1$, $T_i = 80$

Całkowanie w tym momencie występuje, widać powolny spadek uchybu, jednak regulacja w

tym momencie następuje zbyt wolno. Rozwiążaniem tego problemu (w jakimś małym stopniu) mogłoby być wprowadzenie różniczkowania, dlatego dla każdego regulatora ustawiono wartość parametru $T_d = 0.5$:



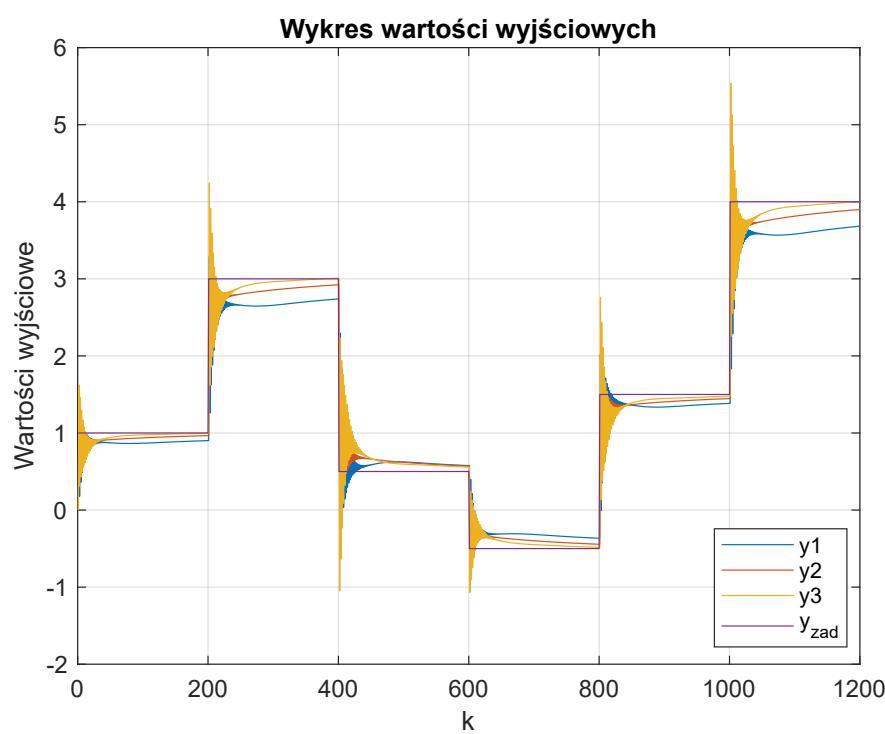
Rys. 1.8. Wyjścia procesu dla regulatorów PID: $K = 1$, $T_i = 80$, $T_d = 0.5$



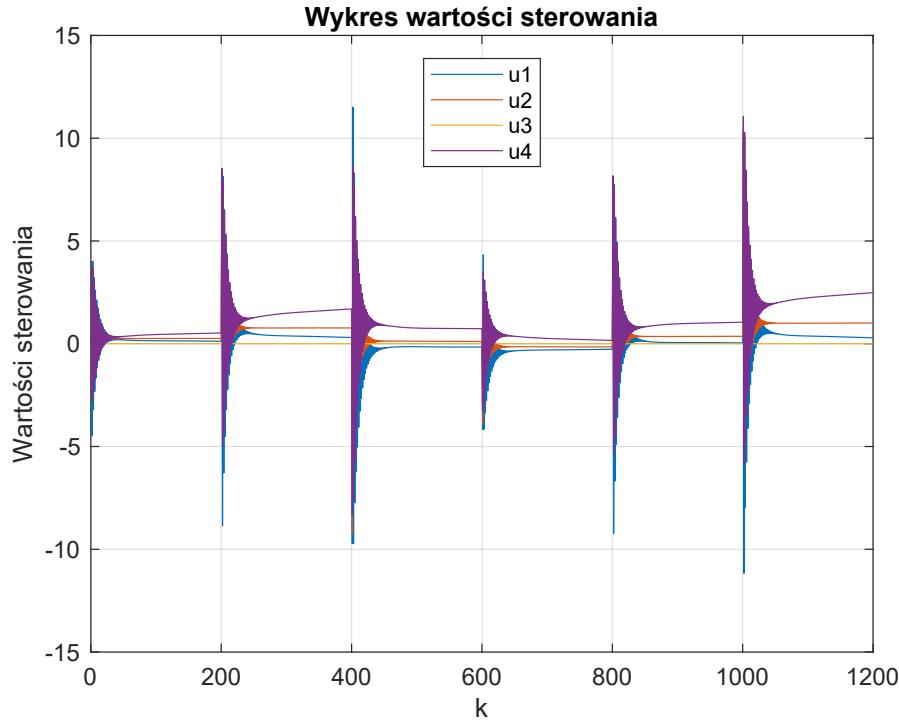
Rys. 1.9. Wejścia procesu dla regulatorów PID: $K = 1$, $T_i = 80$, $T_d = 0.5$

Regulacja po "włączeniu" każdego parametru nie jest zadowalająca, ale proces jest stabilny

- w znaczeniu takim, że żadne wartości któregoś wejścia nie rozjechały się do zbyt dużych wartości. Zatem w tym momencie rozpoczęto próbę doboru (zwiększenia) parametrów K - widać, że największy wzrost powinien nastąpić w przypadku regulatora pierwszego, najmniejszy w przypadku regulatora trzeciego. Jednak podczas eksperymentów zauważono, że dodanie różniczkowania wprowadziło sytuację, w której jakikolwiek wzrost parametrów K skutkuje ogromnymi oscylacjami sterowania, dla przykładu sytuacja dla $K = 2$ w każdym regulatorze:



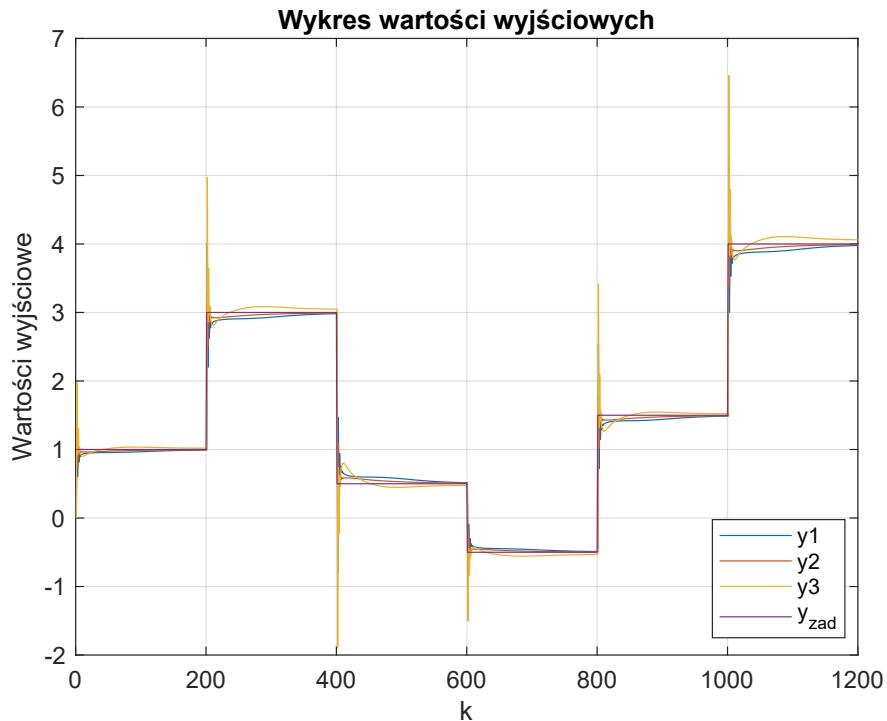
Rys. 1.10. Wyjścia procesu dla regulatorów PID: $K = 2$, $T_i = 80$, $T_d = 0.5$



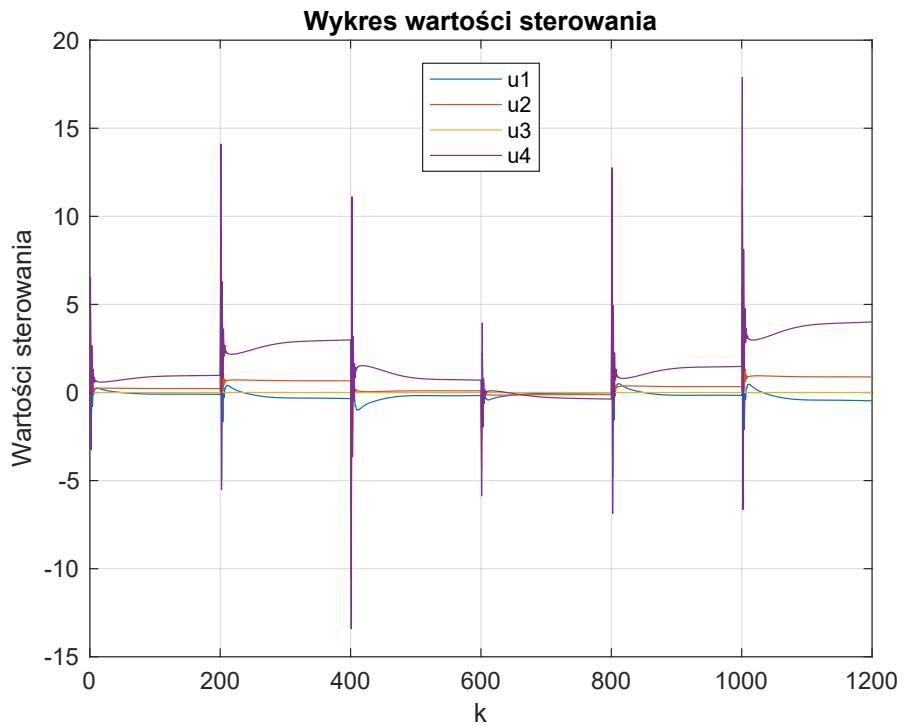
Rys. 1.11. Wejścia procesu dla regulatorów PID: $K = 2$, $T_i = 80$, $T_d = 0.5$

Z tego powodu, zdecydowano się na całkowite wyłączenie różniczkowania ($T_d = 0$ dla każdego regulatora). W taki sposób łatwiej będzie eksperymentalnie dobrąć odpowiednie nastawy regulatorów. Patrząc na sytuację przed włączeniem różniczkowania widać, że najgorzej regulowane jest wyjście y_1 i to w tym torze należałoby najbardziej zwiększyć wzmacnianie oraz całkowanie. Nieco lepiej regulator radzi sobie z wyjściem y_2 i tutaj zmiany będą mniejsze, a najmniejszy wzrost parametrów przewidywany jest dla toru regulacji wyjścia y_3 .

Po serii eksperymentów z doborem parametrów K oraz T_i zdecydowano się na wartości: $K = [6.5; 4.5; 2.5]$ oraz $T_i = [25; 45; 60]$ (są to po kolejni wartości nastaw dla kolejnych regulatorów lokalnych PI):



Rys. 1.12. Wyjścia procesu dla regulatorów PI z ostatecznymi nastawami

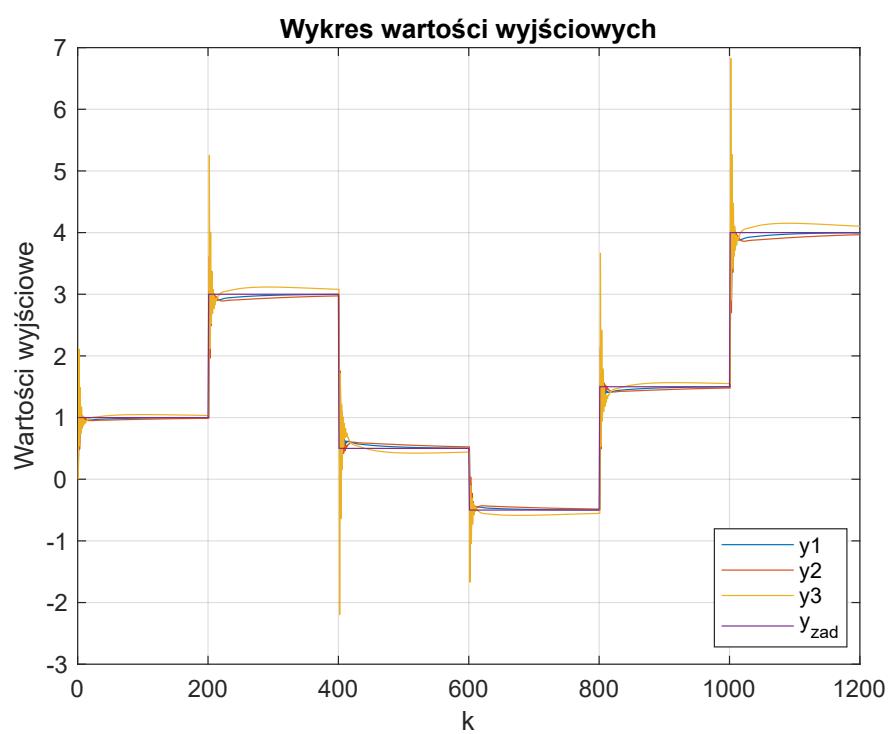


Rys. 1.13. Wejścia procesu dla regulatorów PI z ostatecznymi nastawami

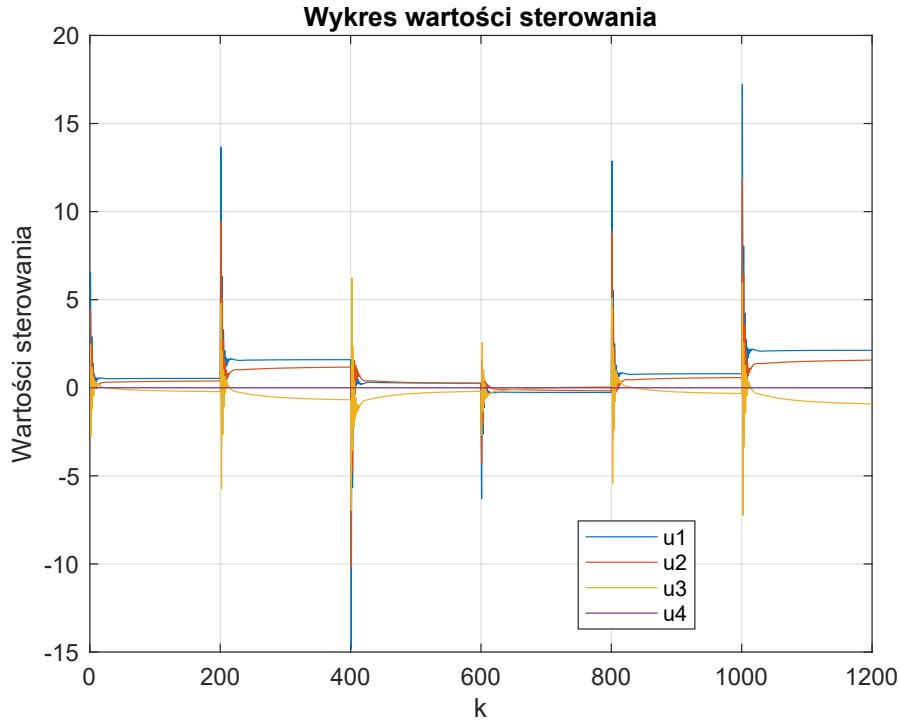
Pomimo wielu prób, nie udało się zredukować występującego, bardzo krótkiego, przeregulowania na torze wyjścia 3 przy jednoczesnej szybkiej i dokładnej regulacji w pozostałych torach. Źródło tego prawdopodobnie wywodzi się z faktu, że na wyjście 3 wszystkie wykorzystane wejścia wpływają stosunkowo mocno - to jest, zgodnie z otrzymanymi odpowiedziami skokowymi, widać, że w każdym torze wykorzystywanych sterowań, dla wyjścia 3 każde wejście ma stosunkowo

duże wzmocnienie. Z pozytywnych aspektów jednakże każde wyjście stabilizuje się szybko w pobliżu wartości zadanej. Nie występuje także uchyb ustalony, a w wyjściach 1 i 2 nie występuje przeregulowanie.

Z tak dobranymi nastawami sprawdzono działanie regulatorów w przypadku, gdy występują następujące zależności pomiędzy uchybami a wejściami: $e_1 \rightarrow u_1$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_3$, a wejście u_4 jest zerowane:



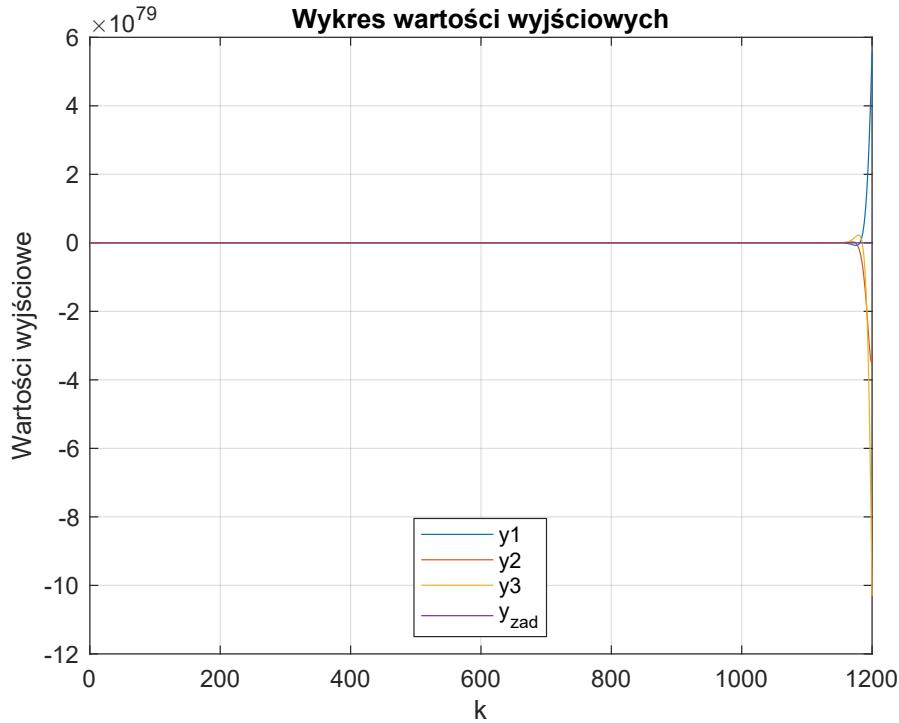
Rys. 1.14. Wyjścia procesu dla regulatorów PI z ostatecznymi nastawami i zmienioną konfiguracją



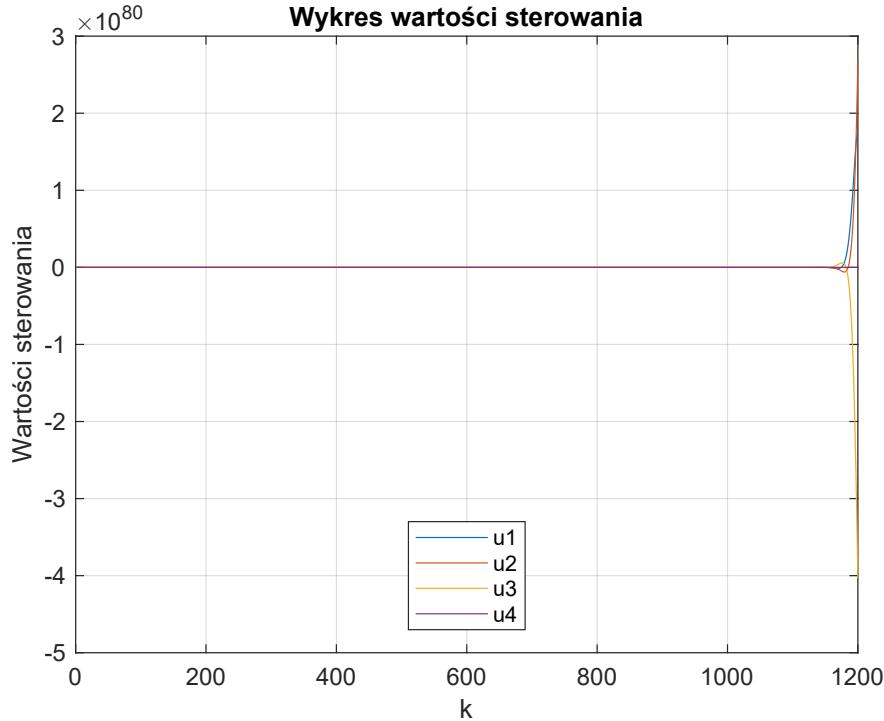
Rys. 1.15. Wejścia procesu dla regulatorów PI z ostatecznymi nastawami i zmienioną konfiguracją

Z uwagi na podobne wzmacniania w wykorzystanych tym razem torach - regulacja jest podobna do uzyskanej wyżej, jednak widać gorszą regulację dla wyjścia 3. Ponadto, w tym momencie wszystkie wejścia gwałtownie zmieniają się przy zmianie wartości zadanej, a nie tylko wejście u_4 , co nie jest pożądane.

Na koniec przetestowano działanie, w konfiguracji "losowej", zupełnie niezwiązanej z otrzymanymi odpowiedziami skokowymi: $e_1 \rightarrow u_3$, $e_2 \rightarrow u_1$ oraz $e_3 \rightarrow u_2$, wejście u_4 zerowane:



Rys. 1.16. Wyjścia procesu dla regulatorów PI z ostatecznymi nastawami i zmienioną konfiguracją



Rys. 1.17. Wejścia procesu dla regulatorów PI z ostatecznymi nastawami i zmienioną konfiguracją

Jak widać, tak dobrane nastawy całkowicie wypaczają działanie regulatorów. Wniosek z tego jest prosty - każda konfiguracja wymaga oddzielnego strojenia nastaw regulatorów, tym odmiennym, czym różne są odpowiedzi skokowe otrzymywane w wykorzystywanych torach po zmianie ich konfiguracji.

Poniższa tabela ukazuje dodatkowo otrzymywane błędy średniokwadratowe w każdym ukazywanym przebiegu:

Nr przebiegu	1	2	3	4	5	6	7
Błąd	1048.0	454.1	440.9	278.3	115.9	137.0	wypaczone działanie

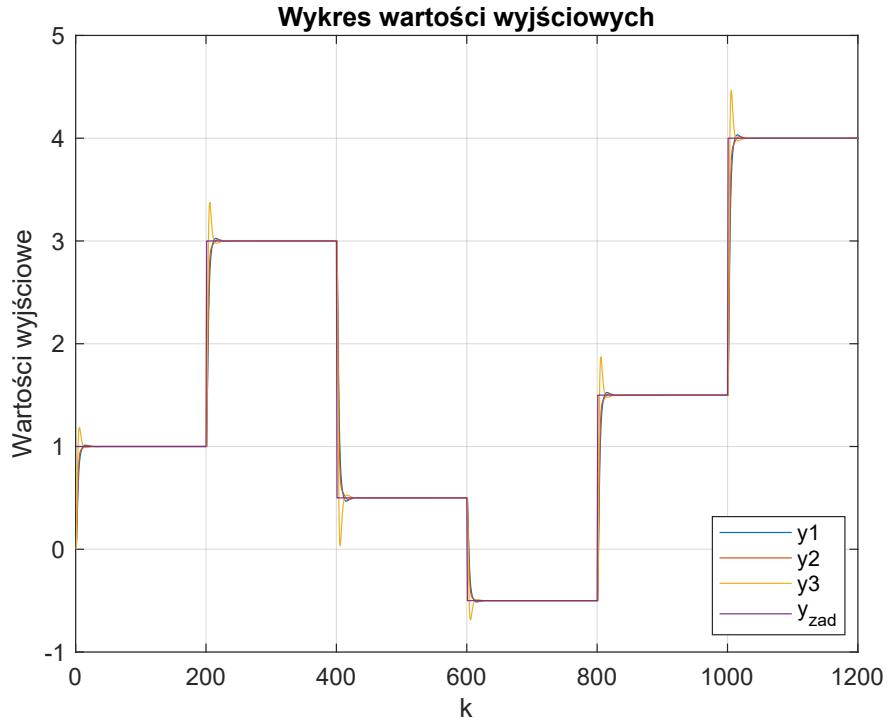
Tab. 1.10. Tabela wyników regulatorów PID

Jak widać, wraz ze strojeniem regulatora uzyskiwany błąd średniokwadratowy również maleł. Widać także, że zmieniona konfiguracja (przebieg 6) przyniosła trochę gorsze rezultaty od konfiguracji wyjściowej, co także było widać w przypadku analizy otrzymanych wykresów.

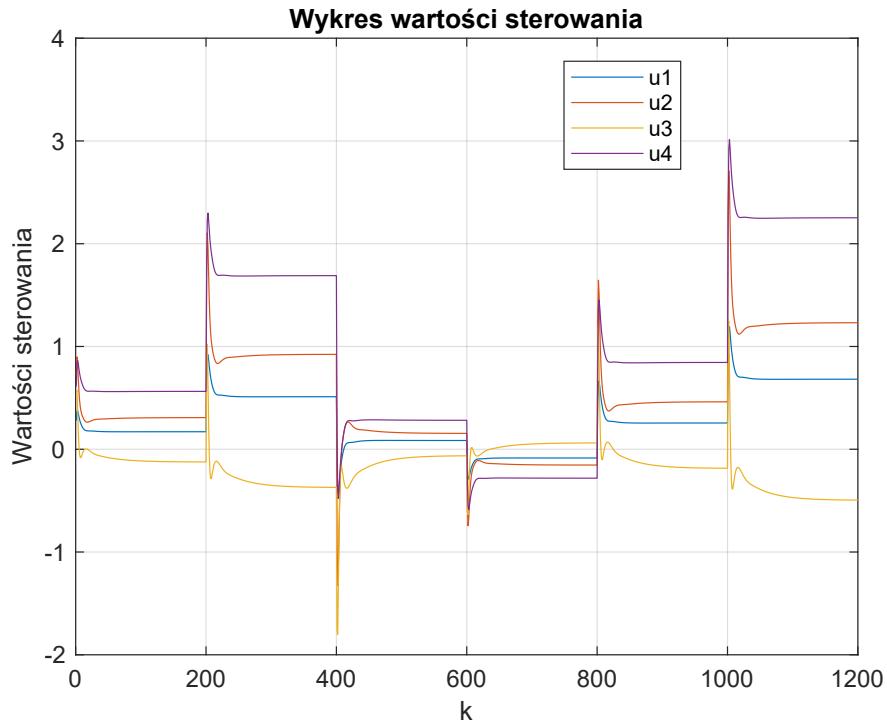
1.4.2. Algorytm DMC

Trajektoria wartości zadanej pozostała identyczna jak powyżej, w przypadku testowania regulatora PID. W trakcie wszystkich symulacji ukazanych w tej podsekcji sprawozdania przyjęto, że wektory Λ oraz Ψ składają się z samych jedynek. Ich wpływem i doborem zajęto się w następnej sekcji sprawozdania.

Proces strojenia rozpoczęto tradycyjnie od sytuacji, gdy $D = N = N_u = 200$ i otrzymano następujące wyniki:



Rys. 1.18. Wyjścia procesu dla regulatora DMC w przypadku, gdy $D = N = N_u = 200$

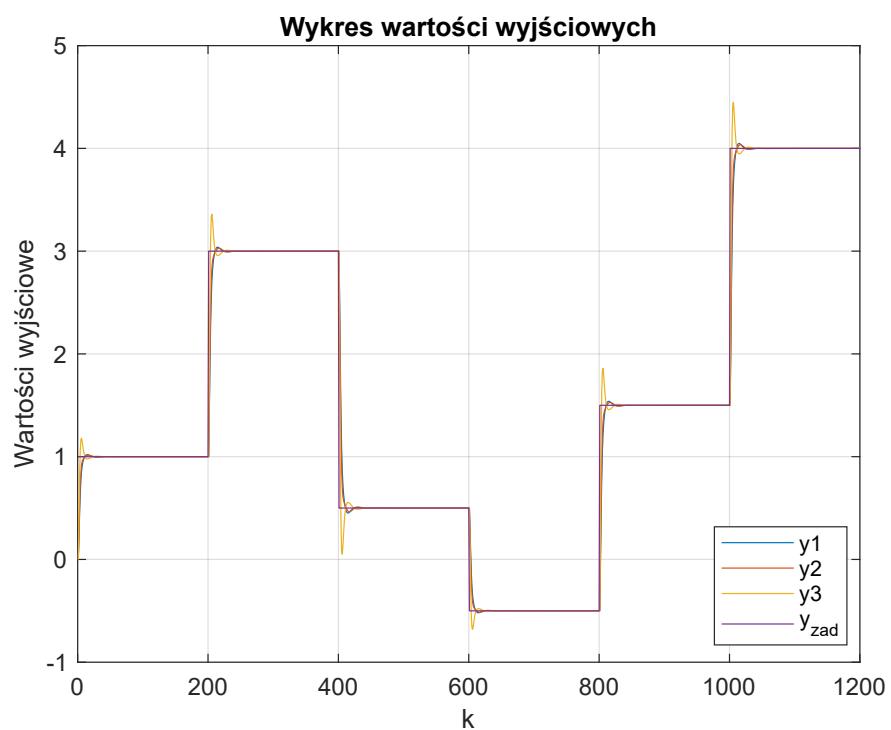


Rys. 1.19. Wejścia procesu dla regulatora DMC w przypadku, gdy $D = N = N_u = 200$

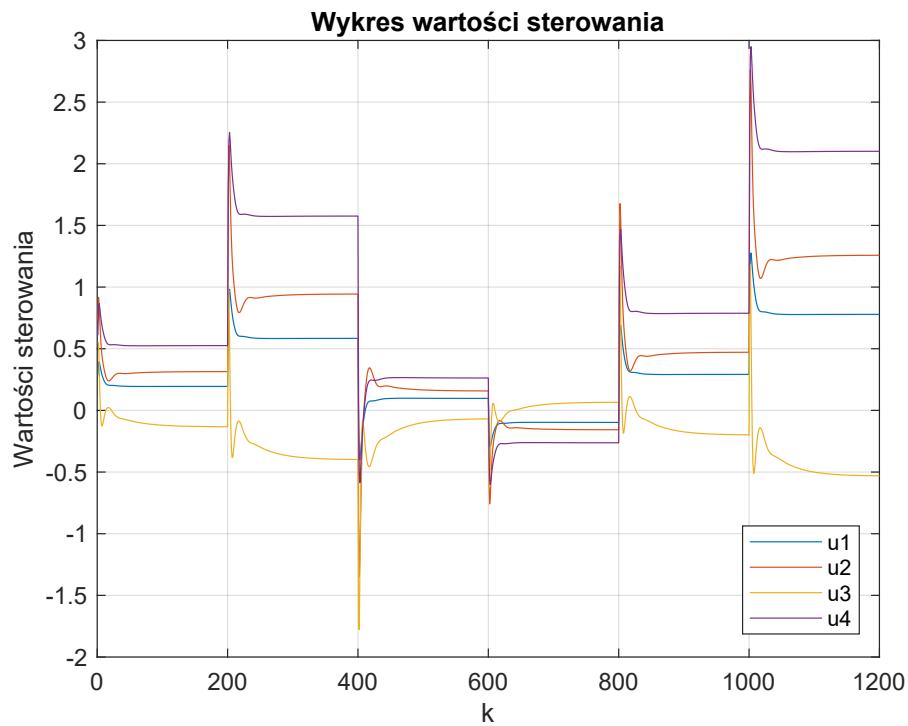
Regulator DMC już w początkowej konfiguracji działa świetnie. Ponownie, jedynie w przypadku wyjścia y_3 występuje przeregulowanie, jednak jest one mniejsze w porównaniu do tego, które generował regulator PID nawet w najlepszej konfiguracji, a samo przeregulowanie wynika zapewne z tego samego faktu, który był wymieniony w przypadku regulatora PID - na wyjście 3 w stosunkowo dużym stopniu oddziaływają wszystkie wejścia procesu. Czas ustalenia wszystkich

wyjść jest niewielki, a skoki sygnałów wejściowych nagłe i szybkie, ale nie osiągają przesadnie dużych wartości.

Kolejnym krokiem strojenia było zmniejszanie wartości parametru N tak długo, dopóki uzyskiwany błąd średniokwadratowy malał (ponownie błędy przedstawione zostały w tabeli na samym końcu podsekcji po ukazaniu wszystkich symulacji). Przyjęte zostało tutaj kryterium ilościowe, ponieważ pod względem jakościowym praktycznie niemożliwe było zauważenie jakichkolwiek zmian na lepsze - wyjściowe przebiegi już były bardzo dobrej jakości i jakakolwiek zmiana (powodująca poprawę) parametrów wizualnie nie przynosiła żadnych zauważalnych zmian. W tym momencie również, dalej została utrzymana zależność $N = N_u$. Ostateczną graniczną wartością okazało się $N = N_u = 8$:

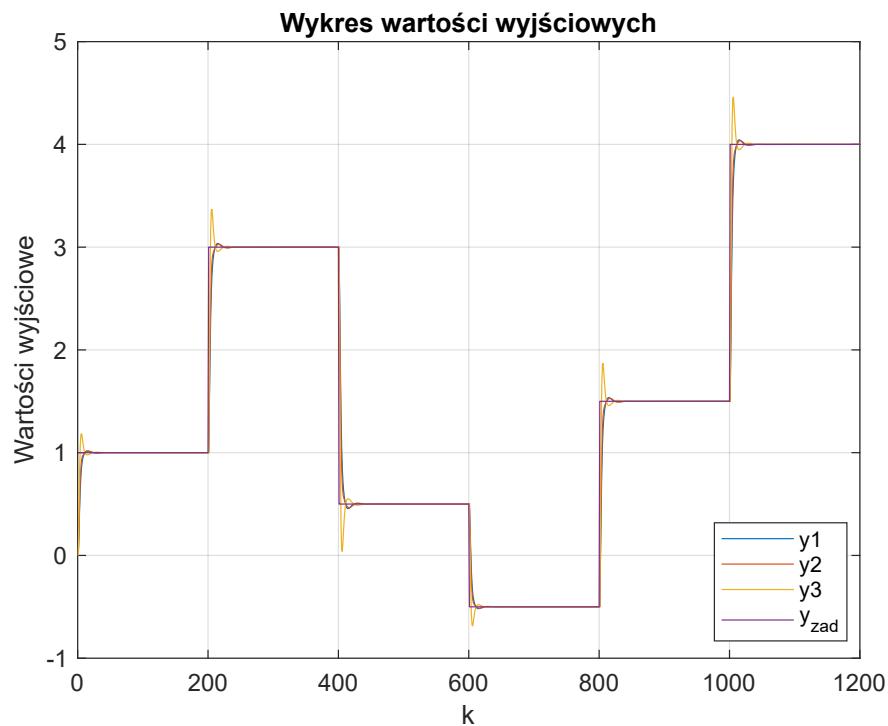


Rys. 1.20. Wyjścia procesu dla regulatora DMC w przypadku, gdy $D = 200$ i $N = N_u = 8$

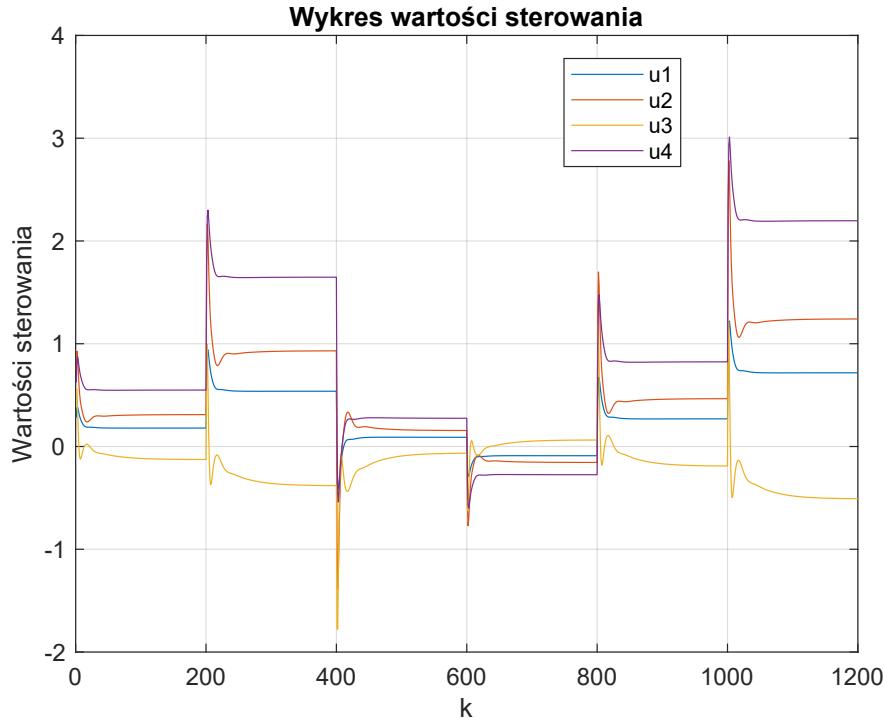


Rys. 1.21. Wejścia procesu dla regulatora DMC w przypadku, gdy $D = 200$ i $N = N_u = 8$

Na koniec takie same podejście zastosowano przy zmniejszaniu parametru N_u aż do wartości wynoszącej 4:



Rys. 1.22. Wyjścia procesu dla regulatora DMC w przypadku, gdy $D = 200$, $N = 8$ i $N_u = 4$



Rys. 1.23. Wejścia procesu dla regulatora DMC w przypadku, gdy $D = 200$, $N = 8$ i $N_u = 4$

Podobnie jak w przypadku algorytmu PID, poniżej znajduje się tabela przedstawiająca otrzymane błędy średniokwadratowe przeprowadzonych symulacji:

Nr przebiegu	1	2	3
Błąd	124.1	123.8	122.8

Tab. 1.11. Tabela wyników regulatorów DMC

Jak widać, nawet początkowe parametry regulatora DMC dawały już świetną regulację, zarówno pod względem jakościowym jak i ilościowym, chociaż ostateczny błąd średniokwadratowy okazał się niewiele większy od najniższego błędu uzyskanego przez regulator PID. Jednakże, sytuację tą może poprawić zmiana wartości wektorów Λ i Ψ . Mimo wszystko, regulator DMC nie potrzebował, w przeciwieństwie do regulatora PID, na dobrą sprawę żadnego strojenia, aby dawać bardzo dobre wyniki.

1.5. Dobór nastaw metodą optymalizacyjną

1.5.1. Algorytm PID

Do optymalizacji wektorów K , T_i oraz T_d skorzystaliśmy, tak jak przy okazji poprzednich projektów, z optymalizatora ga - algorytmu genetycznego. Przyjęto taki sam schemat powiązań wyjść i wejść: $e_1 \rightarrow u_4$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_1$, wejście u_3 jest zerowane oraz taką samą trajektorię wartości zadanej. Proces optymalizacji przeprowadziliśmy z następującymi parametrami:

```
% Parametry symulacji
kk = 1200;
Tp = 0.5;
yzad1 = [repmat(1, 1, 200), repmat(3, 1, 200), ...
repmat(0.5, 1, 200), repmat(-0.5, 1, 200), ...]
```

```

repmat(1.5, 1, 200), repmat(4, 1, 200)];
yzad2 = yzad1;
yzad3 = yzad1;
Y_zad = [yzad1; yzad2; yzad3];
U_nums = [4, 2, 1];

% Ograniczenia
K_min = [0, 0, 0];
K_max = [10, 10, 10];
Ti_min = [0, 0, 0];
Ti_max = [200, 200, 200];
Td_min = [0, 0, 0];
Td_max = [1.0, 1.0, 1.0];
lower_bounds = [K_min, Ti_min, Td_min];
upper_bounds = [K_max, Ti_max, Td_max];

% Parametry początkowe
K_init = [1.0, 1.0, 1.0];
Ti_init = [80.0, 80.0, 80.0];
Td_init = [0.0, 0.0, 0.0];
initial_params = [K_init, Ti_init, Td_init];

IntCon = [];
options = optimoptions('ga', 'Display', 'iter', ...
'PopulationSize', 100, 'MaxGenerations', 200, ...
'InitialPopulationMatrix', initial_params, ...
'UseParallel', false);

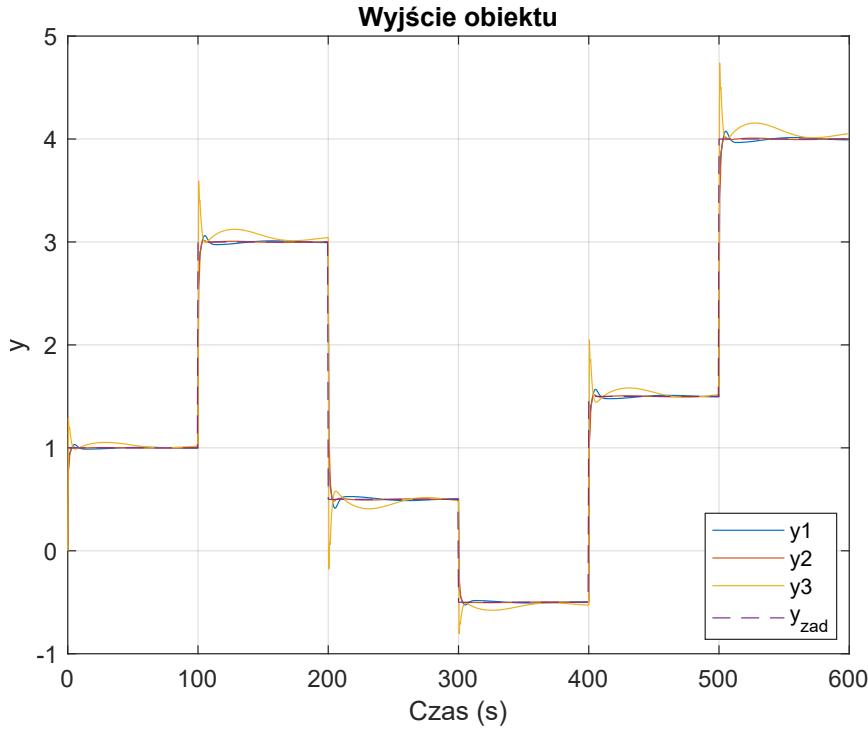
optimal_params = ga(@(params) ...
pid_cost_function(params, kk, Tp, Y_zad, U_nums), ...
9, [], [], [], lower_bounds, ...
upper_bounds, [], IntCon, options);

```

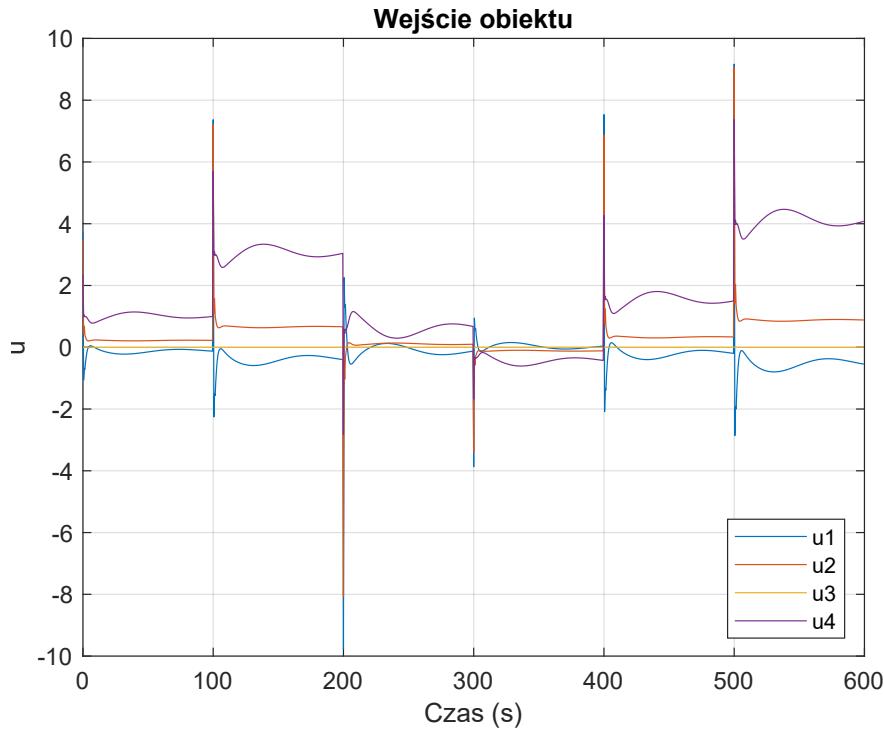
Tab. 1.12. Parametry poszczególnych pętli regulatorów PID

Parametr	Pętla $e_1 \rightarrow u_4$	Pętla $e_2 \rightarrow u_2$	Pętla $e_3 \rightarrow u_1$
K	1,97	2,92	3,74
T_i	1,39	9,76	114,37
T_d	0,00	0,09	0,00

Wektory parametrów będące wynikiem optymalizacji pokazane zostały w tabelce 1.12. Jak widać, optymalizator także wyzerował (z wyjątkiem drugiego, ale dalej bardzo bliskiego zeru) różniczkowanie. Błąd średniokwadratowy uzyskany z takimi parametrami wynosi jedynie 78,19. Przebiegi wyglądają następująco:



Rys. 1.24. Wyjścia procesu z optymalnymi parametrami



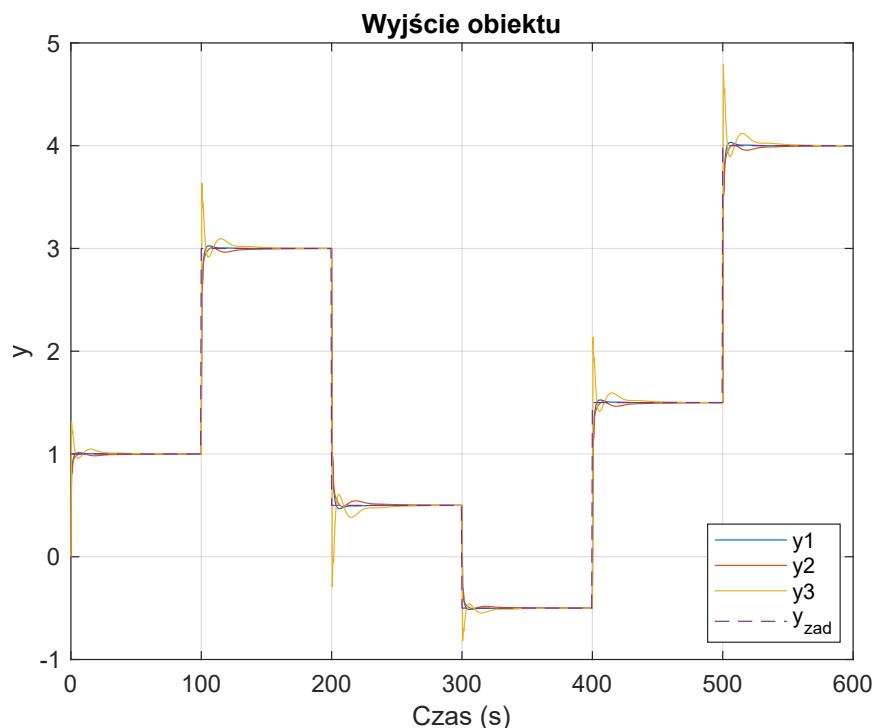
Rys. 1.25. Wejścia procesu z optymalnymi parametrami

Optymalizacja przyniosła najlepsze rezultaty, znacznie lepsze od wyników uzyskanych metodą eksperymentalną pod względem ilościowym. Jakościowo przebiegi wyjść także są zadowalające, najmniej dokładny regulator jest ponownie w torze wyjścia y_3 . Wejścia regulatora natomiast są poddawane nagłym i dużym skokom podczas zmiany wartości zadanej, co nie jest pożądanym zjawiskiem.

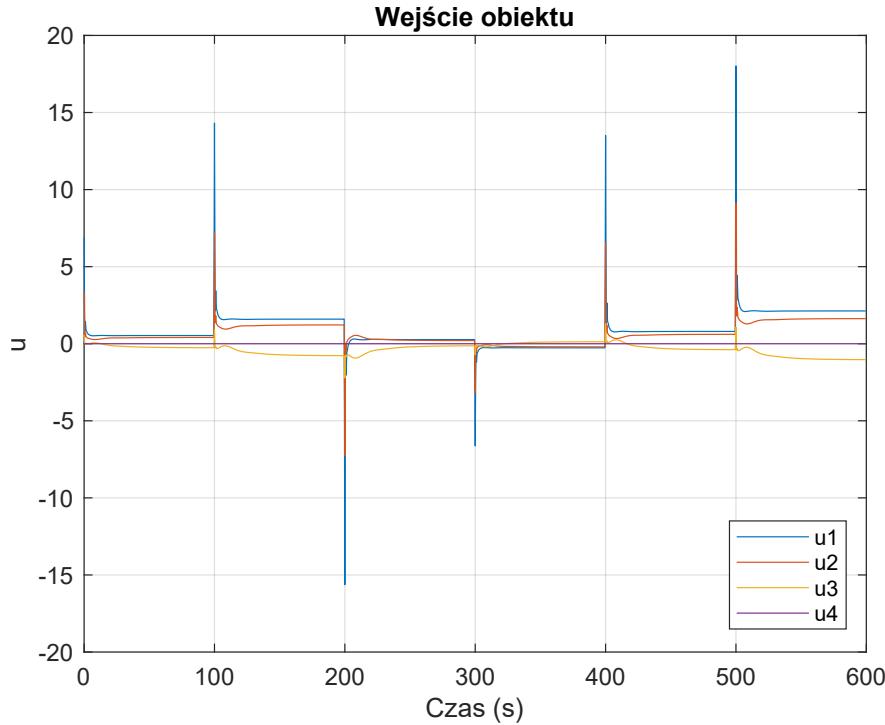
Postanowiono również optymalizację parametrów przy testowaniu innych konfiguracji wpływu uchybów na sygnał sterujący, $e_1 \rightarrow u_1$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_3$. Parametry zostały przedstawione w tabelce 1.13, a przebiegi na rysunkach 1.26 i 1.27. Błąd średniokwartatowy wyniósł 77,36, co jest pewną nieznaczną poprawą względem poprzedniej konfiguracji, jednak może to nie wynikać z samej konfiguracji co lepszej lub gorszej optymalizacji parametrów. Porównując przebiegi wyjścia dla tych dwóch konfiguracji najbardziej rzucający się w oczy jest przebieg wyjścia 3, gdzie dla pierwszej konfiguracji wybrzuszenie po osiągnięciu wartości zadanej trwa dość długo, natomiast dla konfiguracji alternatywnej oscylacje ustępują szybciej.

Tab. 1.13. Parametry poszczególnych pętli regulatorów PID w alternatywnej konfiguracji

Parametr	Pętla $e_1 \rightarrow u_1$	Pętla $e_2 \rightarrow u_2$	Pętla $e_3 \rightarrow u_3$
K	5,61	2,93	0,45
T_i	6,99	8,54	1,12
T_d	0,08	0,06	0,03



Rys. 1.26. Wyjścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_1$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_3$

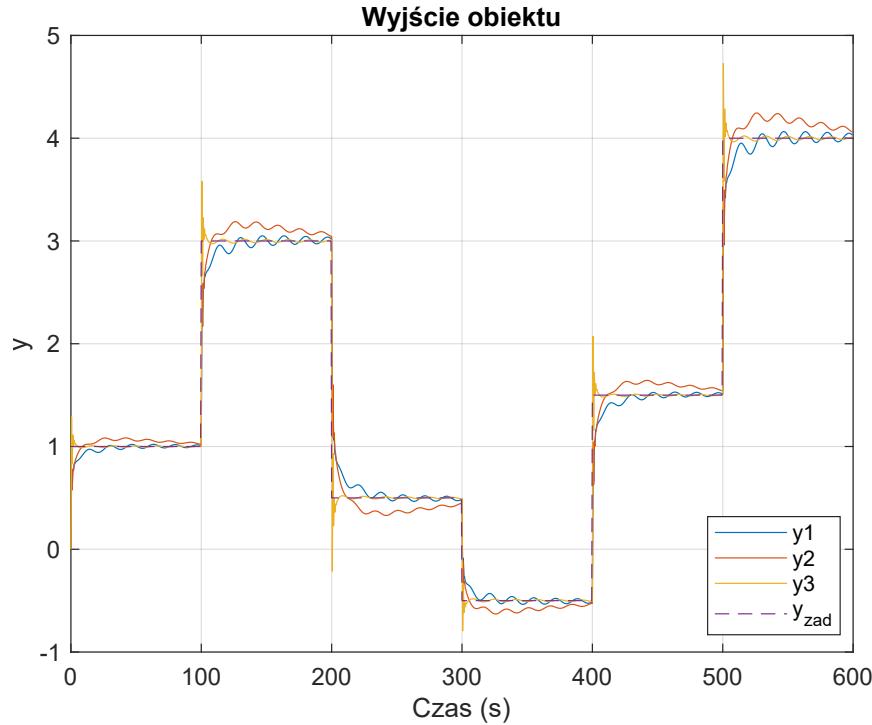


Rys. 1.27. Wejścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_1$, $e_2 \rightarrow u_2$ oraz $e_3 \rightarrow u_3$

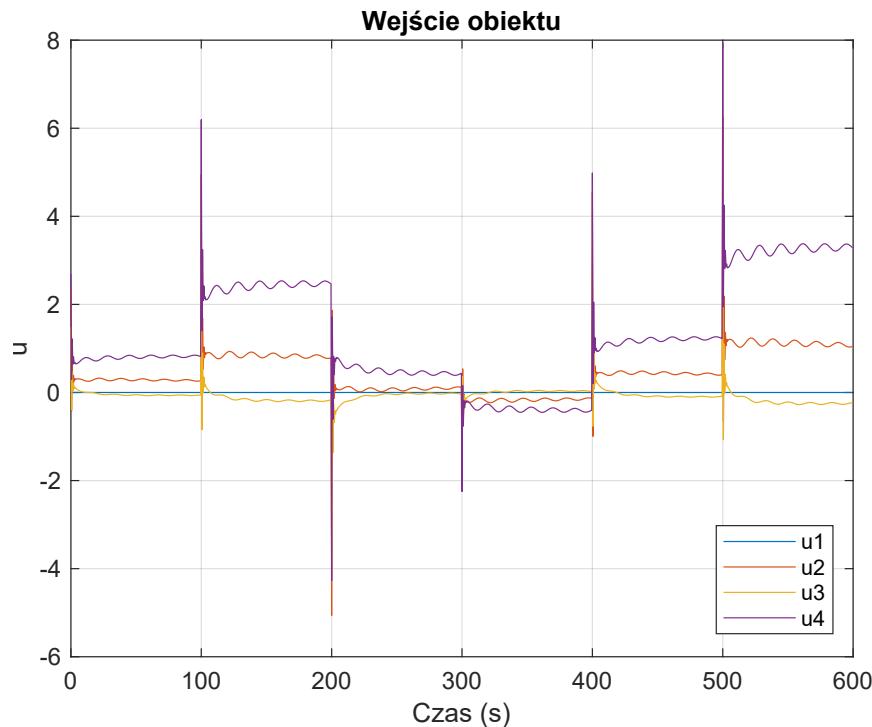
Kolejno sprawdzono konfigurację $e_1 \rightarrow u_2$, $e_2 \rightarrow u_3$ oraz $e_3 \rightarrow u_4$. Błąd średniokwadratowy przy takiej konfiguracji to 109, parametry przedstawione zostały w tabeli 1.14, a przebiegi na rysunkach 1.28 i 1.29. Widoczne są tu gasnące oscylacje zarówno w sygnałach wyjściowych jak i sterujących, a swoim kształtem przypominają jakby nachodziły na swojego rodzaju wyższą harmoniczną, co może być spowodowane małym okresem całkowania. Widać, że ta konfiguracja jest gorsza od poprzednio badanych, ale regulator wciąż jest w stanie być w pobliżu wartości zadanej.

Tab. 1.14. Parametry poszczególnych pętli regulatorów PID

Parametr	Pętla $e_1 \rightarrow u_2$	Pętla $e_2 \rightarrow u_3$	Pętla $e_3 \rightarrow u_4$
K	1,16	0,75	2,15
T_i	11,27	56,99	1,01
T_d	0,50	0,48	0,00



Rys. 1.28. Wyjścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_2$, $e_2 \rightarrow u_3$ oraz $e_3 \rightarrow u_4$



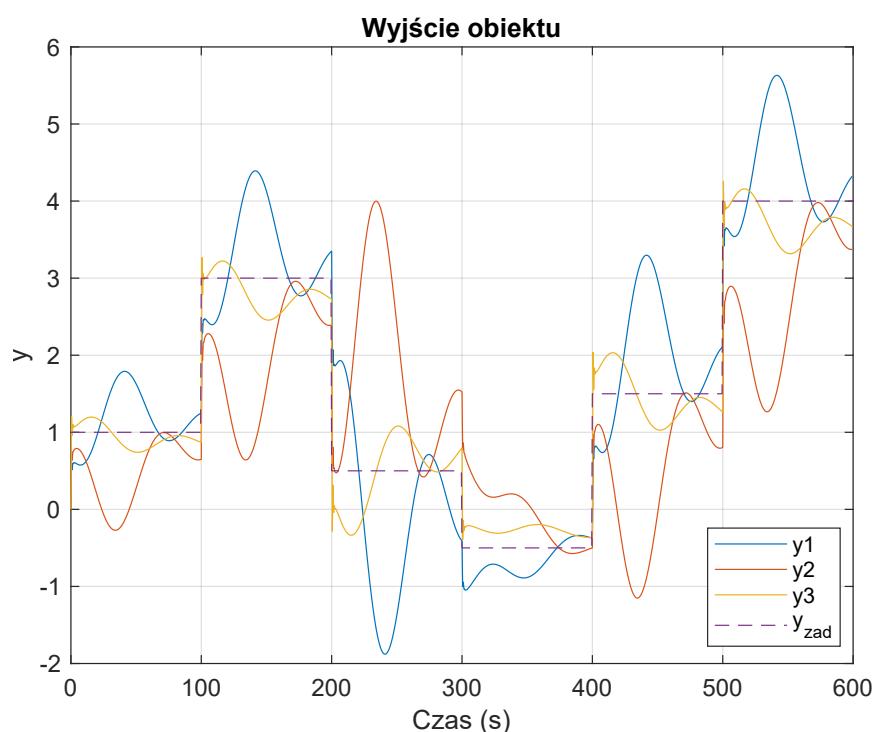
Rys. 1.29. Wejścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_2$, $e_2 \rightarrow u_3$ oraz $e_3 \rightarrow u_4$

Postanowiono na koniec sprawdzić działanie optymalizacji, kiedy wybrano widocznie niekorzystną konfigurację $e_1 \rightarrow u_3$, $e_2 \rightarrow u_4$ oraz $e_3 \rightarrow u_2$. Błąd średnioadratowy to 2885. Parametry w tabeli 1.15, a przebiegi na rysunkach 1.30 i 1.31. Optymalizator mimo kilku prób strojenia nie

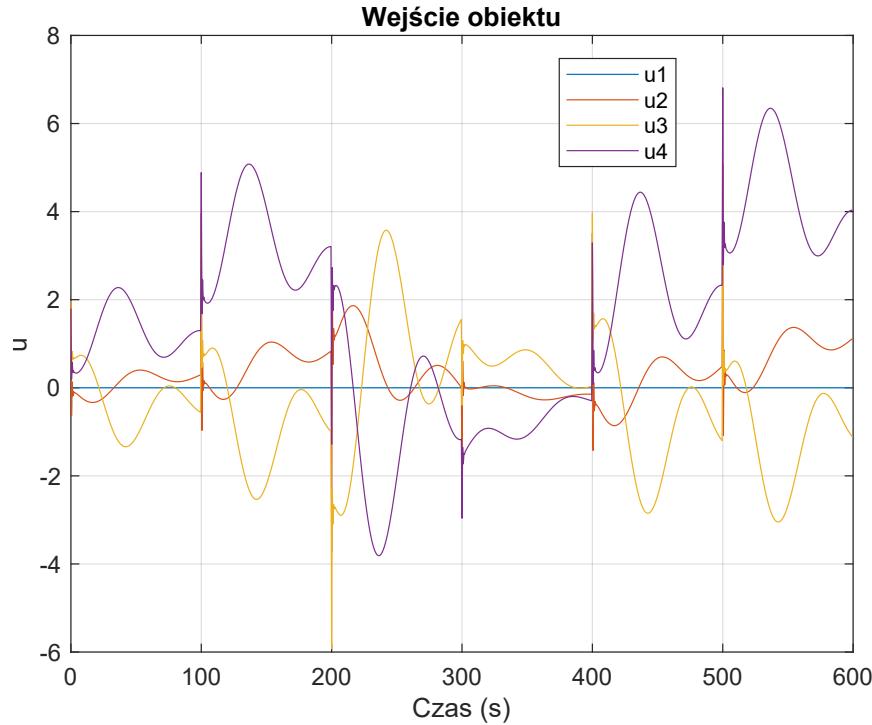
był w stanie znaleźć parametrów dla danej konfiguracji zapewniających akceptowalne przebiegi. Ta konkretna konfiguracja cechuje się małymi wzmacnieniami

Tab. 1.15. Parametry poszczególnych pętli regulatorów PID

Parametr	Pętla $e_1 \rightarrow u_3$	Pętla $e_2 \rightarrow u_4$	Pętla $e_3 \rightarrow u_2$
K	1,65	1,50	1,55
T_i	165,93	101,30	92,67
T_d	0,09	0,10	0,09



Rys. 1.30. Wyjścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_3$, $e_2 \rightarrow u_4$ oraz $e_3 \rightarrow u_2$



Rys. 1.31. Wejścia procesu z optymalnymi parametrami
 $e_1 \rightarrow u_3$, $e_2 \rightarrow u_4$ oraz $e_3 \rightarrow u_2$

1.5.2. Algorytm DMC

Ponownie wykorzystaliśmy algorytm genetyczny oraz taką samą trajektorię zadaną. Przyjęto $D = N = N_u = 200$, parametry stałe podczas optymalizacji. Parametry optymalizacji:

```
% Parametry symulacji
kk = 1200;
Tp = 0.5;
yzad1 = [repmat(1, 1, 200), repmat(3, 1, 200), ...
          repmat(0.5, 1, 200), repmat(-0.5, 1, 200), ...
          repmat(1.5, 1, 200), repmat(4, 1, 200)];
yzad2 = yzad1;
yzad3 = yzad1;
Y_zad = [yzad1; yzad2; yzad3];
D = 200;
N = 200;
Nu = 200;

% Ograniczenia
lambda_min = [0.1, 0.1, 0.1, 0.1];
lambda_max = [100, 100, 100, 100];
psi_min = [0, 0, 0];
psi_max = [100, 100, 100];
lower_bounds = [lambda_min, psi_min];
upper_bounds = [lambda_max, psi_max];

% Parametry początkowe
lambda_init = [1.0, 1.0, 1.0, 1.0];
```

```
psi_init = [1.0, 1.0, 1.0];
initial_params = [lambda_init, psi_init];

PopulationSize = 50;
InitialPopulationMatrix=repmat(initial_params, PopulationSize, 1);

IntCon = [];
options = optimoptions('ga', 'Display', 'iter', ...
'PopulationSize', PopulationSize, ...
'MaxGenerations', 20, 'InitialPopulationMatrix', ...
InitialPopulationMatrix, 'UseParallel', false);

optimal_params = ga(@(params) dmc_cost_function(params, kk, ...
Y_zad, D, N, Nu), ...
7, [], [], [], lower_bounds, ...
upper_bounds, [], IntCon, options);
```

Wektor λ został ograniczony od dołu chociaż do wartości 0.1, aby zapewnić chociaż minimalne kary za przyrosty sterowań, w przypadku optymalizacji pozwalającej na wartości 0, sygnał sterujący był poddawany gwałtownym skokom a symulacja wydawała się błędna, dawała nienaturalne wyniki. Ostatecznie w procesie optymalizacji ostateczne parametry wynoszą: $\lambda = [1; 0.5; 1; 0.5]$ oraz $\psi = [4; 4; 6]$. Od razu warto zaznaczyć, że nie było stosowane ograniczenie wartości do liczb całkowitych (parametr IntCon optymalizatora jest pustą listą), tak "równe" wyniki optymalizacji nie zostały w żaden sposób spreparowane:

```
>> p5_DMC_MIMO

Single objective optimization:
7 Variable(s)

Options:
CreationFcn:          @gacreationuniform
CrossoverFcn:        @crossoverscattered
SelectionFcn:         @selectionstochunif
MutationFcn:          @mutationadaptfeasible

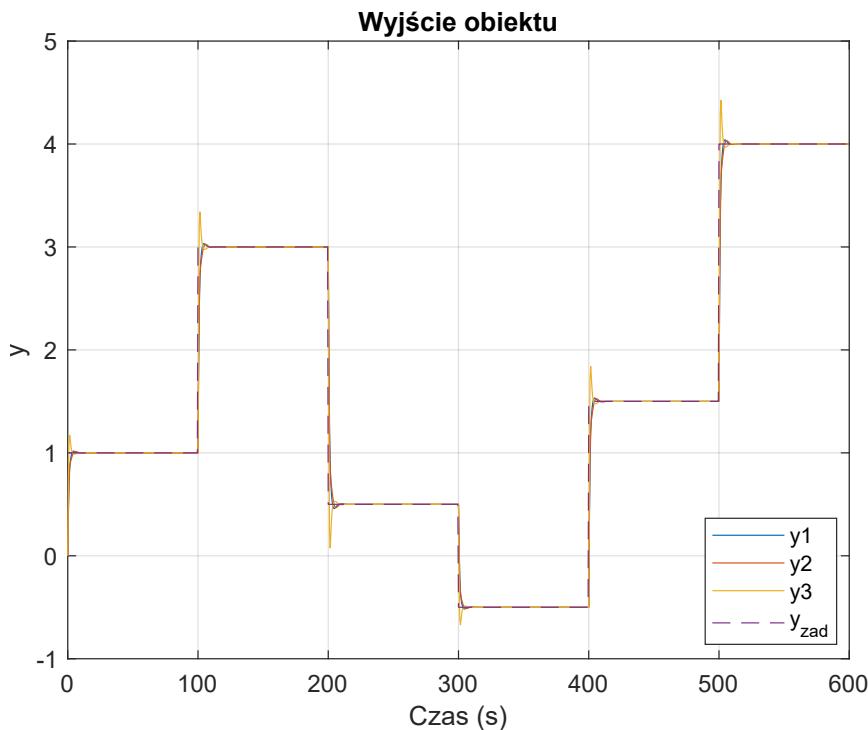


| Generation | Func-count | Best f(x) | Mean f(x) |
|------------|------------|-----------|-----------|
| 1          | 100        | 109.7     | 175.2     |
| 2          | 147        | 109.2     | 189.6     |
| 3          | 194        | 109.2     | 219.6     |
| 4          | 241        | 105.9     | 196       |
| 5          | 288        | 104.7     | 162.5     |
| 6          | 335        | 103.5     | 110.6     |
| 7          | 382        | 102       | 165       |
| 8          | 429        | 100.2     | 106.2     |
| 9          | 476        | 98.46     | 104.5     |
| 10         | 523        | 98.46     | 103.6     |
| 11         | 570        | 97.72     | 102.2     |
| 12         | 617        | 97        | 101.3     |
| 13         | 664        | 95.33     | 100.2     |
| 14         | 711        | 94.88     | 99.14     |
| 15         | 758        | 93.42     | 97.8      |
| 16         | 805        | 93.42     | 97.36     |
| 17         | 852        | 90.33     | 96.35     |
| 18         | 899        | 90.33     | 95.55     |
| 19         | 946        | 89.36     | 94.95     |
| 20         | 993        | 89.36     | 93.86     |

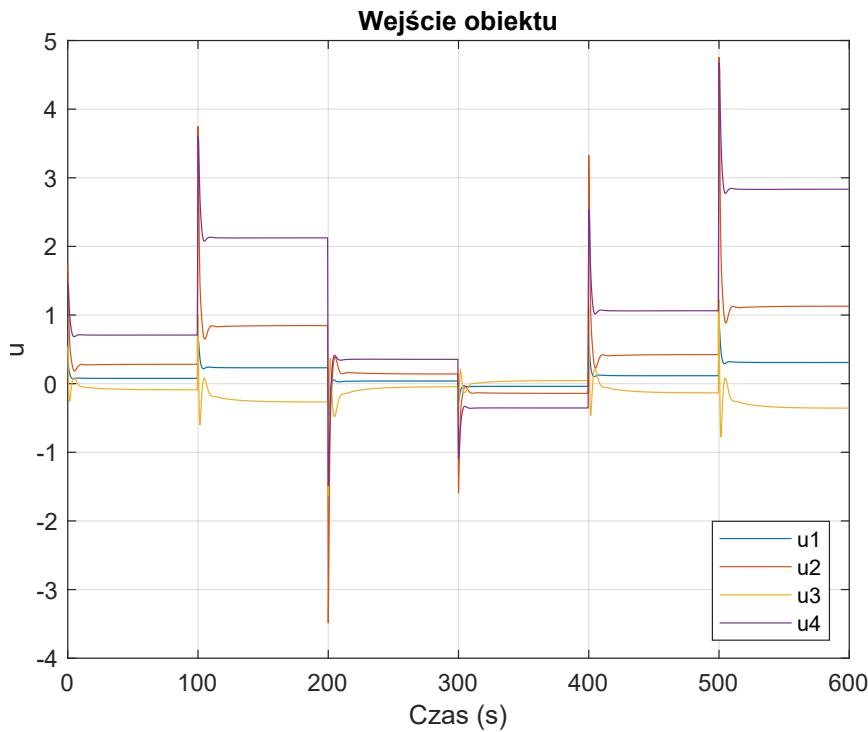

Optimization terminated: maximum number of generations
Optimized DMC parameters:
lambda: 1          0.5          1          0.5
psi: 4  4  6
>>
```

Rys. 1.32. Proces optymalizacji

A tak prezentują się przebiegi z optymalnymi parametrami:



Rys. 1.33. Wyjścia procesu z optymalnymi parametrami



Rys. 1.34. Wejścia procesu z optymalnymi parametrami

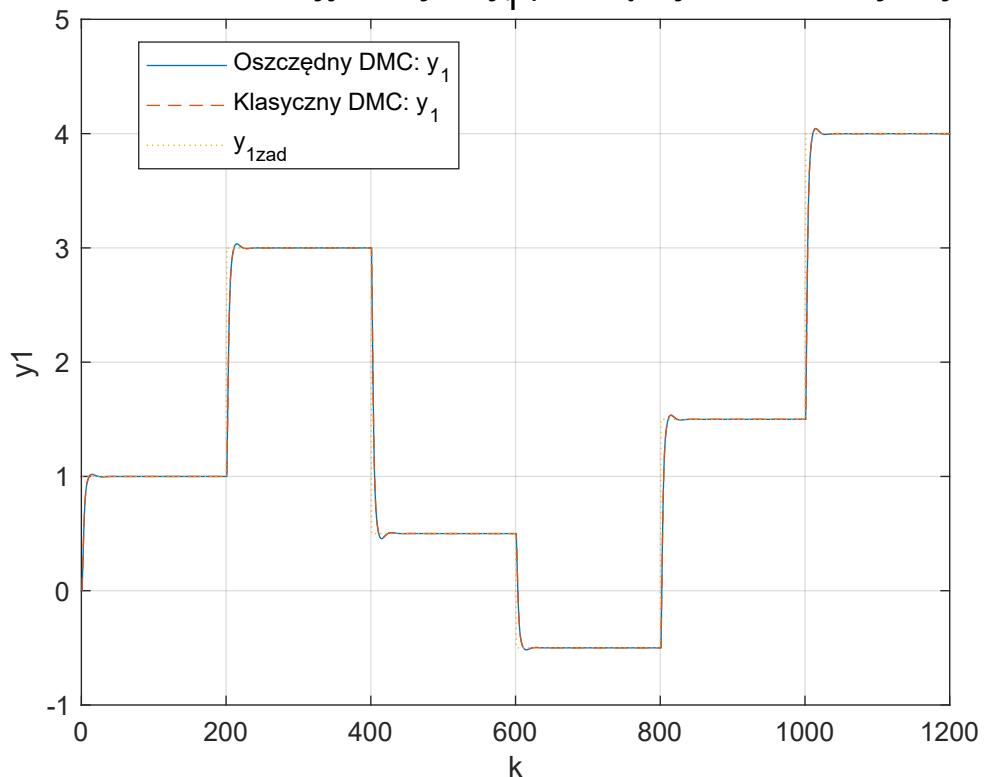
Ostateczny błąd średniokwadratowy w tym przypadku wyniósł 89,36. Pod względem ilościowym taki regulator DMC jest zdecydowanie lepszy od poprzednio opisanych. Słabiej jednak w tym kryterium wypada względem regulatora PID. Pod względem jakościowym natomiast regulator DMC daje lepsze rezultaty od regualatora PID - przeregulowanie i czas ustalenia się wyjścia

y_3 zdaje się być mniejszy oraz sygnały sterujące, chociaż dalej podlegają nagłym skokom, to ich przebiegi są mniej gwałtowne niż w przypadku regulatora PID.

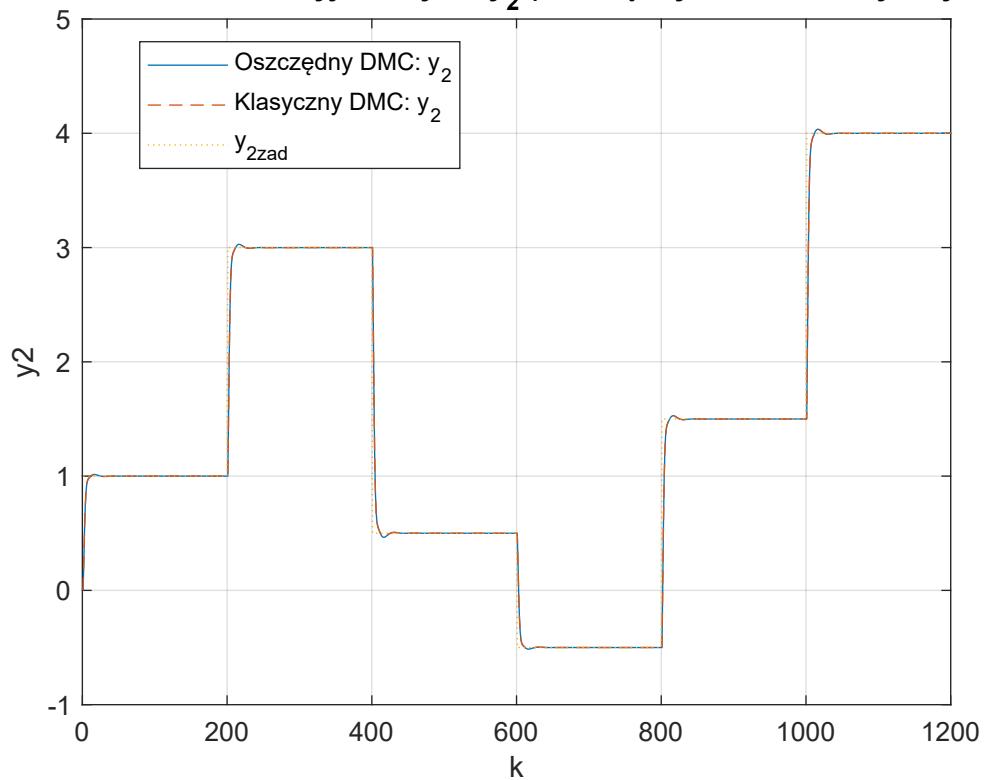
1.6. Porównanie regulatorów DMC

Zaimplementowano 2 wersje wielowymiarowego regulatora DMC, klasyczną oraz oszczędną. Do porównania użyto parametrów wyznaczonych metodą eksperymentalną, tzn $D = 200$, $N = 8$, $N_u = 4$ oraz wektorów λ i ψ składających się z samych jedynek. Wykorzystano trajektorie wartości zadanych wyjść używaną do strojenia regulatorów. Oczekiwano, że otrzymane przebiegi wartości wyjściowych dla obydwu wersji DMC, będą się ze sobą pokrywały oraz błędy kwadratowe obliczone dla całych przebiegów będą sobie równe, ponieważ są to wersje analityczne. Błąd dla oszczędnego regulatora DMC wyniósł 122,7593, a dla klasycznego DMC 122,7593, co oznacza, że błędy są sobie równe i sugeruje, że przebiegi regulatorów są identyczne, jednakże dla upewnienia się co do identyczności przebiegów wartości wyjściowych oraz sterowań przedstawiono je na wykresach od 1.35 do 1.41. Jak można na nich zauważyć przebiegi się pokrywają dla wszystkich wyjść oraz sterowań co potwierdza oczekiwane działanie obu regulatorów, a dokładniej, że działają tak samo co do wartości wyjść procesu i sterowań, ponieważ są to implementacje analityczne tego samego regulatora, które rozróżnia głównie złożoność obliczeniowa i ostateczny czas wykonywania.

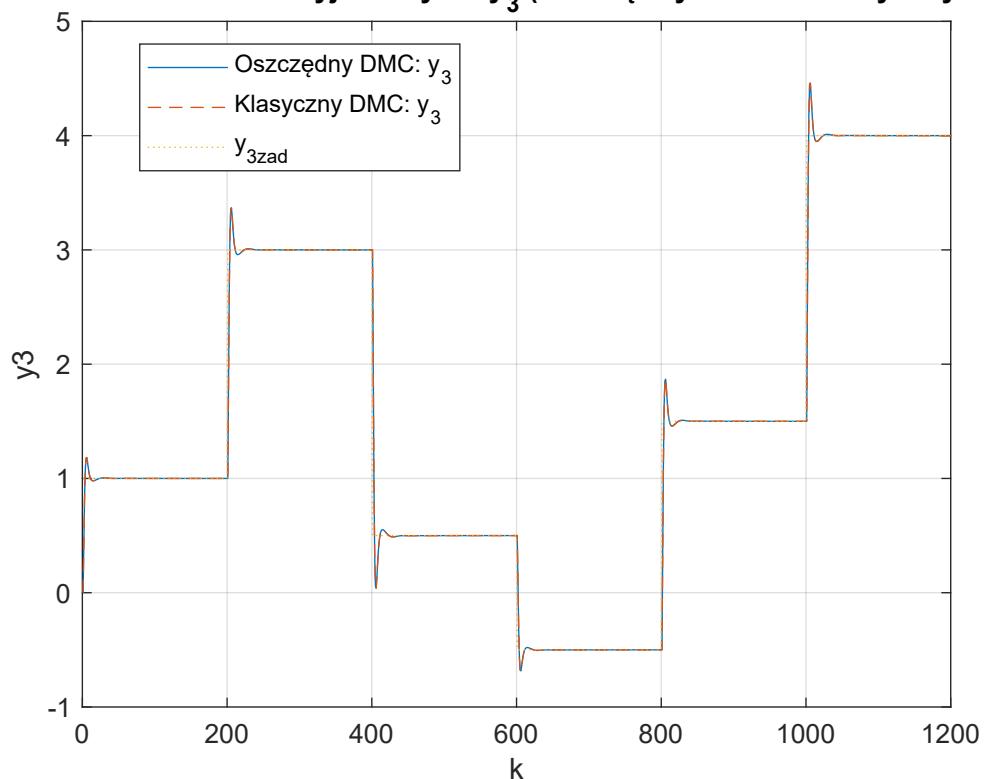
Porównanie wartości wyjściowych: y_1 (Oszczędny DMC vs Klasyczny DMC)



Rys. 1.35. Porównanie wartości wyjścia 1 procesu dla regulatorów DMC

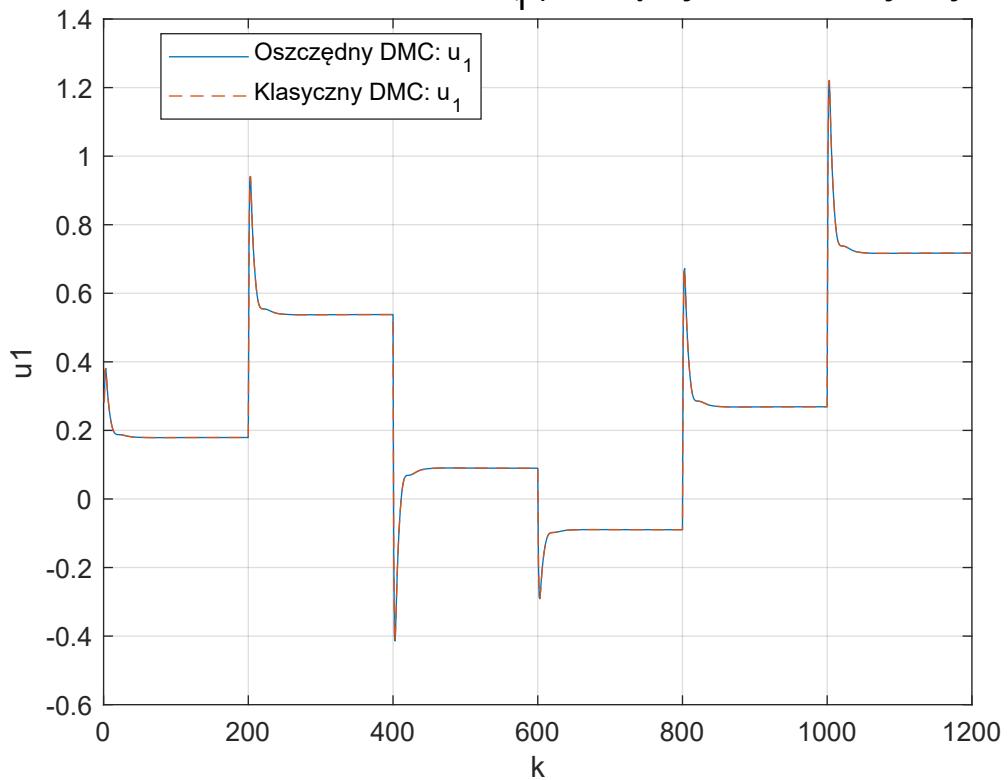
Porównanie wartości wyjściowych: y_2 (Oszczędny DMC vs Klasyczny DMC)

Rys. 1.36. Porównanie wartości wyjścia 2 procesu dla regulatorów DMC

Porównanie wartości wyjściowych: y_3 (Oszczędny DMC vs Klasyczny DMC)

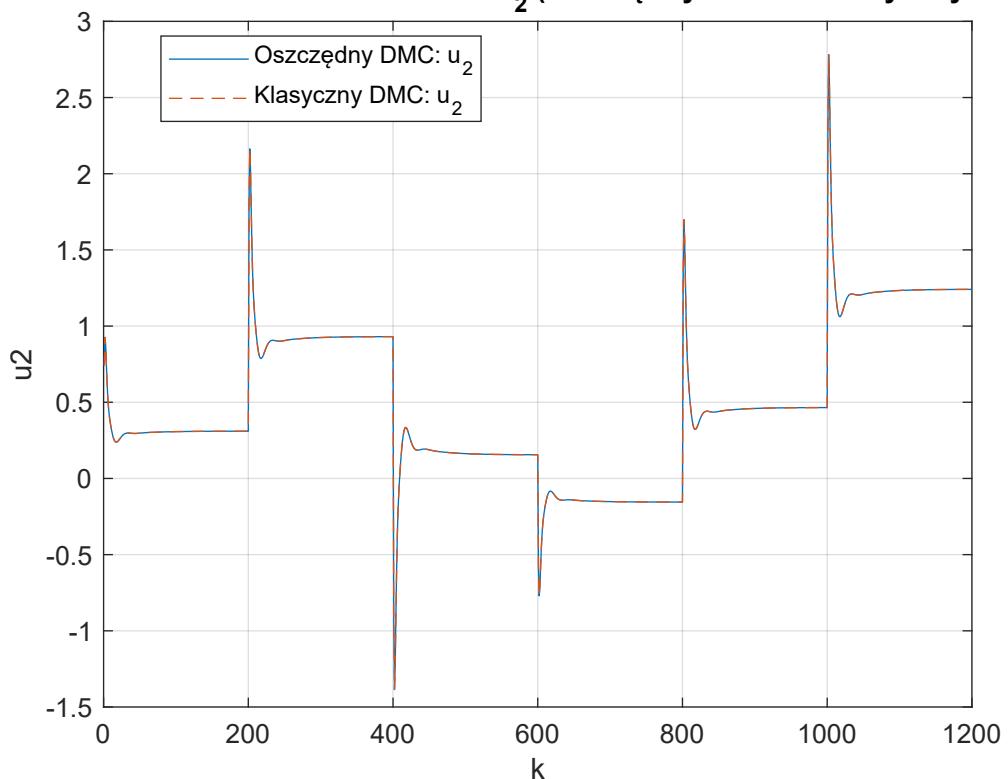
Rys. 1.37. Porównanie wartości wyjścia 2 procesu dla regulatorów DMC

Porównanie wartości sterowania: u_1 (Oszczędny DMC vs Klasyczny DMC)

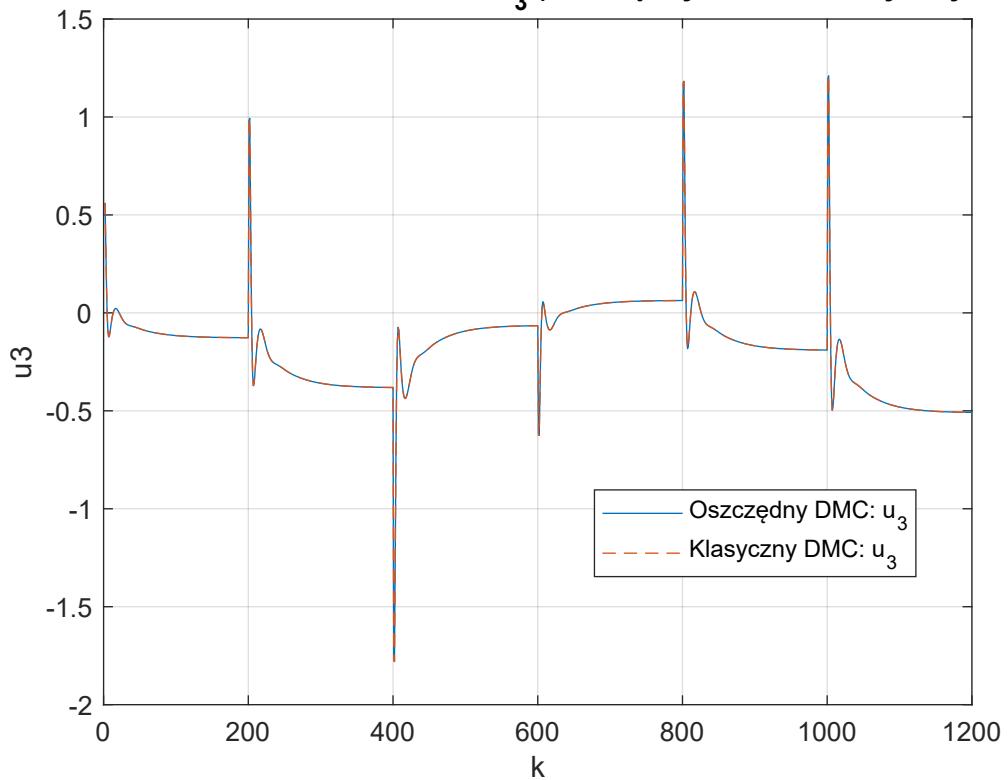


Rys. 1.38. Porównanie wartości sterowania 1 procesu dla regulatorów DMC

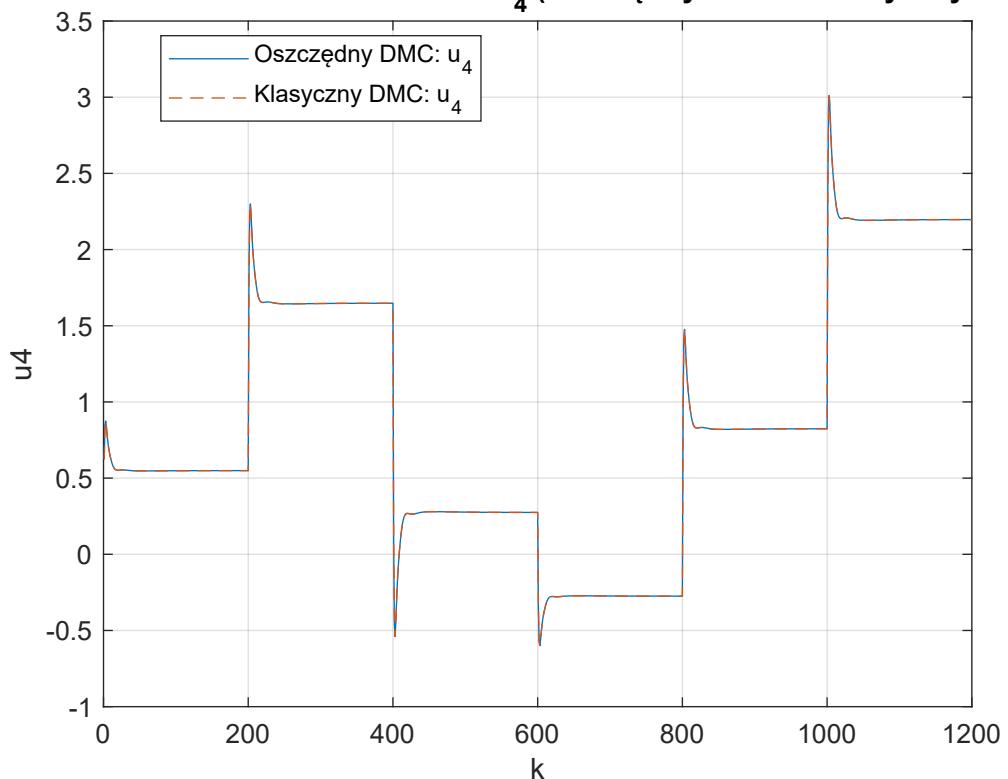
Porównanie wartości sterowania: u_2 (Oszczędny DMC vs Klasyczny DMC)



Rys. 1.39. Porównanie wartości sterowania 2 procesu dla regulatorów DMC

Porównanie wartości sterowania: u_3 (Oszczędny DMC vs Klasyczny DMC)


Rys. 1.40. Porównanie wartości sterowania 1 procesu dla regulatorów DMC

Porównanie wartości sterowania: u_4 (Oszczędny DMC vs Klasyczny DMC)


Rys. 1.41. Porównanie wartości sterowania 4 procesu dla regulatorów DMC

2. Laboratorium - stanowisko grzewczo-chłodzące

2.1. Komunikacja ze stanowiskiem i organizacja projektu w GxWorks3

Na podstawie analizy otrzymanego szablonu oczytano rejesty wykorzystywane do komunikacji ze stanowiskiem grzewczo-chłodzącym. Rejestry D100–D106 przechowują pomiary temperatury oraz prądu i napięcia, natomiast z rejestrów D110–D115 wysyłane są sterowania do wentylatorów i elementów grzewczych. Wykorzystywane w tym bloku laboratoryjnym elementy i ich rejesty przedstawiono w tabeli 2.1, dodatkowo rejesty przypisano do zmiennych, aby wygodnie korzystać z nich przy pisaniu kodu i debugowaniu. Prefiks "w" oznacza, że zmienna jest typu Word [Signed] i przyjęto, że służy wyłącznie do komunikacji. W pozostałej części kodu (tj. do obliczeń i wysyłania danych do MATLAB przy pomocy socketa) używano zmiennych FLOAT [Single Precision].

Tab. 2.1. Rejestry wykorzystywane do komunikacji z poszczególnymi elementami

Element	Rejestr	Nazwa zmiennej
T1	D100	wT1
T3	D102	wT3
W1	D110	ww1
W2	D111	ww2
G1	D114	wG1
G2	D115	wG2

Stworzono strukturę **struct_Proces**, aby zapewnić czytelność kodu i łatwiejsze jego rozszerzanie. Jej skład zawartość przedstawiono w tabeli 2.1. Stworzono dwie zmienne o podanej strukturze - Proces1 odpowiadająca za G1 i T1 oraz Proces2 odpowiadająca za G2 i T3.

	Label Name	Data Type	Comment
1	y_k	FLOAT [Single Precision]	Temperatura zmierzona w chwili k
2	u_k	FLOAT [Single Precision]	Moc grzałki w chwili k
3	u_km1	FLOAT [Single Precision]	Moc grzałki w chwili poprzedniej
4	y_zad	FLOAT [Single Precision]	Temperatura zadana
5	y_pp	FLOAT [Single Precision]	Temperatura w punkcie pracy

Rys. 2.1. Struktura struct_Proces

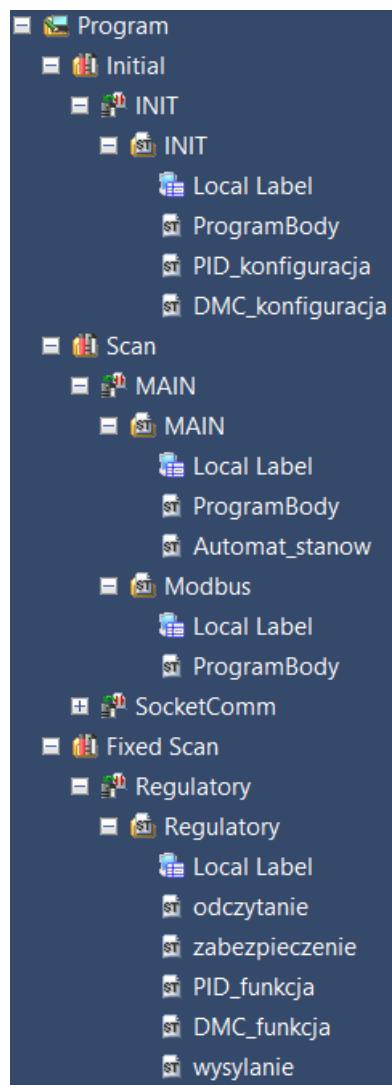
Drzewo programu projektu przedstawiono na rys. 2.2. Postanowiono pokrótko omówić najważniejsze jego części:

- **INIT**
 - **ProgramBody** – odpowiada za konfigurację komunikacji Modbus w początkowym skale.
 - **PID_konfiguracja** oraz **DMC_konfiguracja** – zostaną omówione w dalszych częściach.
- **MAIN**
 - **ProgramBody** – służy do opóźnienia startu komunikacji po Modbus oraz włącza przerwania.
 - **Automat_stanow** – zarządza pracą sterownika w zależności od wybranego trybu sterowania:
 - **0** – ustalenie punktu pracy,

- **10** – regulator PID,
- **20** – regulator DMC.

Dodatkowo jeśli wybrany zostanie tryb regulatorów, włączany jest automat stanów odpowiedzialny za zmiany wartości zadanych (listing 2.4).

- **Modbus** – odpowiada za pobieranie i wysyłanie danych do stanowiska grzewczo-chłodzącego.
- **SocketComm** – wysyłanie danych do MATLABa w celu archiwizacji.
- **Regulatory** - blok programu ustawiony na przerwania co 4 sekundy.
 - **odczytanie** – konwersja pomiaru temperatury na typ FLOAT i przypisanie do odpowiedniego procesu.
 - **zabezpieczenie** – wykrycie temperatury przekraczającej bezpieczną wartość.
 - **PID_funkcja** – omówione w dalszej części.
 - **DMC_funkcja** – omówione w dalszej części.
 - **wysylanie** – ograniczenie sygnału sterującego i konwersja.



Rys. 2.2. Drzewo programu projektu

2.2. Wyznaczenie punktu pracy

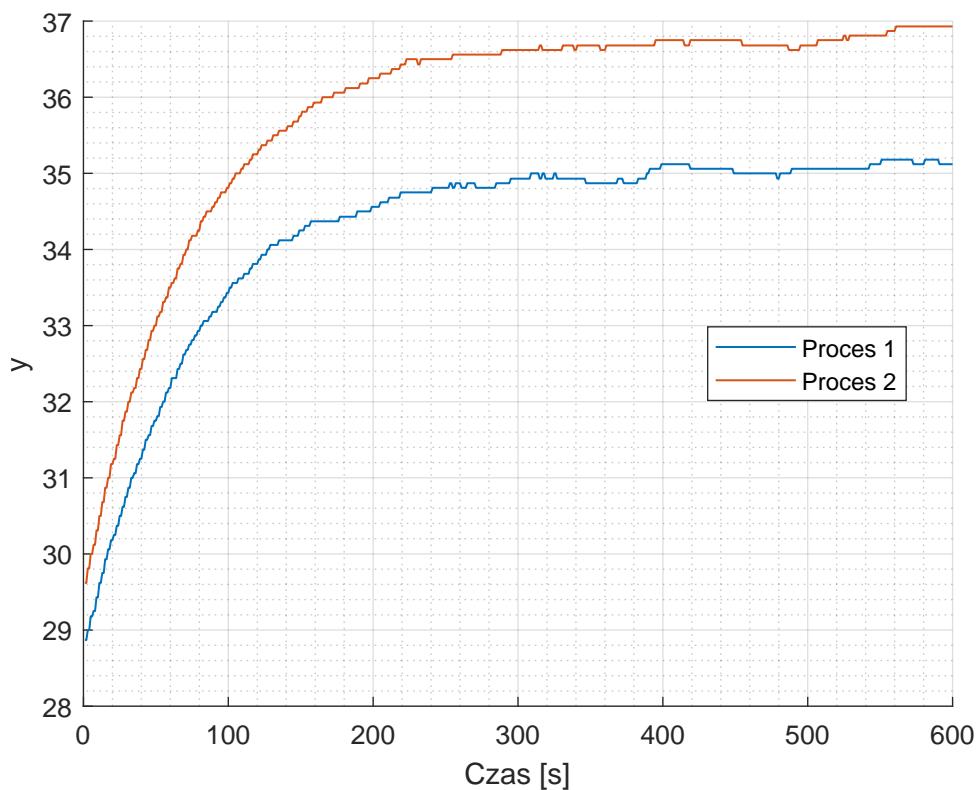
Aby wyznaczyć punkt pracy dla $G1=25$, $G2=30$, $W1=W2=50$, należało ustawić odpowiednie moce na wentylatorach i grzałkach oraz zapisywać pomiary temperatury. Ze względu na różne typy zmiennych należało posłużyć się przeliczeniami między wartościami całkowitoliczbowymi a zmiennoprzecinkowymi i odwrotnie. W listingu 2.1 przedstawiono uproszczony kod do wyznaczenia punktu pracy (w praktyce kod został podzielony między blokami programów, w celu jego uniwersalności).

Listing 2.1. Kod do wysterowania punktu pracy

```
Proces1.y_k := INT_TO_REAL(wT1)/100.0;
Proces2.y_k := INT_TO_REAL(wT3)/100.0;

wW1 := 500;
wW2 := 500;

Proces1.u_k := 25.0;
Proces2.u_k := 30.0;
wG1 := REAL_TO_INT(Proces1.u_k*10.0);
wG2 := REAL_TO_INT(Proces2.u_k*10.0);
```



Rys. 2.3. Wyznaczenie punktu pracy

Wartość pomiaru temperatury w punkcie pracy: $T1=35,2$, $T3=37,0$.

2.3. Mechanizm zabezpieczający przed uszkodzeniem stanowiska

Przy przekroczeniu temperatury 250°C (co oznacza uszkodzenie czujnika) grzałka sąsiadująca z czujnikiem, który zmierzył niebezpieczną temperaturę musi zostać wyłączona (ustawiona na

moc 0). Jeśli awaria nastąpi podczas działania regulatorów, ich praca zostaje wstrzymana aż do zniknięcia awarii.

Listing 2.2. Zawartość **Regulatory/zabezpieczenie**

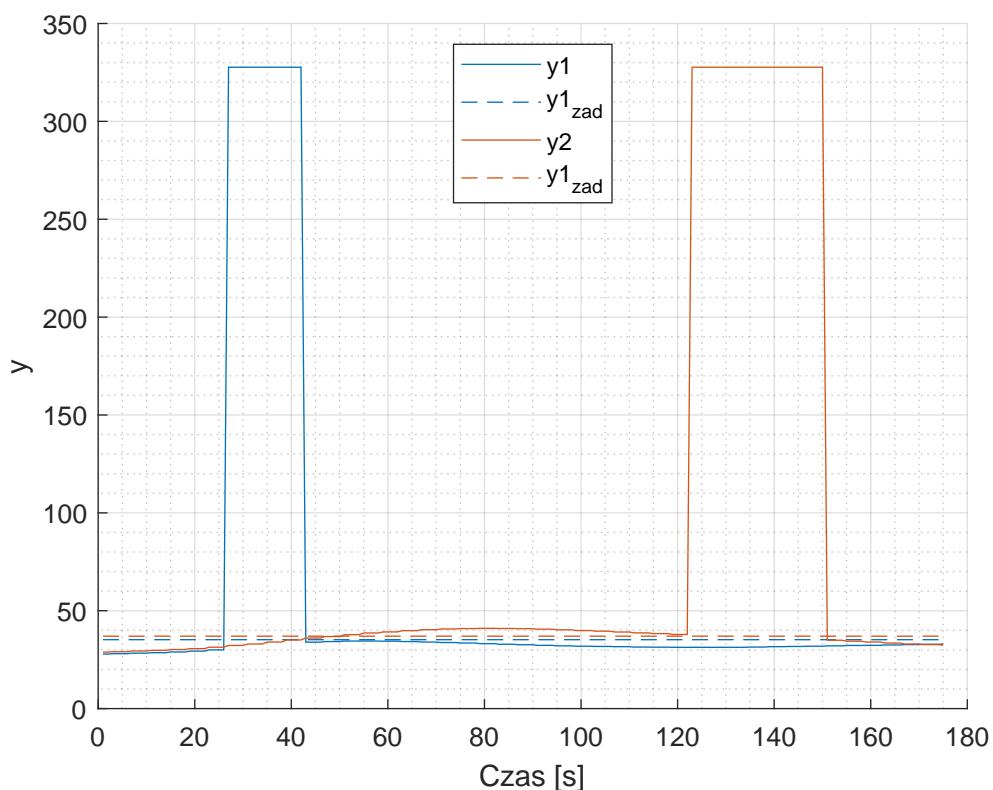
```

IF Proces1.y_k > 250.0 THEN
    awaria_T1 := TRUE;
    Proces1.u_k := 0.0;
ELSE
    awaria_T1 := FALSE;
END_IF;

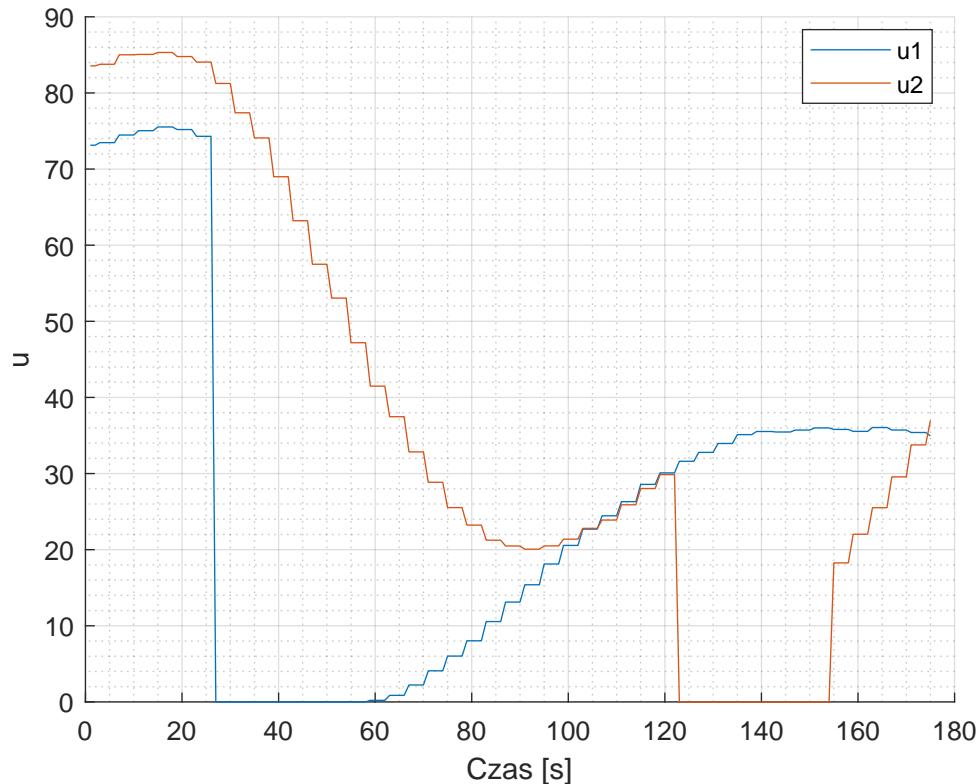
IF Proces2.y_k > 250.0 THEN
    awaria_T2 := TRUE;
    Proces2.u_k := 0.0;
ELSE
    awaria_T2 := FALSE;
END_IF;

```

Przetestowano działanie zabezpieczenia podczas działania regulatora PID i sprawdzono działanie dla awarii najpierw na czujniku T1, następnie na czujniku T3, co pokazano na rysunkach 2.4 i 2.5. W momencie wykrycia awarii na którymś z czujników, odpowiedni sygnał sterujący jest ustalany na 0, a regulator jest wyłączony. Po ustąpieniu awarii regulator wznowia pracę, jednak pamięta, że ostatnia wartość sygnału sterującego wynosiła 0.



Rys. 2.4. Działanie mechanizmu zabezpieczającego - sygnał wyjściowy



Rys. 2.5. Działanie mechanizmu zabezpieczającego - sygnał sterujący

2.4. Automaty stanów

Listing 2.3. Zawartość MAIN/Automat_stanow - Automat zarządzający trybem pracy

```

CASE tryb_pracy OF
  0: // punkt pracy, regulatory wyłączone
    PID1.control_ON := FALSE;
    PID2.control_ON := FALSE;
    DMC.control_ON := FALSE;

    wW1 := 500;
    wW2 := 500;
    Proces1.u_k := 25.0;
    Proces2.u_k := 30.0;
    stan_yzad := 0;

  10: // PID
    DMC.control_ON := FALSE;
    PID1.control_ON := TRUE;
    PID2.control_ON := TRUE;
    stan_yzad := 10;

    tryb_pracy := 11; // przejście do pustego stanu

  20: // DMC
    PID1.control_ON := FALSE;
    PID2.control_ON := FALSE;
    DMC.control_ON := TRUE;
    stan_yzad := 10;
    tryb_pracy := 21; // przejście do pustego stanu
END_CASE;

```

Automat stanów zmieniający wartości zadane względem temperatur z punktu pracy ($T1pp$, $T3pp$) = $(0, 0) \rightarrow (+15, 0) \rightarrow (+15, +15) \rightarrow (0, +15)$

Listing 2.4. Zawartość **MAIN/Automat_stanow** - Automat do zmieniania wartości zadanej

```

TIM_yzad(PT:= T#300s);

CASE stan_yzad OF
  0: // automat wyłączony, wartości zadane w punkcie pracy
    TIM_yzad.IN := FALSE;
    Proces1.y_zad := Proces1.y_pp;
    Proces2.y_zad := Proces2.y_pp;

  10: // restet timera, jeśli regulator został ponownie uruchomiony
    TIM_yzad.IN := FALSE;
    stan_yzad := 15;

  15: // pierwszy skok, wartości zadane w punkcie pracy
    TIM_yzad.IN := TRUE;
    Proces1.y_zad := Proces1.y_pp;
    Proces2.y_zad := Proces2.y_pp;

    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 25;
    END_IF;

  20: // drugi skok
    TIM_yzad.IN := TRUE;
    Proces1.y_zad := Proces1.y_pp+15.0;
    Proces2.y_zad := Proces2.y_pp;

    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 25;
    END_IF;

  25: // trzeci skok
    TIM_yzad.IN := TRUE;
    Proces1.y_zad := Proces1.y_pp+15.0;
    Proces2.y_zad := Proces2.y_pp+15.0;

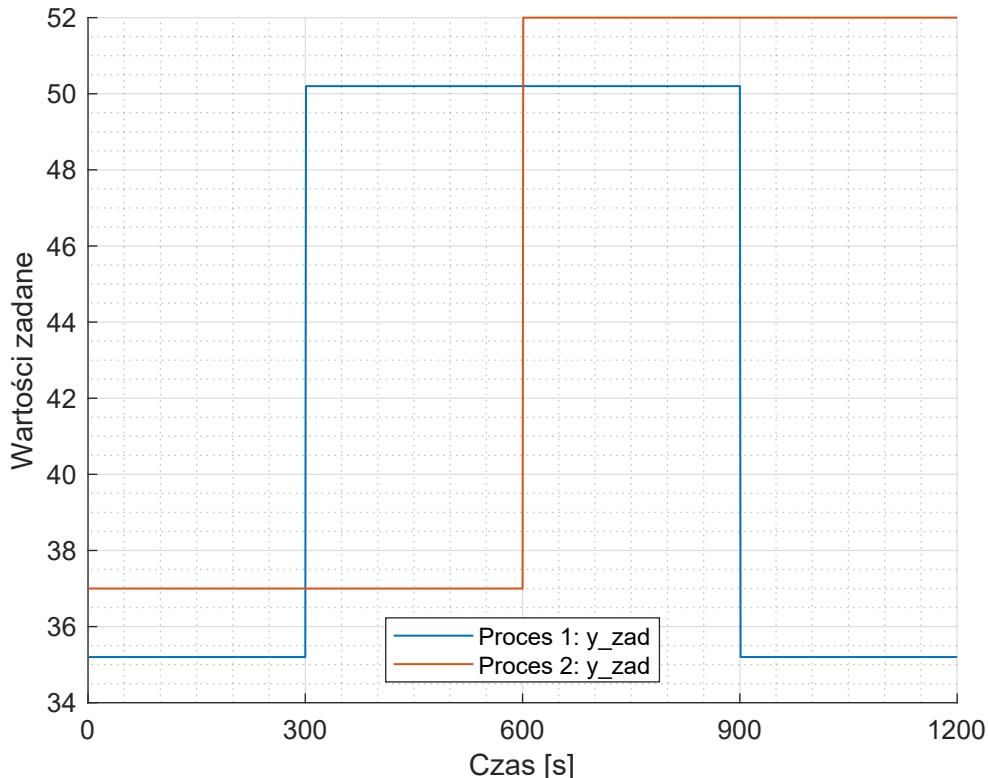
    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 30;
    END_IF;

  30: // czwarty skok
    TIM_yzad.IN := TRUE;
    Proces1.y_zad := Proces1.y_pp;
    Proces2.y_zad := Proces2.y_pp+15.0;

    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 15;
    END_IF;

END_CASE;

```



Rys. 2.6. Trajektoria wartości zadanych

2.5. Dwupętlowy regulator PID

2.5.1. Implementacja

	Label Name	Data Type	Comment
1	K_gain	FLOAT [Single Precision]	Wzmocnienie K
2	Ti	FLOAT [Single Precision]	Stała całkująca Ti
3	Td	FLOAT [Single Precision]	Stała różniczkująca Td
4	r_0	FLOAT [Single Precision]	
5	r_1	FLOAT [Single Precision]	
6	r_2	FLOAT [Single Precision]	
7	e_k	FLOAT [Single Precision]	Uchyb w chwili k
8	e_km1	FLOAT [Single Precision]	Uchyb w chwili poprzedniej
9	e_km2	FLOAT [Single Precision]	Uchyb w chwili przedostatniej
10	T_p	FLOAT [Single Precision]	Okres próbkowania
11	control_ON	Bit	Czy regulator jest włączony

Rys. 2.7. Struktura struct_PID

Listing 2.5. Zawartość INIT/PID_konfiguracja

```
PID1.control_ON := FALSE;
PID1.K_gain := 6.0;
PID1.Ti := 80.0;
PID1.Td := 0.3;

PID1.T_p := 4.0;
PID1.e_k := 0.0;
PID1.e_km1 := 0.0;
PID1.e_km2 := 0.0;
```

```

PID1.r_0 := PID1.K_gain*(1.0+(PID1.T_p/(2.0*PID1.Ti))+PID1.Td/PID1.T_p);
PID1.r_1 := PID1.K_gain*((PID1.T_p/(2.0*PID1.Ti))-(2.0*PID1.Td/PID1.T_p)-1.0);
PID1.r_2 := PID1.K_gain*PID1.Td/PID1.T_p;

PID2.control_ON := FALSE;
PID2.K_gain := 6.0;
PID2.Ti := 80.0;
PID2.Td := 0.3;

PID2.T_p := 4.0;
PID2.e_k := 0.0;
PID2.e_km1 := 0.0;
PID2.e_km2 := 0.0;

PID2.r_0 := PID2.K_gain*(1.0+(PID2.T_p/(2.0*PID2.Ti))+PID2.Td/PID2.T_p);
PID2.r_1 := PID2.K_gain*((PID2.T_p/(2.0*PID2.Ti))-(2.0*PID2.Td/PID2.T_p)-1.0);
PID2.r_2 := PID2.K_gain*PID2.Td/PID2.T_p;

```

Obliczenia online

Listing 2.6. Zawartość **Regulatory/PID_funkcja**

```

IF PID1.control_ON AND NOT awaria_T1 THEN
    PID1.e_k := Proces1.y_zad - Proces1.y_k;

    Proces1.u_k := PID1.r_2 * PID1.e_km2 + PID1.r_1 * PID1.e_km1 +
                  PID1.r_0 * PID1.e_k + Proces1.u_km1;

    PID1.e_km1 := PID1.e_k;
    PID1.e_km2 := PID1.e_km1;
END_IF;

IF PID2.control_ON AND NOT awaria_T2 THEN
    PID2.e_k := Proces2.y_zad - Proces2.y_k;

    Proces2.u_k := PID2.r_2 * PID2.e_km2 + PID2.r_1 * PID2.e_km1 +
                  PID2.r_0 * PID2.e_k + Proces2.u_km1;

    PID2.e_km1 := PID2.e_k;
    PID2.e_km2 := PID2.e_km1;
END_IF;

```

Ograniczenia i aktualizacja wcześniejszych sterowań

Listing 2.7. Zawartość **Regulatory/wysylanie**

```

IF Proces1.u_k > 100.0 THEN
    Proces1.u_k := 100.0 ;
ELSIF Proces1.u_k < 0.0 THEN
    Proces1.u_k := 0.0 ;
END_IF;

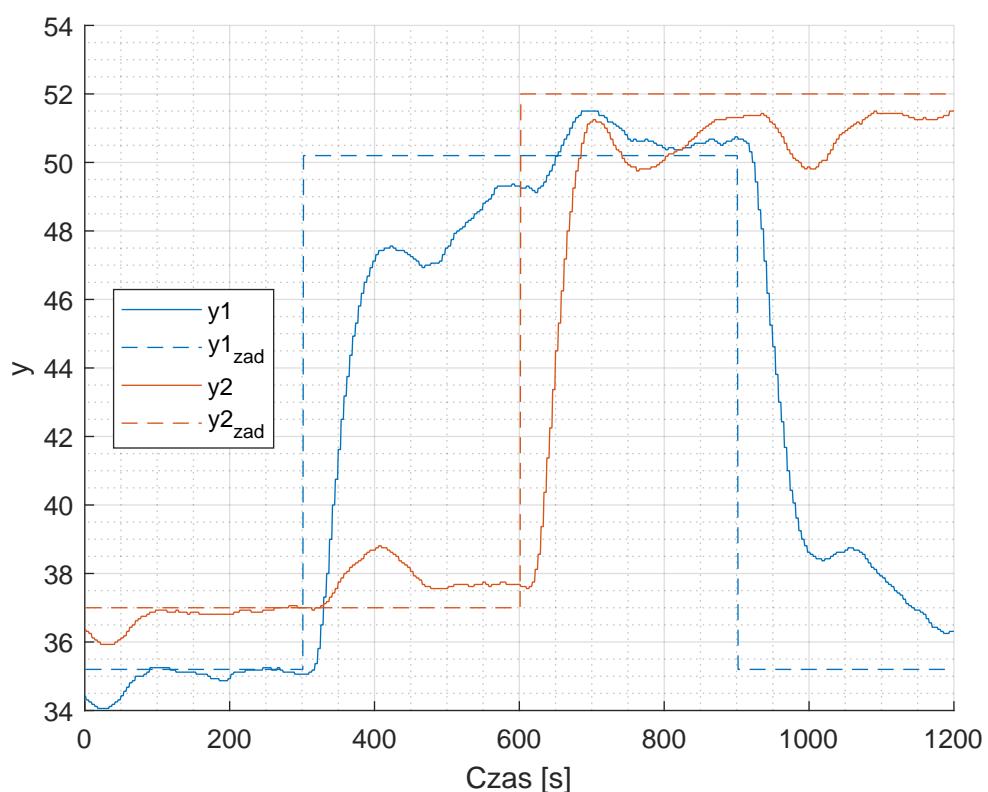
IF Proces2.u_k > 100.0 THEN
    Proces2.u_k := 100.0 ;
ELSIF Proces2.u_k < 0.0 THEN
    Proces2.u_k := 0.0 ;
END_IF;

Proces1.u_km1 := Proces1.u_k;
Proces2.u_km1 := Proces2.u_k;

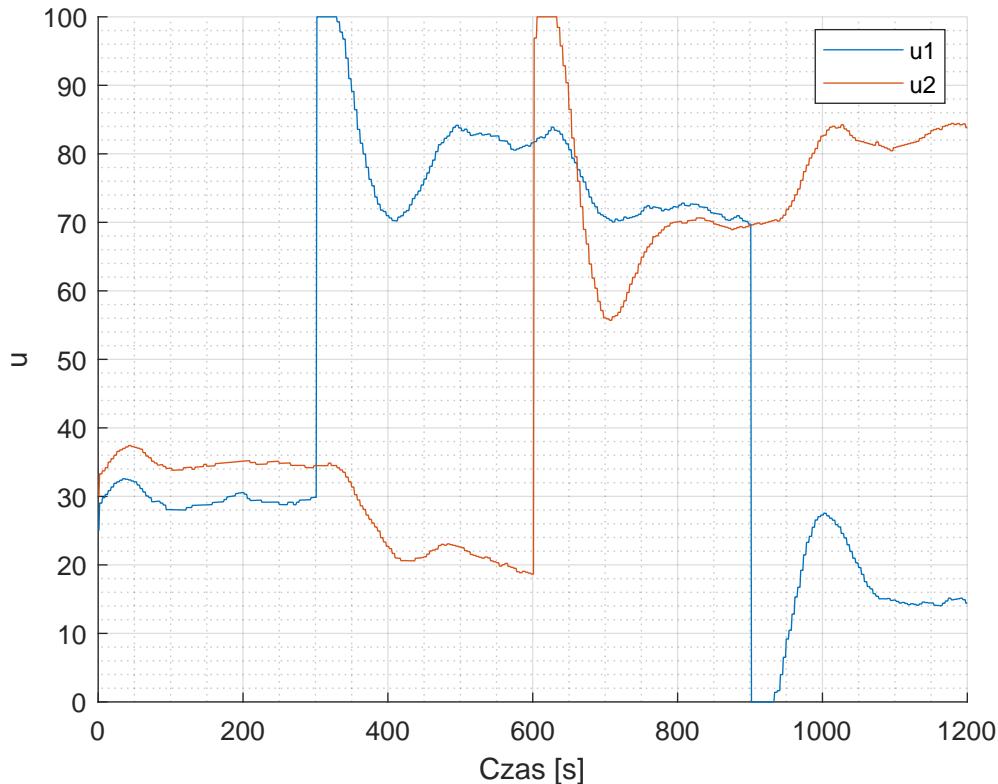
```

2.5.2. Strojenie

Na początku sprawdzono regulator PID z parametrami zapożyczonymi z poprzednich laboratoriów - $K=5$, $T_i=100$, $T_d=0,1$ - a działanie pokazano na rysunkach 2.8 i 2.9. Regulator dobrze sobie radzi na początku przebiegu, kiedy temperatury znajdują się blisko swoich wartości zadanego i kiedy wartości zadane nie różnią się zbytnio między procesami. Mimo wielowymiarowości zdąży do odległej wartości zadanej, np przy skoku wartości zadanej procesu 1 do wartości 50,2. Wtedy temperatura na y_2 również chwilowo wzrosła, jednak regulator procesu drugiego starał się niwelować wpływ grzałki z pierwszego procesu. Występują pewne niedosterowania na obu procesach, dlatego postanowiono zwiększyć wpływ członów proporcjonalnego i całkującego (różniczkujący ze względu na złożoność problemu pozostawiono bez zmian).

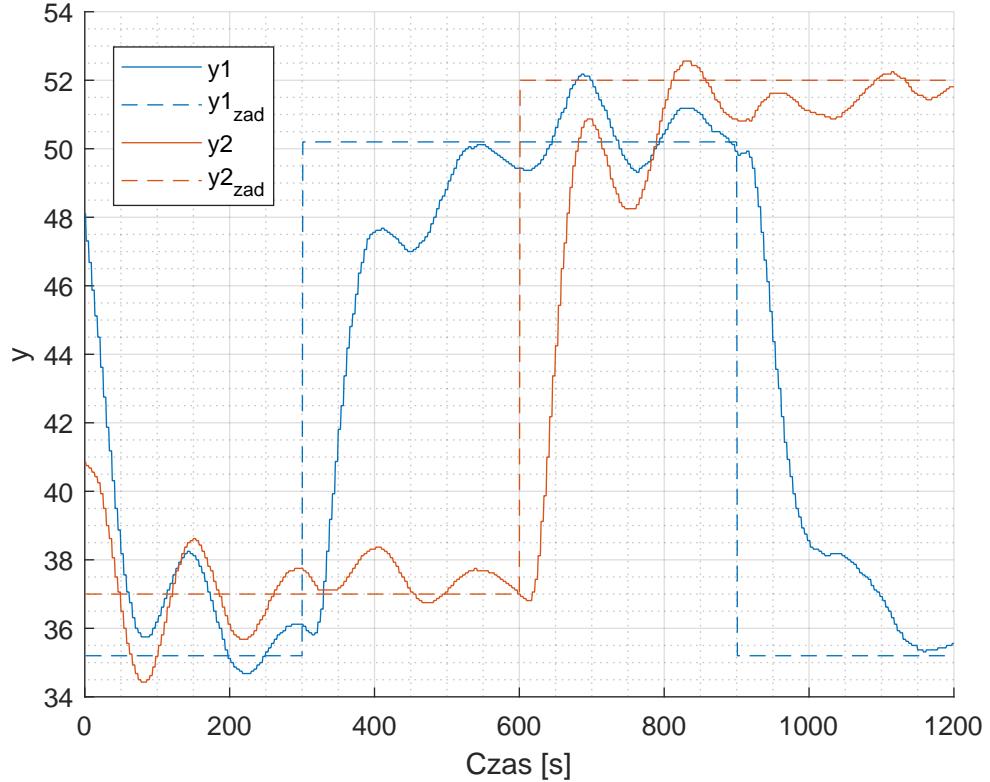


Rys. 2.8. Regulator PID z parametrami $K=5$, $T_i=100$, $T_d=0,1$ - sygnał wyjściowy

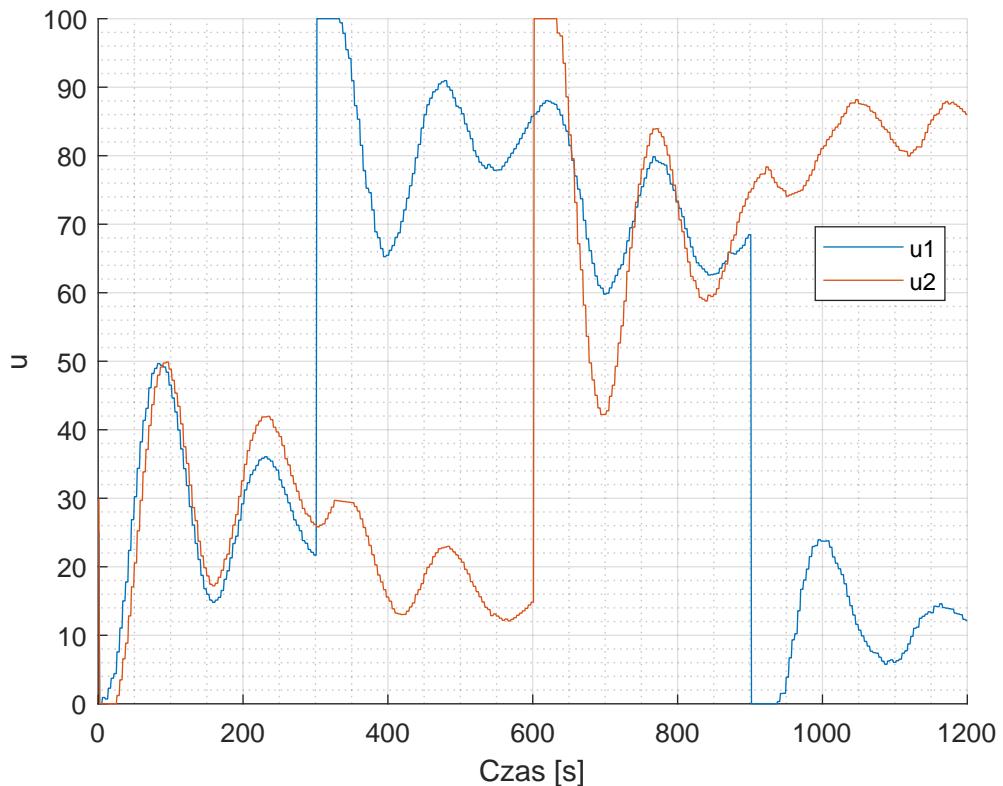


Rys. 2.9. Regulator PID z parametrami $K=5$, $T_i=100$, $T_d=0,1$ - sygnał sterujący

Kolejno przetestowano regulator z parametrami $K=8$, $T_i=70$, $T_d=0,1$, a działanie pokazano na rysunkach 2.10 i 2.11. Temperatury obu procesów bardzo oscylują, co jest spowodowane wspomnianym zwiększeniem wpływu członów proporcjonalnego i całkującego. Jednak mimo oscylacji temperatury szybciej zbiegają do wyższych wartości zadanych i w przypadku procesu 2 uchyb ustalony jest mniej zauważalny.



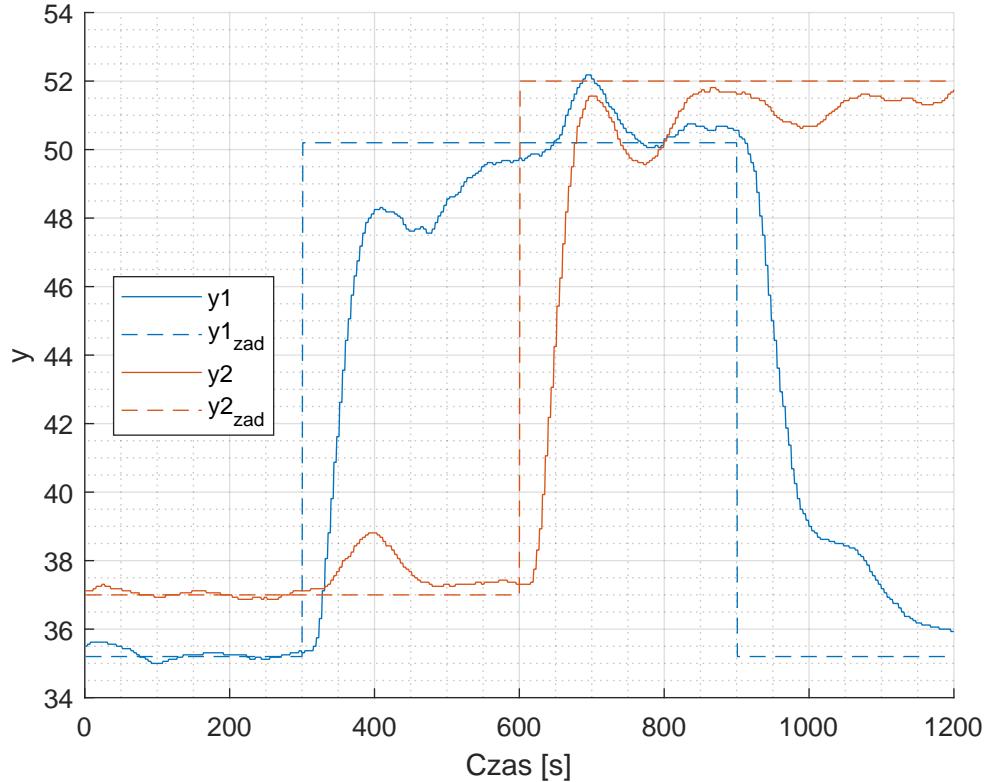
Rys. 2.10. Regulator PID z parametrami $K=8$, $T_i=70$, $T_d=0,1$ - sygnał wyjściowy



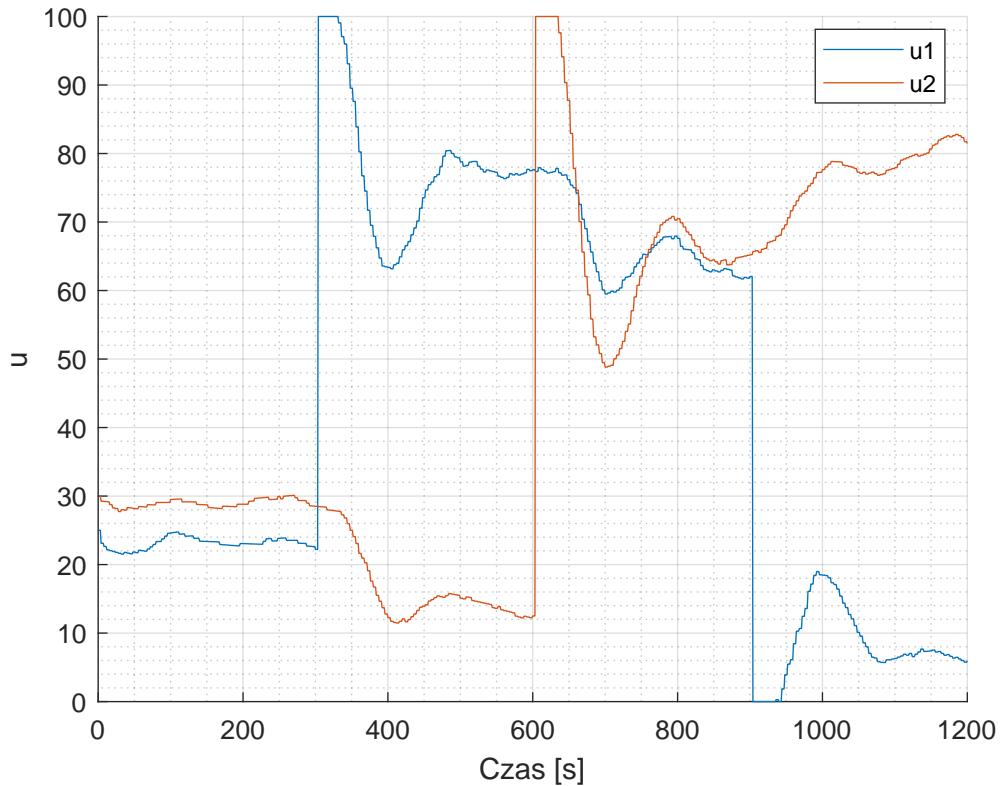
Rys. 2.11. Regulator PID z parametrami $K=8$, $T_i=70$, $T_d=0,1$ - sygnał sterujący

Ostatecznie sprawdzono regulator PID z parametrami $K=6$, $T_i=80$, $T_d=0,1$ (rys. 2.12 i 2.13). Regulator dobrze sobie radzi z utrzymywaniem temperatury zadanej, jeśli utrzymuje się w jej

otoczeniu. Uchyb ustalony jest zdecydowanie mniejszy w porównaniu z pierwszymi nastawami regulatora PID ($K=5$, $T_i=100$, $T_d=0,1$), oscylacje również są mniejsze.



Rys. 2.12. Regulator PID z parametrami $K=6$, $T_i=80$, $T_d=0,1$ - sygnał wyjściowy



Rys. 2.13. Regulator PID z parametrami $K=6$, $T_i=80$, $T_d=0,1$ - sygnał sterujący

W tabelce 2.2 zestawiono błędy średnio kwadratowe wszystkich trzech badanych regulatorów, dla pomiarów z chwil 300-1200, aby pominąć wpływ początku działania regulatora. Widać, że najlepszy PID pod względem błędu to ostatni badany o nastawach $K=6$, $T_i=80$, $T_d=0,1$.

Tab. 2.2. Wartości parametrów i błędów regulatorów PID

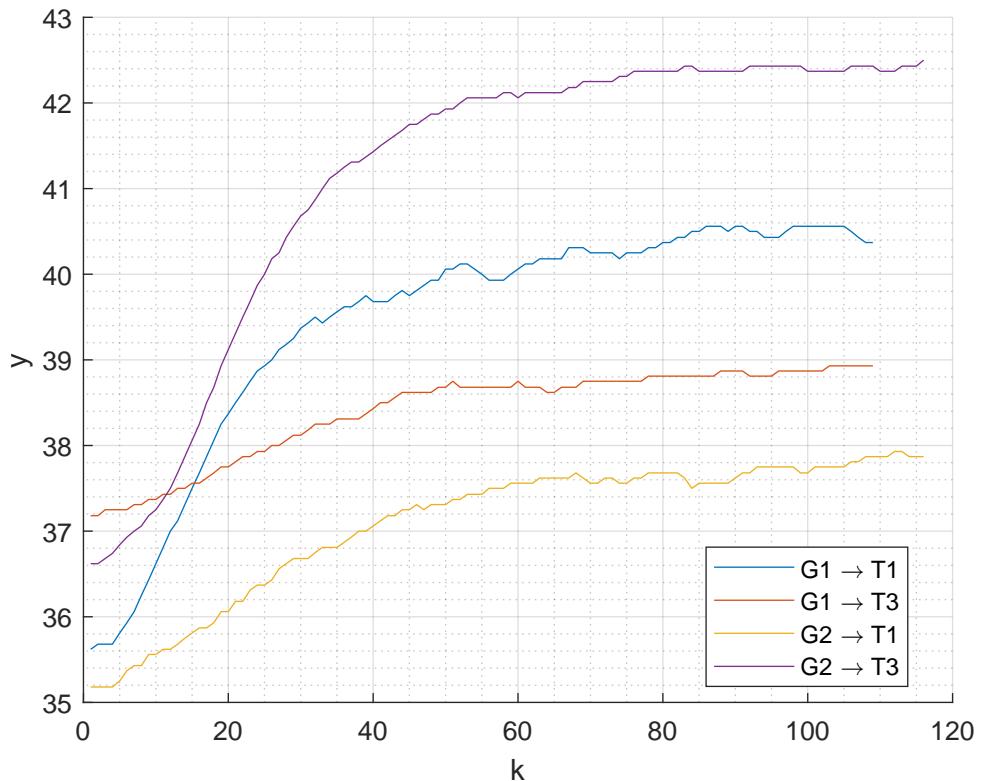
Parametry PID	Błąd (E)
$K=5$, $T_i=100$	25,231
$K=8$, $T_i=70$	22,312
$K=6$, $T_i=80$	20,852

Czas na ustabilizowanie się był dość krótki - zaledwie 300s - jednak wystarczyło to na ocenę jakości tych regulatorów.

2.6. Regulator DMC 2×2 w wersji analitycznej

2.6.1. Odpowiedzi skokowe

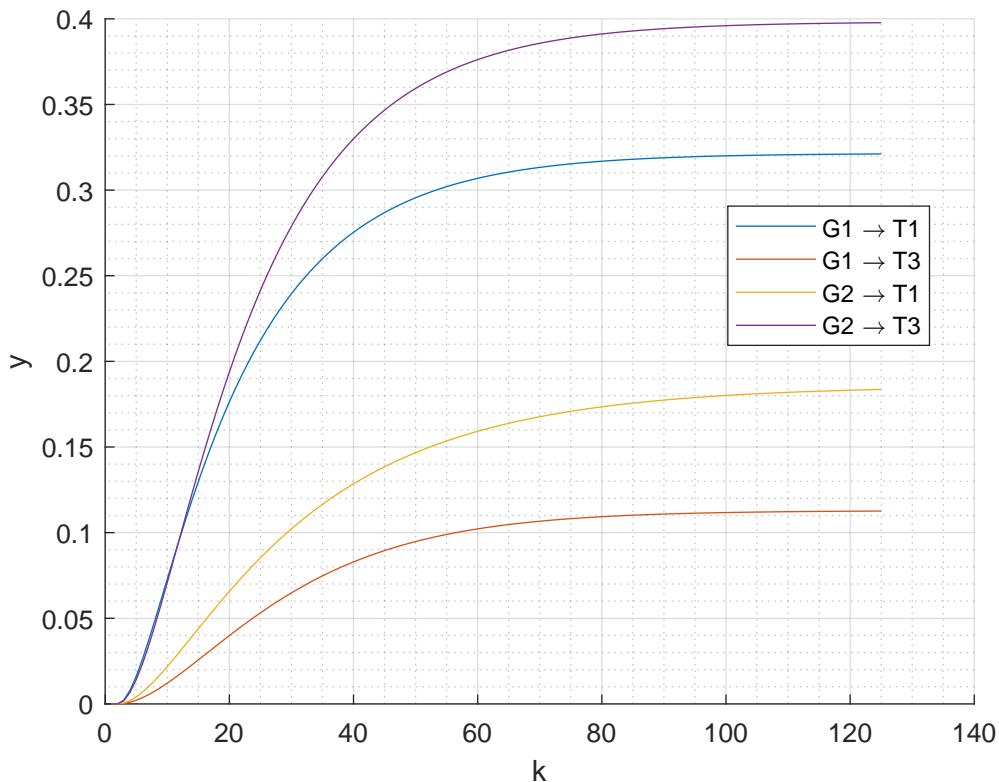
Wykonano skoki o wartość +15 z punktu pracy dla każdego z wejść. Ponieważ jest to obiekt MIMO, dwa skoki sygnału sterującego generują cztery odpowiedzi skokowe, czyli we wszystkich możliwych torach sterowanie → wyjście.



Rys. 2.14. Pozyskane odpowiedzi skokowe

Do aproksymacji użyto modelu dwuinercyjnego z opóźnieniem i algorytmu genetycznego w celu optymalnego dobrania parametrów modelu.

W poprzednich blokach laboratoryjnych, gdzie okresu próbkowania wynosił 1s, przyjmowano horyzont dynamiki 500. W tym bloku w tym bloku nastąpiła zmiana okresu na 4s, jednak postanowiono przyjąć taki sam czas na ustalenie się temperatury - 500s, czyli początkowy horyzont dynamiki przyjęto $D=125$.



Rys. 2.15. Aproksymowane odpowiedzi skokowe

2.6.2. Implementacja

Implementację regulatora DMC na sterownik PLC, rozpoczęto od obliczeń offline w MATLAB (listing 2.8), ze względu na złożoność obliczeniową. Do wyznaczenia parametrów K_e i K_u wykorzystano odpowiedzi skokowe pozyskane w rozdziale 2.6.1 oraz funkcje **DMC_MIMO_offline**, omówioną już w części projektowej.

Przeniesienie gotowych wartości parametrów do PLC, przeprowadzono poprzez wygenerowanie pliku txt, zawierającego przypisanie wartości do zmiennych, zachowując przy tym składnię języka ST i przyjętą strukturę danych **struct_DMC** wykorzystywaną na zmienne w regulatorze DMC (rys. 2.16). Zawartość wygenerowanego pliku przekopiowano następnie do **INI-T/DMC_konfiguracja** w sterowniku PLC (listing 2.9).

Listing 2.8. Obliczenia K_e i K_u w MATLAB z konwersją do PLC

```

D= 125;
N = D;
Nu = D;

S = odp_jedn(D+N);
lambda=[1 1]; psi=[1 1];

[Ke,Ku]=DMC_MIMO_offline(S,D,N,Nu,lambda,psi);

% Definiowanie pliku wyjściowego
fileID = fopen('output.txt', 'w');

fprintf(fileID,'DMC.D_ := %d;\n', D);
% Wypisywanie macierzy Ke do pliku

```

```

for row = 1:2
    for col = 1:2
        fprintf(fileID, 'DMC.Ke[%d,%d] := %.4f;\n', ...
            row-1, col-1, Ke(row, col));
    end
end

% Wypisywanie macierzy Ku do pliku
for i = 1:(D-1)
    for row = 1:2
        for col = 1:2
            fprintf(fileID, 'DMC.Ku[%d,%d,%d] := %.4f;\n', ...
                i-1, row-1, col-1, Ku(row, col+(i-1)*2));
        end
    end
end

fclose(fileID);

```

	Label Name	Data Type	Comment
1	Ke	FLOAT [Single Precision](0..1,0..1)	...
2	Ku	FLOAT [Single Precision](0..124,0..1,0..1)	...
3	control_ON	Bit	Czy regulator jest włączony
4	delta_u1	FLOAT [Single Precision]	Przyrost sterowania procesu 1
5	delta_u2	FLOAT [Single Precision]	Przyrost sterowania procesu 2
6	D_	Word [Signed]	Horyzont dynamiki
7	delta_u1_p	FLOAT [Single Precision](0..124)	Wektor przyrostów przeszlych sterowań procesu 1
8	delta_u2_p	FLOAT [Single Precision](0..124)	Wektor przyrostów przeszlych sterowań procesu 2

Rys. 2.16. Struktura struct_DMC

Listing 2.9. Zawartość INIT/DMC_konfiguracja

```

DMC.D_ := 125;
DMC.Ke[0,0] := 0.9336;
DMC.Ke[0,1] := -0.0660;
DMC.Ke[1,0] := 0.0407;
DMC.Ke[1,1] := 0.9277;
DMC.Ku[0,0,0] := 0.1324;
DMC.Ku[0,0,1] := 0.0268;
DMC.Ku[0,1,0] := 0.0265;
DMC.Ku[0,1,1] := 0.1481;
...

```

Listing 2.10. Zawartość Regulatory/DMC_funkcja

```

IF DMC.control_ON THEN
    DMC.delta_u1 := DMC.Ke[0,0]*(Proces1.y_zad-Proces1.y_k) +
                    DMC.Ke[0,1]*(Proces2.y_zad-Proces2.y_k);
    DMC.delta_u2 := DMC.Ke[1,0]*(Proces1.y_zad-Proces1.y_k) +
                    DMC.Ke[1,1]*(Proces2.y_zad-Proces2.y_k);

    FOR krok := 0 TO DMC.D_ - 2 BY 1 DO
        DMC.delta_u1 := DMC.delta_u1 - DMC.Ku[krok,0,0]*DMC.delta_u1_p[krok] -
                        DMC.Ku[krok,0,1]*DMC.delta_u2_p[krok];
        DMC.delta_u2 := DMC.delta_u2 - DMC.Ku[krok,1,0]*DMC.delta_u1_p[krok] -
                        DMC.Ku[krok,1,1]*DMC.delta_u2_p[krok];

```

```

END_FOR;

// ograniczenia
IF proces1.u_km1 + DMC.delta_u1 < 0.0 THEN
    DMC.delta_u1 := 0.0 - proces1.u_km1;
ELSIF proces1.u_km1 + DMC.delta_u1 > 100.0 THEN
    DMC.delta_u1 := 100.0 - proces1.u_km1;
ENDIF;

// wyliczenie sygnału sterującego
IF NOT awaria_T1 THEN
    proces1.u_k := proces1.u_km1 + DMC.delta_u1;
ELSE // dodatkowy warunek, aby po zakończeniu awarii DMC miał poprawne dane
    Proces1.u_k := 0.0;
    DMC.delta_u1 := proces1.u_k - proces1.u_km1;
ENDIF;

// aktualizacja wektora przeszłych przyrostów
FOR krok := DMC.D_ - 2 TO 1 BY -1 DO
    DMC.delta_u1_p[krok] := DMC.delta_u1_p[krok-1];
END_FOR;
DMC.delta_u1_p[0] := DMC.delta_u1;

IF proces2.u_km1 + DMC.delta_u2 < 0.0 THEN
    DMC.delta_u2 := 0.0 - proces2.u_km1;
ELSIF proces2.u_km1 + DMC.delta_u2 > 100.0 THEN
    DMC.delta_u2 := 100.0 - proces2.u_km1;
ENDIF;

IF NOT awaria_T2 THEN
    proces2.u_k := proces2.u_km1 + DMC.delta_u2;
ELSE
    Proces2.u_k := 0.0;
    DMC.delta_u2 := proces2.u_k - proces2.u_km1;
ENDIF;

FOR krok := DMC.D_ - 2 TO 1 BY -1 DO
    DMC.delta_u2_p[krok] := DMC.delta_u2_p[krok-1];
END_FOR;
DMC.delta_u2_p[0] := DMC.delta_u2;

ENDIF;

```

Ilość wykorzystanych rejestrów 32-bitowych na zmienne FLOAT w regulatorze DMC i zależnych bezpośrednio od długości horyzontu dynamiki:

- Ku: $D \cdot 2 \cdot 2$
- Wektory przeszłych przyrostów: $2 \cdot D$

Łącznie wykorzystuje się ponad $D \cdot 6$ rejestrów 32-bitowych.

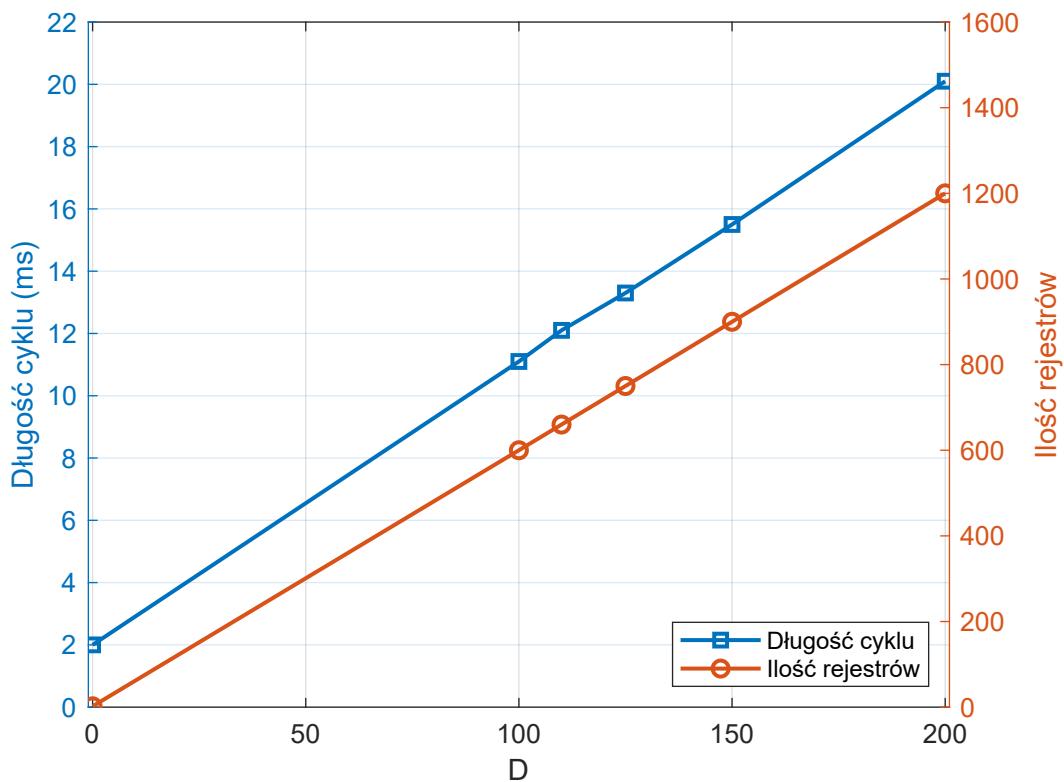
Przy horyzoncie dynamiki $D=125$, ilość wykorzystanych rejestrów to 750, a czas cyklu wynosi około 13,3ms. Dla porównania, cykl z wykonaniem przerwania, ale z wyłączoneymi regulatorami trwa około 2ms. Sprawdzono jeszcze wpływ długości D na czas cyklu i ilość rejestrów, wyniki pokazano w tabeli 2.3 oraz na wykresie rys. 2.17. W przybliżeniu są to zależności liniowe.

Z uwagi na ograniczony czas i chęć pełnego przetestowania stanowiska INTECO, pominieto sprawdzanie wpływu długości D na jakość regulacji. Uznano, że cykl regulatora DMC z $D=125$ nie jest krytycznie długi patrząc na potrzeby stanowiska (pomiary i sterowanie co 4s), a skracac-

nie horyzontu może tylko pogorszyć jakość regulacji, dlatego **pozostano przy początkowym horyzoncie dynamiki $D=125$** .

Tab. 2.3. Zużycie rejestrów i czasu w zależności od horyzontu dynamiki D

D	Ilość rejestrów	Długość cyklu (ms)
0 (OFF)		1,9
100	600	11,1
110	660	12,1
125	750	13,3
150	900	15,5
200	1200	20,1

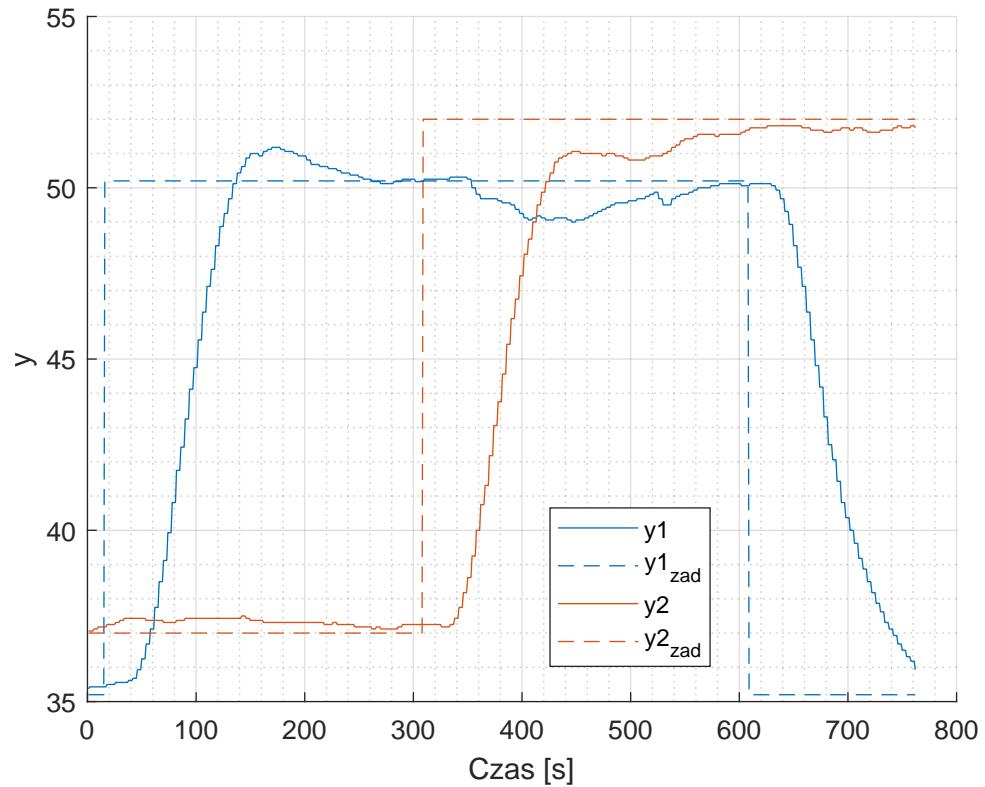
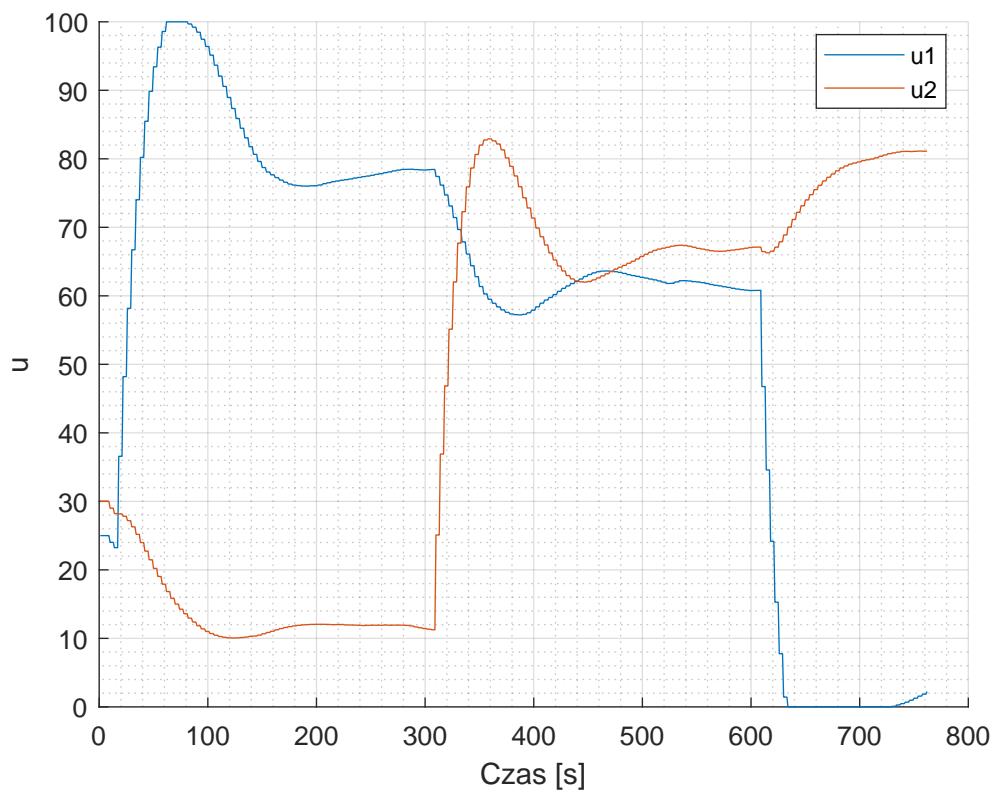


Rys. 2.17. Zużycie rejestrów i czasu w zależności od horyzontu dynamiki D

2.6.3. Strojenie

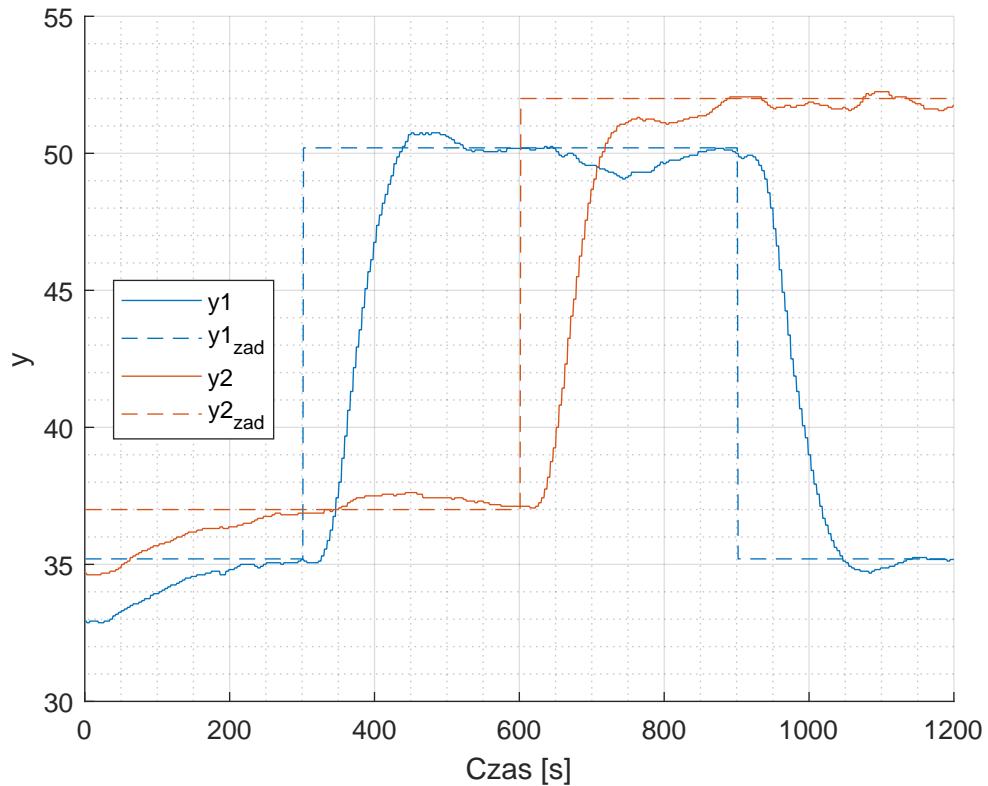
Przetestowano działanie regulatora DMC z parametrem $\lambda=1$ dla części trajektorii sygnałów zadanych, a wyniki przedstawiono na rys. 2.18 i 2.19. Stwierdzono, że implementacja regulatora jest poprawna i regulator działa według założeń.

Podeczas przeprowadzania pomiaru, stwierdzono, że temperatura zbyt wolno zbiega do wartości zadanej, dlatego postanowiono zmniejszyć współczynnik kary, by regulator działał bardziej agresywnie.

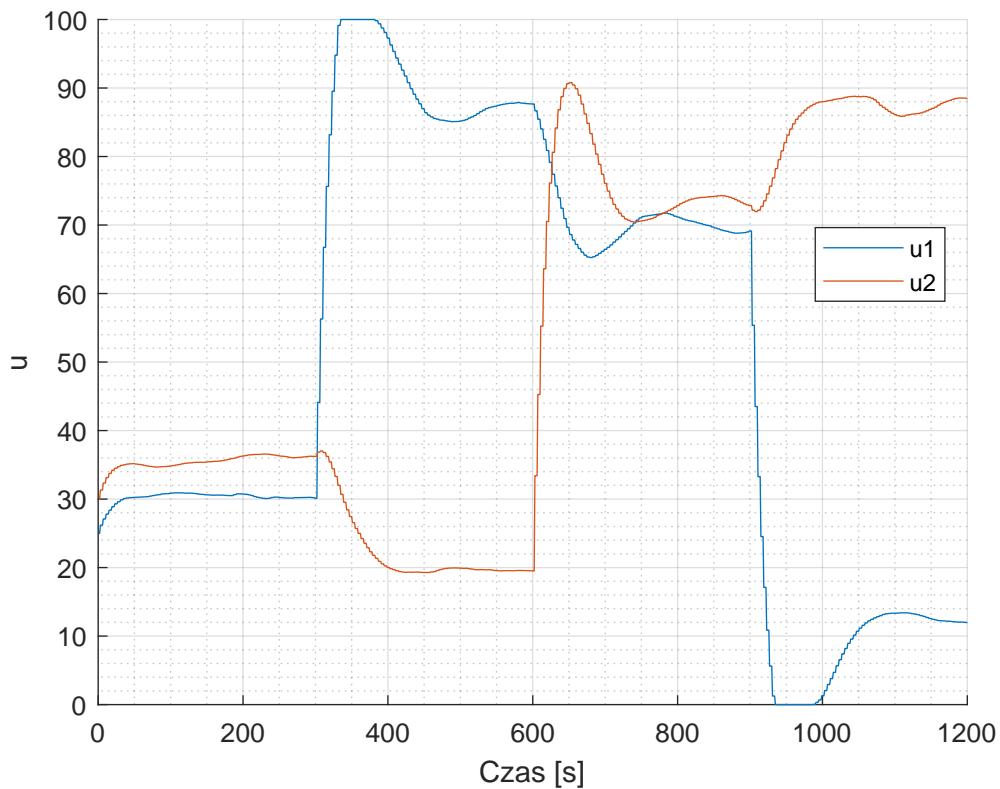
Rys. 2.18. Regulator DMC z $\lambda=1$ - sygnał wyjściowyRys. 2.19. Regulator DMC z $\lambda=1$ - sygnał sterujący

Zmieniono współczynnik kary na $\lambda=0,5$ (rys. 2.20 i 2.21). Sygnał sterujący zmienia się za-

uważalnie szybciej i osiąga wyższe wartości (dla skoku y_{2zad} do wartości 52 sygnał sterujący procesu 2 osiąga wartości rzędu 90, a dla poprzedniej nastawy ledwo przekraczał 80 po 600s).



Rys. 2.20. Regulator DMC z $\lambda=0,5$ - sygnał wyjściowy

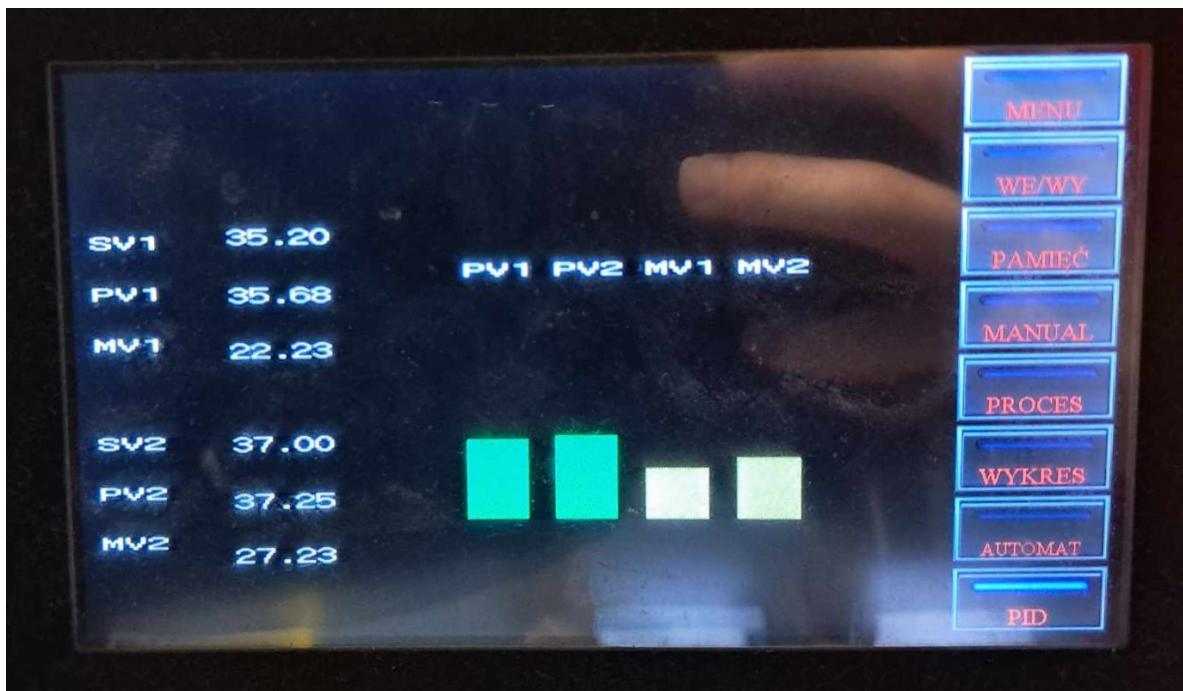


Rys. 2.21. Regulator DMC z $\lambda=0,5$ - sygnał sterujący

Regulator DMC zdecydowanie lepiej sobie radzi z obiektami MIMO w porównaniu z regulatorami PID. Np. podczas pierwszego skoku wartości zadanej procesu 1, temperatura procesu 2 praktycznie się nie zmienia. Jest to spowodowane jego implementacją, ponieważ przy wyznaczaniu sterowania bierze pod uwagę wpływ wszystkich wejść na każde z wyjść.

2.7. Panel operatora

Zadaniem było wyświetlić na panelu operatora wartości mierzone, zadane oraz sterowania oraz zaprezentować je w najprostszej formie graficznej. Do tego celu przerobiono gotowy szablon wyświetlający wartości procesu jednowymiarowego. Ostateczny panel HMI w formie minimalistycznej zaprezentowano na rys. 2.22.



Rys. 2.22. HMI stanowiska grzewczo-chłodzącego

W przedstawionym przykładzie na panelu operatora wyświetlane są wartości dla obu procesów:

- SV1 i SV2 – (ang. Set Value) wartości zadane
- PV1 i PV2 – (ang. Process Value) wartości mierzone
- MV1 i MV2 – (ang. Manipulated Value) wartości sterowania

Wszystkie wartości przedstawiane są w odpowiednich jednostkach - wartości zadane i mierzone w °C, a sterowania w procentach mocy. Dodatkowo aktualne wartości mierzone i sterowania wyświetlono na wykresach słupkowych.

3. Laboratorium - stanowisko INTECO TCRANE

Zadaniem wyznaczonym przez prowadzącego jest sterowanie wózkiem dźwiga w osi X. Pomiarem jest położenie odczytywane z enkodera, a wartością sterującą jest wypełnienie sygnału PWM, odpowiadającego za prędkość wózka.

3.1. Struktura kodu

Zmieniono okres wywołania fixed scana na 0,1s aby sterowanie mogło szybciej reagować na wydarzenia.

	Label Name	Data Type
1	enc_X	Double Word [Signed]
2	sterowanie_X	FLOAT [Single Precision]
3	krancowka_X	Bit
4	hamulec_X	Bit
5	kierunek_X	Bit
6	zad_X	Double Word [Signed]
7	sterowanie_X_ukm1	FLOAT [Single Precision]

Rys. 3.1. Struktura struct_dzwig

Listing 3.1. Struktura kodu

```
CASE tryb_pracy_dzwig OF
 0:
  DO := 0;
  DO.0 := 1; // channel 1
  HIOEN( TRUE , K0 , DO , 0); // włączenie sczytywania enkodera
  PWM_ON := FALSE;
  dzwig.hamulec_X := HAMULEC_ON;

  // bazowanie i pidy
  ...
END_CASE;

// ograniczenia zakresu pracy dźwigu 0-8000 poza bazowaniem
...

// zmiana ze wzgledu na logike PWM
dzwig.sterowanie_X := 100.0 - dzwig.sterowanie_X;

// ograniczenia dla PWM
IF dzwig.sterowanie_X < 1.0 THEN
  dzwig.sterowanie_X := 1.0;
ELSIF dzwig.sterowanie_X > 100.0 THEN
  dzwig.sterowanie_X := 100.0;
ENDIF;

// wysłanie sterowania do PWM
PWM( PWM_ON , REAL_TO_INT(dzwig.sterowanie_X) , K100 , Y0 );
```

3.2. Bazowanie

Bazowanie wykonano w taki sposób, że wózek z średnią prędkością zjeżdża do osi dźwigu, a kiedy zostanie wykryty przez krańcówkę - zatrzymuje się. Następnie z mniejszą prędkością odjeżdża od krańcowki i w momencie, gdy krańcówka przestaje go wykrywać, ponownie jest zatrzymywany i zerowany jest enkoder. Następnie wózek odsuwa się do pozycji 2000, aby oddalić się od krańcowki.

Listing 3.2. Bazowanie

```
CASE tryb_pracy_dzwig OF
...
20: //bazowanie do lewej krancowki
    bazowanie := TRUE;
    dzwig.sterowanie_X := 50.0;
    dzwig.hamulec_X := HAMULEC_OFF;
    dzwig.kierunek_X := LEWO;

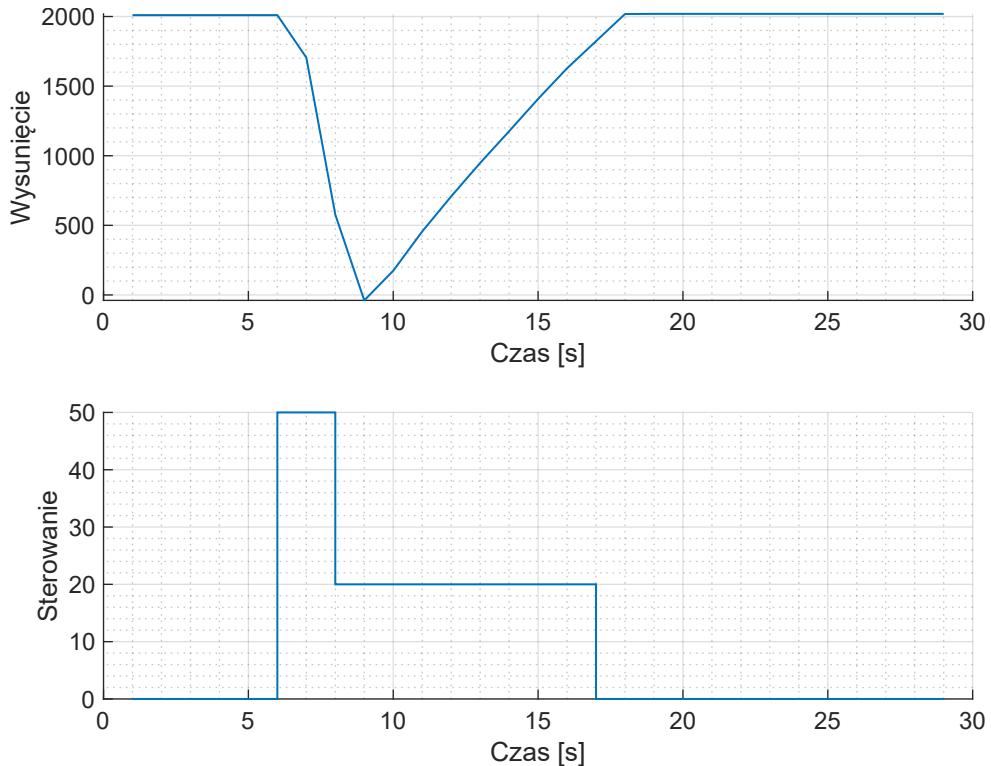
    IF dzwig.krancowka_X THEN
        dzwig.hamulec_X := HAMULEC_ON;
        tryb_pracy_dzwig := 21;
    END_IF;

21: // bazowanie odjazd w prawo do zgasnienia krancowki
    dzwig.sterowanie_X := 20.0;
    dzwig.hamulec_X := HAMULEC_OFF;
    dzwig.kierunek_X := PRAWO;

    IF NOT dzwig.krancowka_X THEN
        dzwig.hamulec_X := HAMULEC_ON;
        DHCMOV( TRUE , 0 , 0 , SD4500 );
        bazowanie := FALSE;
        tryb_pracy_dzwig := 22;
    END_IF;

22: // dojazd na pozycje 2000
    dzwig.sterowanie_X := 20.0;
    dzwig.hamulec_X := HAMULEC_OFF;
    dzwig.kierunek_X := PRAWO;

    IF dzwig.enc_X>2000 THEN
        dzwig.hamulec_X := HAMULEC_ON;
        bazowanie := FALSE;
        tryb_pracy_dzwig := 23; // pusty stan
    END_IF;
...
END_CASE;
```



Rys. 3.2. Wykres pokazujący przebieg bazowania

3.3. Zabezpieczenie zakresu ruchu elementów ruchomych

Po zbazowaniu maksymalny wysięg wózka ograniczony przez części mechaniczne to 8200. Aby nie uszkodzić niczego podczas testowania regulatorów, przyjęto dozwolony zakres ruchu 0-8000, gdzie pozycja 0 oznacza krańcowkę.

Listing 3.3. Zabezpieczenie zakresu ruchu

```
// ograniczenia zakresu pracy dźwigu 0-8000 poza bazowaniem
IF NOT bazowanie THEN
    IF dzwig.kierunek_X = PRAWO AND dzwig.enc_X > 8000 THEN
        dzwig.hamulec_X := HAMULEC_ON;
        dzwig.sterowanie_X := 0.0;
    ELSIF dzwig.kierunek_X = LEWO AND (dzwig.enc_X < 0 OR dzwig.krancowka_X) THEN
        dzwig.hamulec_X := HAMULEC_ON;
        dzwig.sterowanie_X := 0.0;
    END_IF;
END_IF;
```

3.4. Charakterystyka statyczna

Wyznaczenie charakterystyki statycznej układu napędowego z sygnałem sterującym w postaci wypełnienia PWM oraz pomiarem pozycji z enkodera jest niemożliwe ze względu na brak bezpośredniego związku pomiędzy wypełnieniem PWM a pozycją. Pozycja enkodera jest wynikiem całkowania prędkości, dodatkowo nie ma cechy otoczenia, która by temu przeciwdziałała (np. grawitacja), co razem sprawia, że nie jest możliwe uzyskanie zależności statycznej.

3.5. Regulacja położenia wózka

3.5.1. Automat stanów wartości zadanej

Zaimplementowano automat stanów wartości zadanej z czterema zmianami wartości zadanej. Przeskok między stanami następuje co 10s. Przebieg wysuwu zadanego zmieniającego się pod wpływem automatu zamieszczono razem z wykresem regulatora PID (rys. 3.3).

Listing 3.4. Automat stanów wartości zadanej

```

TIM_yzad(PT:= T#10s);

CASE stan_yzad OF
  0:
    TIM_yzad.IN := FALSE;
  10:
    TIM_yzad.IN := FALSE; dzwig.zad_X := 2000;
    stan_yzad := 15;
  15:
    TIM_yzad.IN := TRUE; dzwig.zad_X := 6500;
    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 20;
    END_IF;
  20:
    TIM_yzad.IN := TRUE; dzwig.zad_X := 4000;
    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 25;
    END_IF;
  25:
    TIM_yzad.IN := TRUE; dzwig.zad_X := 500;
    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 30;
    END_IF;
  30:
    TIM_yzad.IN := TRUE; dzwig.zad_X := 9000;
    IF TIM_yzad.Q THEN
      TIM_yzad.IN := FALSE;
      stan_yzad := 0;
    END_IF;
END_CASE;

```

3.5.2. Wykorzystanie własnej implementacji algorytmu PID

Listing 3.5. Własna implementacja algorytmu PID

```

30: // PID autorski
dzwig.hamulec_X := HAMULEC_OFF;

PID3.e_k := DINT_TO_REAL( dzwig.zad_X - dzwig.enc_X)/80.0;
PID3.r_0 := PID3.K_gain*(1.0+(PID3.T_p/(2.0*PID3.Ti))+PID3.Td/PID3.T_p);
PID3.r_1 := PID3.K_gain*((PID3.T_p/(2.0*PID3.Ti))-(2.0*PID3.Td/PID3.T_p)-1.0);
PID3.r_2 := PID3.K_gain*PID3.Td/PID3.T_p;

dzwig.sterowanie_X := PID3.r_2 * PID3.e_km2 + PID3.r_1 * PID3.e_km1
  + PID3.r_0 * PID3.e_k + dzwig.sterowanie_X_ukm1;

PID3.e_km1 := PID3.e_k;
PID3.e_km2 := PID3.e_km1;

```

```

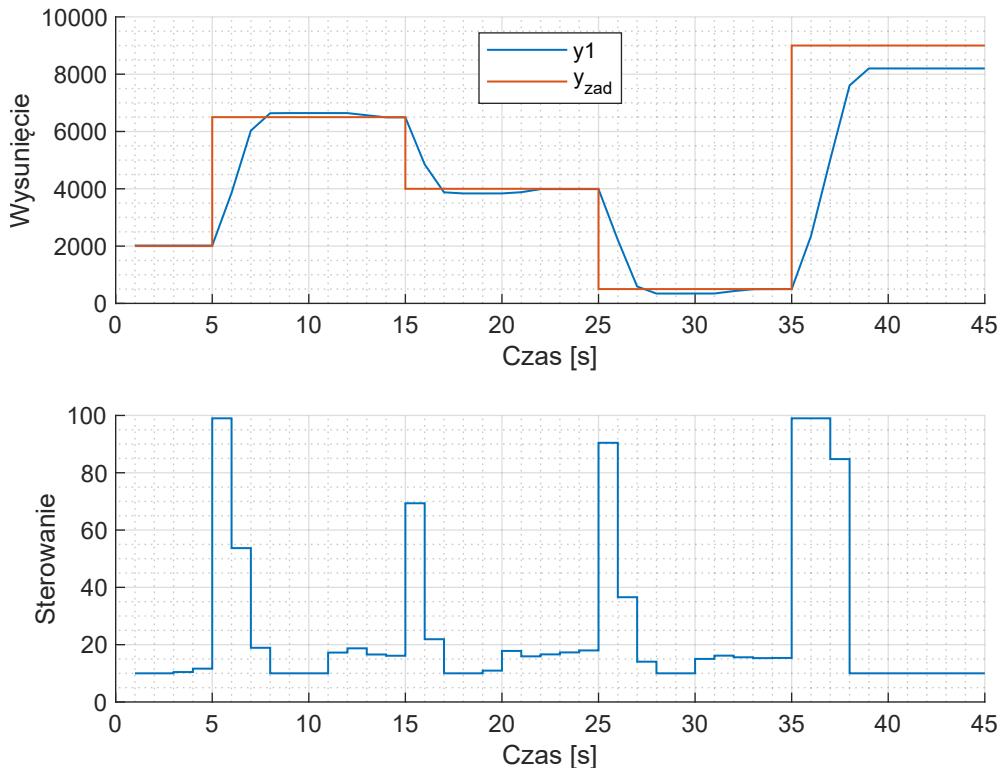
// ograniczenie sterowania z pid i aktualizacja przeszlego sterowania
IF dzwig.sterowanie_X < -99.0 THEN
    dzwig.sterowanie_X := -99.0;
ELSIF dzwig.sterowanie_X > 99.0 THEN
    dzwig.sterowanie_X := 99.0;
ENDIF;
dzwig.sterowanie_X_ukm1 := dzwig.sterowanie_X;

// wybor kierunku bo PWM dziala tylko w jedna strone
IF dzwig.sterowanie_X >= 0.0 THEN
    dzwig.kierunek_X := PRAWO;
ELSE
    dzwig.kierunek_X := LEWO;
    dzwig.sterowanie_X := -dzwig.sterowanie_X;
ENDIF;

```

Regulator wylicza sterowanie w ograniczonym zakresie -99 - 99, a następnie na podstawie znaku wybierany jest kierunek ruchu wózka.

Strojenie regulatora PID dokonano metodą prób i błędów. Przy wykorzystaniu jedynie członu proporcjonalnego występowało zjawisko uchybu, z tego powodu zwiększeno wpływ członu całkującego. Tak osiągnięto nastawy: $K=5$, $T_i=1$, $T_d=0$, $T_p=0,1$ pokazane na rys. 3.3 i uznano, że są zadowalające. Po 35 sekundzie zadziałało ograniczenie ruchu - y_{zad} zostało ustalone na 9000, podczas gdy dozwolony zakres ruchu to 0-8000.



Rys. 3.3. Wykres pokazujący działanie własnej implementacji PID
nastawy: $K=5$, $T_i=1$, $T_d=0$, $T_p=0,1$

3.5.3. Wykorzystanie wbudowanego algorytmu PID

Listing 3.6. Wbudowany algorytm PID

```
40: // pid producenta
```

```

dzwig.hamulec_X := HAMULEC_OFF;
PID_M.SV := DINT_TO_INT(dzwig.zad_X / 80);
PID_M.PV := DINT_TO_INT(dzwig.enc_X / 80);

PID( TRUE , PID_M.SV , PID_M.PV , PID_M.params[0] , sterowanie_pid);

PID_M.MV := sterowanie_pid;

dzwig.sterowanie_X := INT_TO_REAL(sterowanie_pid);
IF dzwig.sterowanie_X >=0.0 THEN
    dzwig.kierunek_X := PRAWO;
ELSE
    dzwig.kierunek_X := LEWO;
    dzwig.sterowanie_X := -dzwig.sterowanie_X;
END_IF;

```

Próbowano wykorzystać dostarczony algorytm PID, jednak działał bardzo nieprzewidywalnie, więc nie podjęto jego strojenia i zamieszczania wykresów.

3.6. Panel operatora

Panel HMI uzupełniono o 3 ekranы pozwalające na uzyskanie niezbędnych informacji na temat sterowanego obiektu.

Pierwszy z nich 3.4, przedstawia animowaną wizualizację obiektu laboratoryjnego, którym jest dźwig wieżowy sterowany w osi X. Dodatkowo pozwala na zadawanie położenia dźwigu w regulowanej osi X wyświetlając tą wartość oraz wartość rzeczywistą w jednostkach enkodera.

Drugi z ekranów 3.5 przedstawia wykres wartości zadanej położenia w osi X (na czerwono) oraz wartości rzeczywistej (na biało). Wartym zauważenia jest fakt, że HMI od Mitsubishi generuje wykresy od lewej do prawej, tzn. najnowsze wartości pojawiają się po lewej stronie, co jest przeciwnieństwem do standardowego przedstawiania wartości w czasie na wykresach.

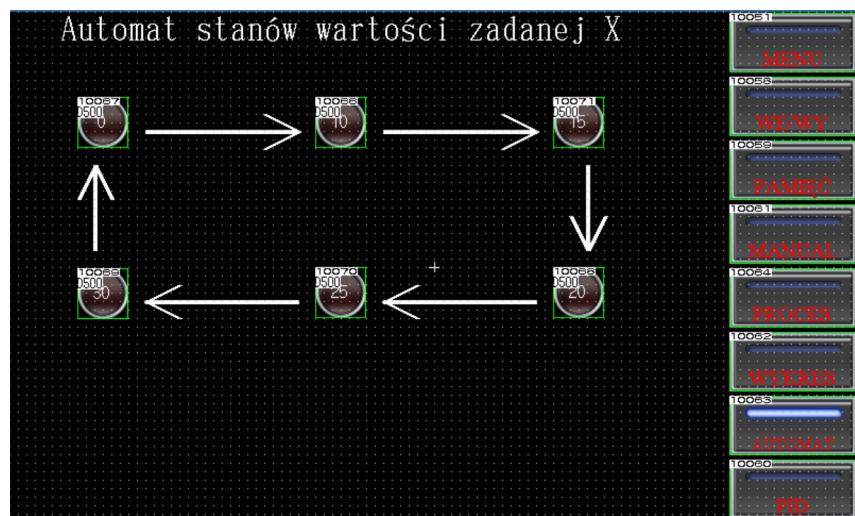
Trzeci z ekranów 3.6 przedstawia automat stanów wartości zadanych regulatora, prezentując za pomocą świecących się lampek/kontrolek w jakim aktualnie stanie znajduje się ten automat.



Rys. 3.4. Ekran HMI pokazujący wartość zadaną i wysunięcie wysunięcia dźwigu



Rys. 3.5. Ekran HMI z wykresem wartości zadanej i wysunięcia czerwony - y_{zad} , biały - y (najnowsze dane idą od lewej strony)



Rys. 3.6. Ekran HMI z automatem stanów wartości zadanej położenia w osi X)