

**Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska**

Systemy mikroprocesorowe w sterowaniu

Sprawozdanie z projektu pierwszego

**Robert Wojtaś
Konrad Winnicki**

Warszawa, 2 grudnia 2018

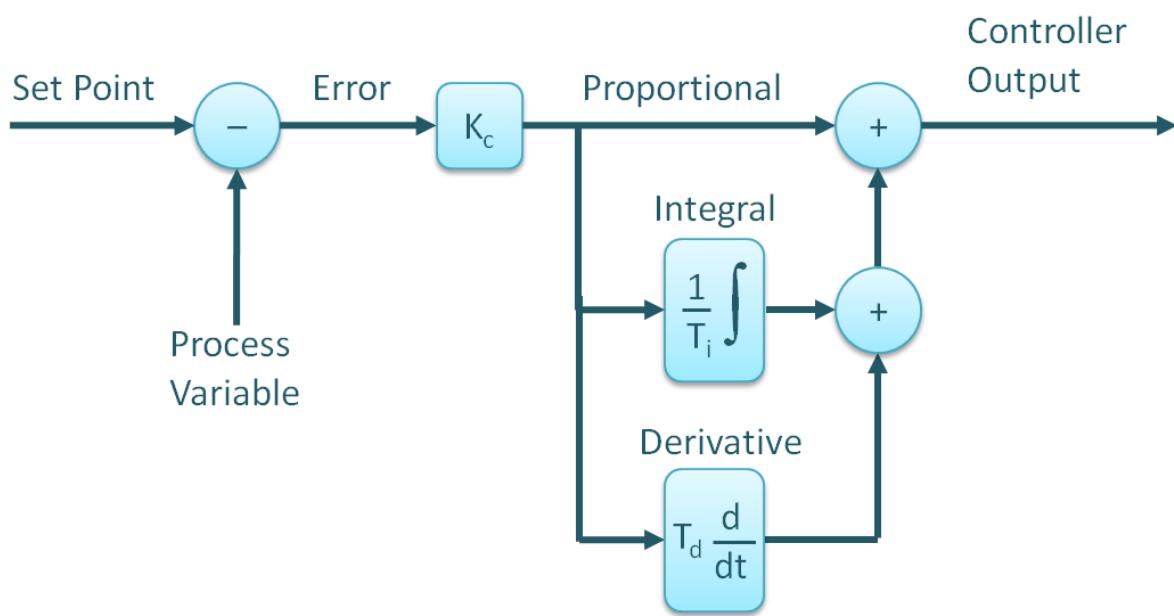
Implementacja regulatora PID

Regulator PID składa się z trzech członów:

- Członu proporcjonalnego P o wzmocnieniu K (na rysunku oznaczone K_C),
- Członu całkującego I o czasie zdwojenia T_I ,
- Członu różniczkowego D o czasie wyprzedzenia T_D

Rysunek poniżej przedstawia strukturę ciągłego w czasie regulatora PID.

Konfigurację jak poniżej po przekształceniu do dziedziny czasu dyskretnego zaimplementowano w trakcie projektu.



Fot.: Struktura zastosowanego regulatora PID
źródło: <http://blog.opticontrols.com/archives/344>

Poszczególne oznaczenia na rysunku przekładają się na zmienne jak poniżej:

- | | | | | |
|---------------------|---|-----------|---|------------------------------------|
| • Set Point | → | y_{ZAD} | - | wartość zadana |
| • Process Variable | → | y | - | wyjście obiektu |
| • Error | → | e | - | uchyb sterowania |
| • Proportional | → | u_P | - | składowa proporcjonalna sterowania |
| • Integral | → | u_I | - | składowa całkowa sterowania |
| • Derivative | → | u_D | - | składowa różniczkowa sterowania |
| • Controller Output | → | u | - | wyjście regulatora |

Po przejściu do dziedziny czasu dyskretnego równanie wyjścia regulatora przedstawiono jako sumę trzech jego składowych, przedstawia się ono następująco:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$

Składowa $u_P(k)$ wyznaczana ze wzoru:

$$u_P(k) = K e(k)$$

Składowa $u_I(k)$ regulatora bez funkcjonalności anti wind-up wyznaczana ze wzoru:

$$u_I(k) = u_I(k - 1) + \frac{K}{T_I} T_p \frac{e(k - 1) + e(k)}{2}$$

Składowa $u_I(k)$ regulatora z funkcjonalnością anti wind-up wyznaczana ze wzoru:

$$u_I(k) = u_I(k - 1) + \frac{K}{T_I} T_p \frac{e(k - 1) + e(k)}{2} + \frac{T_p}{T_V} (u_w(k - 1) - u(k - 1))$$

Składowa $u_D(k)$ wyznaczana ze wzoru:

$$u_D(k) = K T_D \frac{e(k) - e(k - 1)}{T_p}$$

Gdzie:

T_p – Okres próbkowania,

K – Wzmocnienie członu proporcjonalnego P ,

T_I – Czas zdwojenia członu całkującego I ,

T_D – Czas wyprzedzenia członu różniczkowego D

T_V – Parametr anti wind-up,

$e(k)$ – uchyb sterowania w chwili k

Plik nagłówkowy PID.h deklaruje bibliotekę zawierającą:

- Strukturę zawierającą parametry regulatora
- Funkcję PID_init() inicjującą strukturę regulatora
- Funkcję PID_get_control() wyznaczającą nową wartość sterowania

Więcej informacji zawarto w komentarzach w listingu kodu na limonkowym tle poniżej.

```
/*
 * Plik:          PID.h
 * Opis:          Biblioteka implementująca regulator PID z
 *                 funkcjonalnością anti wind-up
 */

typedef struct
{
    float Tp;           // okres próbkowania
    float K;            // wzmacnienie członu P
    float Ti;           // czas zdwojenia członu I
    float Td;           // czas wyprzedzenia członu D
    float Tv;           // parametr anti wind-up;
//                // nieaktywny jeśli mniejszy od zera

    float u_i_past;    // poprzednia wartość składowej całkowania I
    float u_w_past;    // poprzednia wartość sterowania
//                // przekazanego do obiektu

    float u_past;       // poprzednia wartość sterowania regulatora PID
    float e_past;       // poprzedni uchyb sterowania
}PID_type;

/*
 * Inicjacja struktury regulatora PID funkcjonalnością z anti wind-up
 * pid      - wskaźnik na strukturę PID
 * _PID_Tp   - okres próbkowania
 * _PID_K    - wzmacnienie członu proporcjonalnego P
 * _PID_Ti   - parametr członu całkującego I
 * _PID_Td   - parametr członu różniczkowego D
 * _PID_Tv   - parametr anti wind-up;
 *             nieaktywny jeśli mniejszy od zera
 */
void PID_init(PID_type*, float, float, float, float );

/*
 * Wyznaczenie nowej wartości sterowania regulatora PID
 * pid      - wskaźnik na strukturę regulatora
 * e        - uchyb regulacji
 * u_max   - maksymalna wartość sterowania
 * u_min   - minimalna wartość sterowania
 */
float PID_get_control(PID_type*, float, float, float);
```

Plik źródłowy PID.c definiuje funkcje biblioteki regulatora PID:

- PID_init()
- PID_get_control()

Dokładne opisy funkcji zawarte w listingu kodu na limonkowym tle.

```
#include "PID.h"

/*
* Plik:          PID.c
* Opis:          Biblioteka implementująca regulator PID z
*                 funkcjonalnością anti wind-up
*/

/*
*   Inicjacja struktury regulatora PID funkcjonalnością z anti wind-up
* pid           - wskaźnik na strukturę PID
* _PID_Tp       - okres próbkowania
* _PID_K        - wzmacnienie członu proporcjonalnego P
* _PID_Ti       - czas zdwojenia członu całkującego I
* _PID_Td       - czas wyprzedzenia członu różniczkowego D
* _PID_Tv       - parametr anti wind-up;
*                 nieaktywny jeśli mniejszy od zera
*/
void PID_init(PID_type* pid, float _PID_Tp, float _PID_K, float _PID_Ti,
float _PID_Td, float _PID_Tv)
{
    pid->Tp = _PID_Tp;
    pid->K = _PID_K;
    pid->Ti = _PID_Ti;
    pid->Td = _PID_Td;
    pid->Tv = _PID_Tv;

    pid->u_i_past = 0.0;
    pid->u_w_past = 0.0;
    pid->u_past = 0.0;
    pid->e_past = 0.0;
}
```

Funkcja PID_get_control() implementuje regulator PID wedle wzorów podanych na początku tej sekcji.

```
/*
 *   Wyznaczenie nowej wartości sterowania regulatora PID
 *   pid      - wskaźnik na strukturę regulatora
 *   e        - uchyb regulacji
 *   u_max    - maksymalna wartość sterowania
 *   u_min    - minimalna wartość sterowania
 */
float PID_get_control(PID_type* pid, float e, float u_max, float u_min)
{
    float u_p = 0; // składowa sterowania od P
    float u_i = 0; // składowa sterowania od I
    float u_d = 0; // składowa sterowania od D

    // Źródło poniższych wzorów: wzory (2) ze skryptu
    // składowa P równa iloczynowi wzmocnienia K i uchybu sterowania
    u_p = pid->K * e;

    // składowa I powiększana co krok o  $K \cdot T_p \cdot (e_{past} + e) / 2 \cdot T_i$ 
    u_i = pid->u_i_past + pid->K * pid->Tp * (pid->e_past + e) / 2 / pid->Ti;

    // anti wind-up, aktywny jeśli  $T_v > 0$ ;
    // składowa I powiększana dodatkowo co krok o  $T_p \cdot (u_w_{past} - u_{past}) / T_v$ 
    // Źródło: wzór ze skryptu, str. 87
    if( pid->Tv > 0.0 )
        u_i += pid->Tp * (pid->u_w_past - pid->u_past) / pid->Tv;

    // składowa D równa  $K \cdot T_d \cdot (e - e_{past}) / T_p$ 
    u_d = pid->K * pid->Td * (e - pid->e_past) / pid->Tp;

    // wartość sterowania równa sumie składowych;
    // Źródło: wzór (1) ze skryptu
    pid->u_past = u_p + u_i + u_d;

    pid->u_w_past = pid->u_past;
    // nałożenie ograniczeń sterowanie
    if( pid->u_w_past > u_max )
        pid->u_w_past = u_max;
    if( pid->u_w_past < u_min )
        pid->u_w_past = u_min;
    // u_w_past jest ograniczonym u_past -
    // u_w_past to sterowanie przekazane do obiektu

    pid->u_i_past = u_i;
    pid->e_past = e;

    return ( pid->u_w_past );
}
```

Plik konfiguracyjny PID_data.h pojedynczego regulatora zawiera parametry takie jak:

- Okres próbkowania
- Parametr anti wind-up
- Wzmocnienie krytyczne i odpowiadający mu okres oscylacji wyznaczone metodą Zieglera-Nicholsa
- Pierwszy wariant nastaw regulatora PID wg. tabelki Zieglera-Nicholsa
- Drugi wariant nastaw regulatora PID dobranych metodą inżynierską

Plik ten załączany jest przykładowo do pliku main.c

```
//Parametry regulatora PID

// okres próbkowania
#define PID_Tp (1/20.0)
// parametr anti wind-up
#define PID_Tv 8.0f

///////////////////////////////
//parametry regulatora PID wyznaczone metodą Zieglera-Nicholsa
// Wzmocnienie krytyczne
#define PID_Kk 30.0f
// Okres oscylacji
#define PID_Tu (8*PID_Tp)

#define PID_K (0.6*PID_Kk)
#define PID_Ti (0.5*PID_Tu)
#define PID_Td (0.12*PID_Tu)
////////////////////

///////////////////////////////
//parametry regulatora PID wyznaczone metodą inżynierska
/*
#define PID_K 15.00f
#define PID_Ti 3.5f
#define PID_Td 0.040f
*/
///////////////////
```

Eksperymenty i wyniki

Regulator PID

Zadania do wykonania

W tej części projektu zadaniem było strojenie regulatora PID dwiema metodami: Zieglera-Nicholsa oraz metodą inżynierską. Poprzez wykonywanie eksperymentów należało znaleźć odpowiednie wartości poszczególnych nastaw. Po odpowiednim ich wyznaczeniu dla obydwu regulatorów należało porównać wyniki. Następnym zadaniem było zbadanie trajektorii sygnału wyjściowego procesu regulowanego w zależności od parametru związanego z algorytmem anti-windup.

Dobór nastaw regulatora metodą Zieglera-Nicholsa

Metoda Zieglera-Nicholsa oparta jest o pomiar parametrów oscylacji. Eksperymenty rozpoczynamy od wyłączenia członów I oraz D regulatora. Wzmocnienie członu P zwiększamy do momentu osiągnięcia wzmocnienia krytycznego – takiego, przy którym układ znajduje się na granicy stabilności czyli oscyluje ze stałą amplitudą. Po wyznaczeniu wzmocnienia krytycznego, potrzebujemy drugiego parametru do dalszej pracy. Parametrem tym jest okres drgań oscylacji. Posiadając takie dwie wartości można wyznaczyć parametry dla algorytmów P, PI oraz PID korzystając z tabelki zamieszczonej w skrypcie.

Ku – wzmocnienie krytyczne

Tu – okres oscylacji

Regulator	K	Ti	Td
P	0.5Ku	-	-
PI	0.45Ku	Tu/1.2	-
PID	0.6Ku	Tu/2.0	Tu/8

Wyniki

Poszukiwanie wzmocnienia krytycznego i okresu drgań

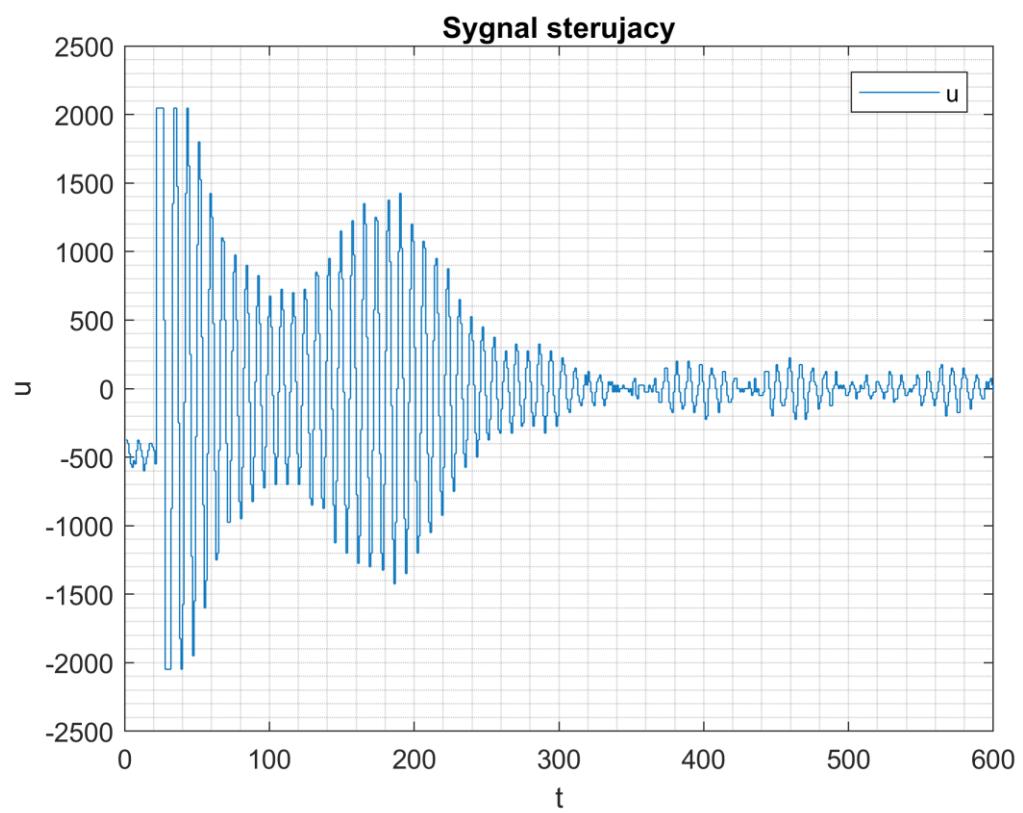
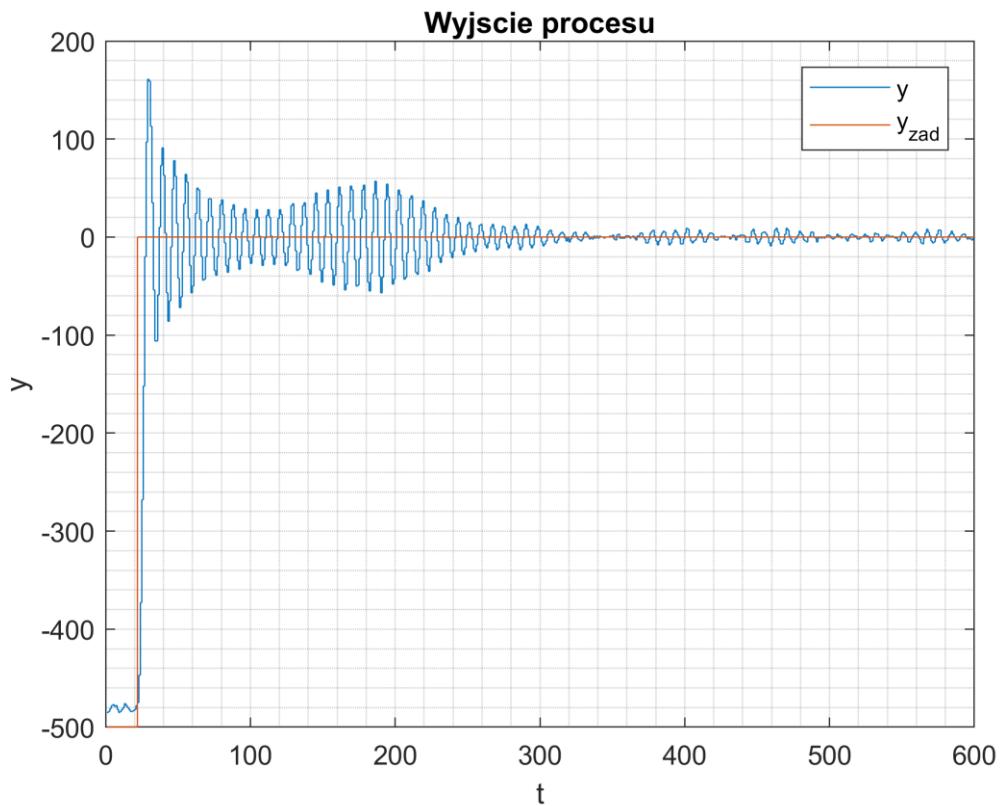
Ważnym aspektem podczas eksperymentów było dla nas zapewnienie wzmocnienia krytycznego, przy którym sygnał sterujący mieścił się w ograniczeniach.

Były one wykonywane poprzez skok z wartości -500 do wartości 0. Po skoku czekaliśmy na ustalenie sygnałów i porównywaliśmy otrzymane przebiegi.

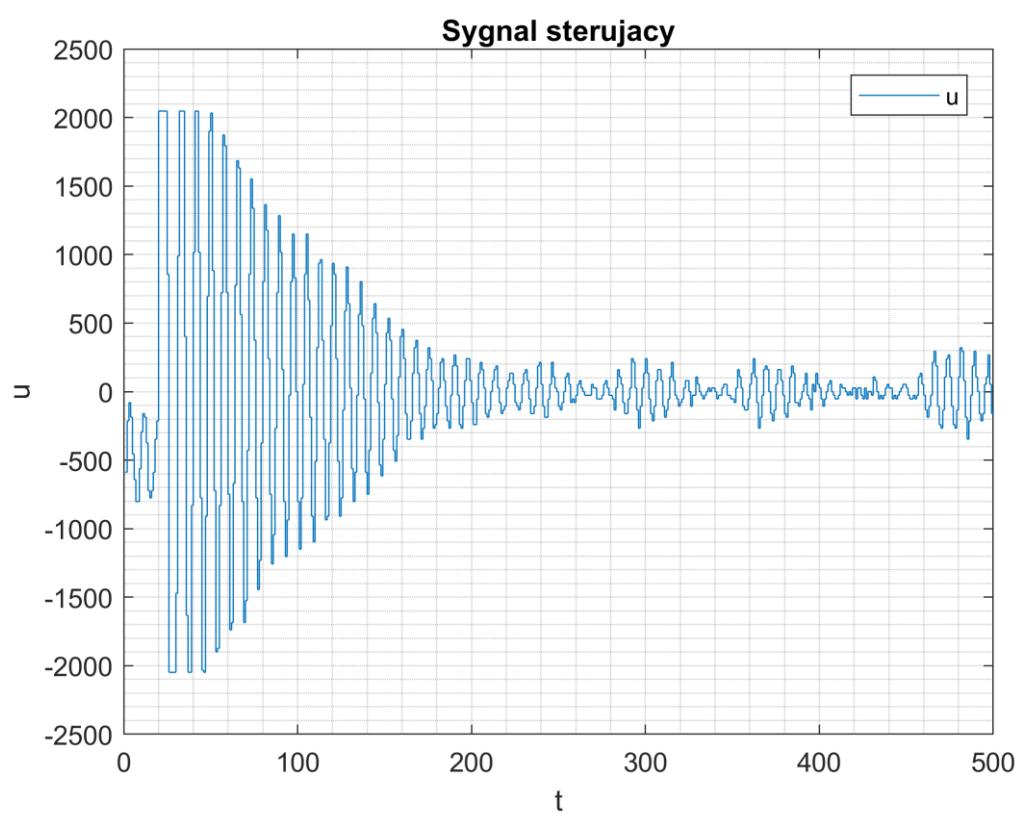
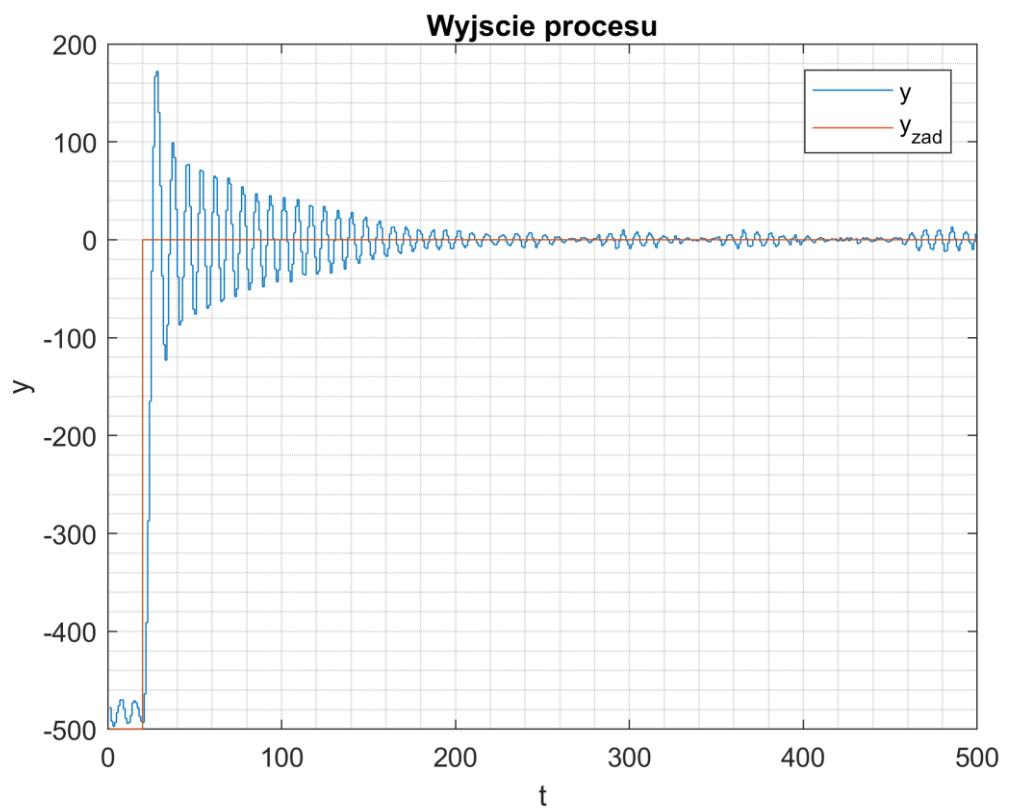
Podczas eksperymentów zauważaliśmy, że wprowadzenie obiektu w stałe oscylacje jest możliwe jedynie wtedy, gdy zadamy wzmocnienie powodujące uderzanie sygnału sterującego w ograniczenia przez cały czas trwania próby. Uznaliśmy, że wybór takiego wzmocnienia byłby błędem i szukaliśmy wartości zapewniającej oscylacje najbardziej zbliżonych do stałych.

Co więcej sygnały co pewien okres czasu zmieniały swoją amplitudę dlatego staraliśmy się wybrać rozwiązanie, przy którym przez większość czasu drgania utrzymywały się na zbliżonym poziomie.

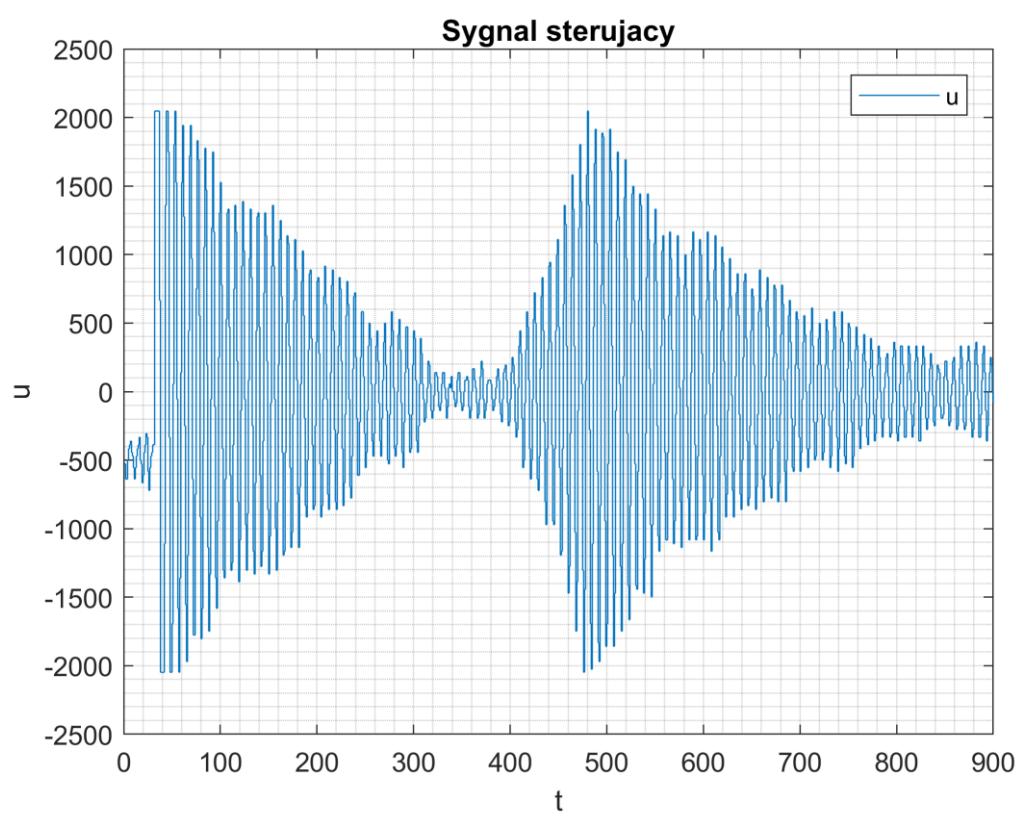
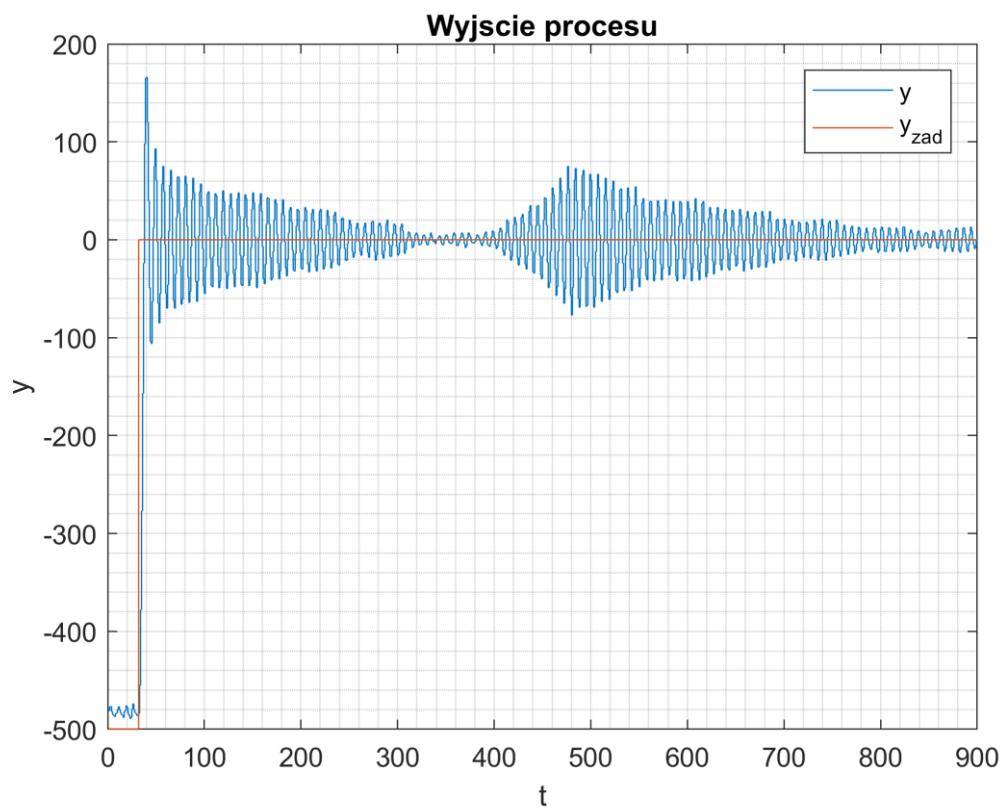
$$K = 25$$



$$K = 26.75$$

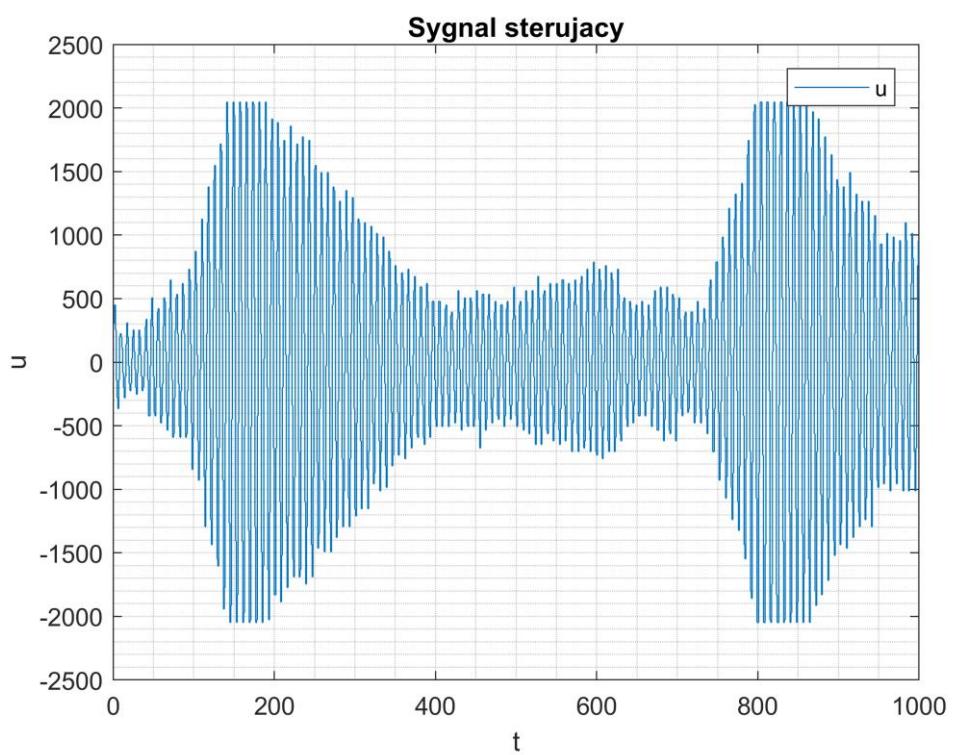
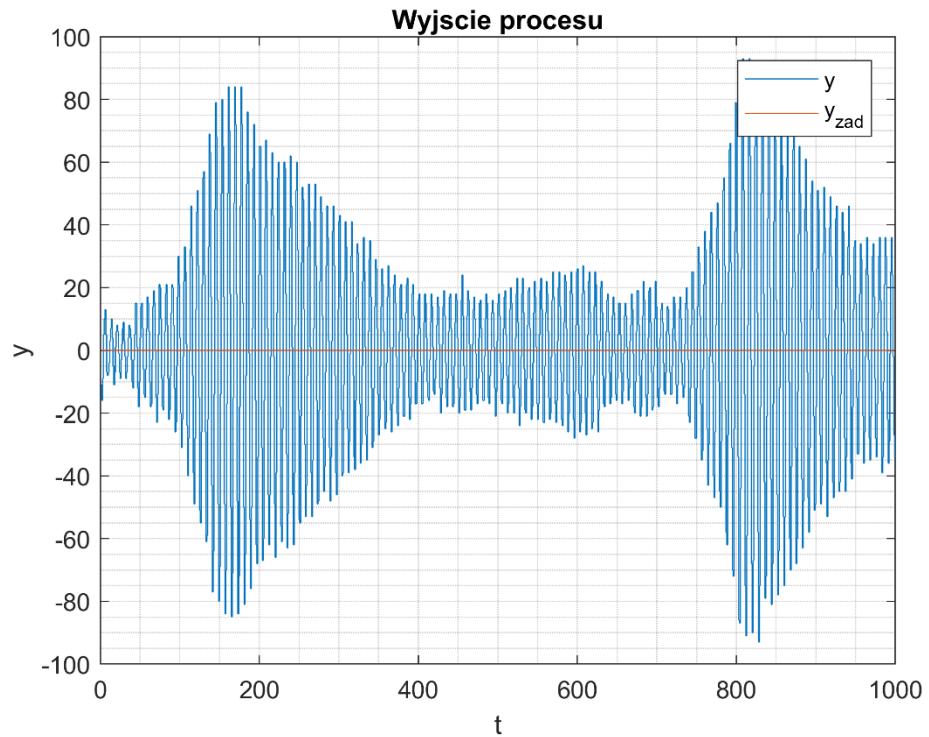


$$K = 27.75$$

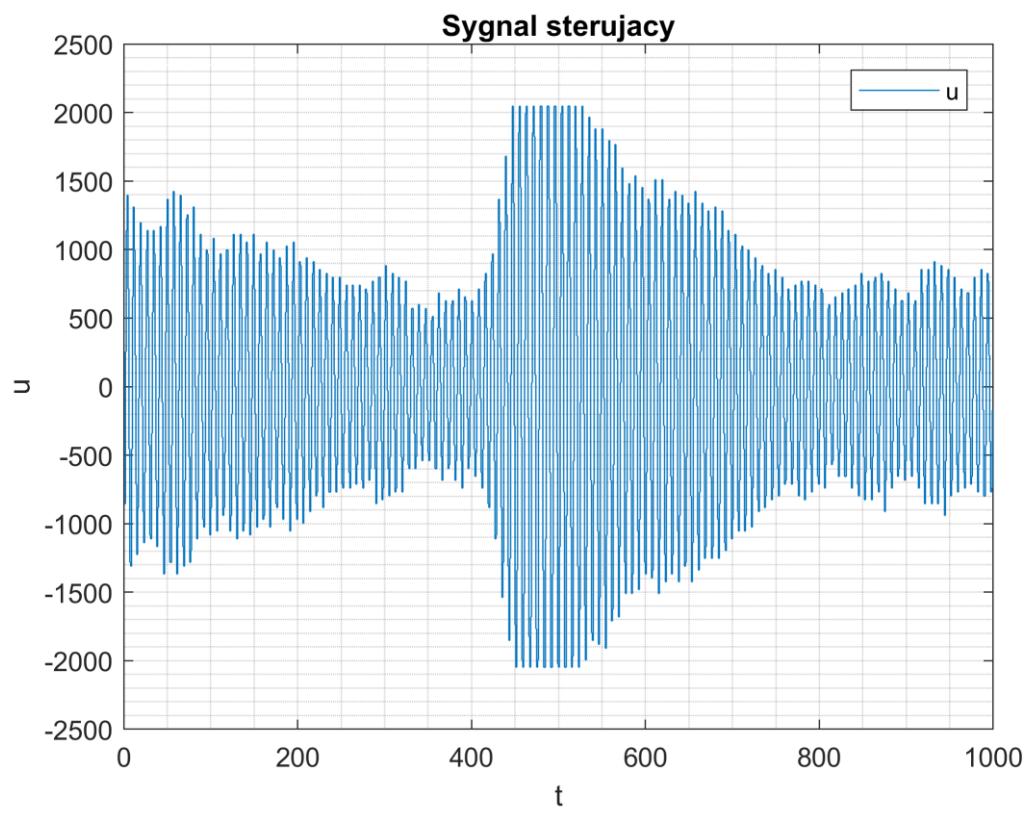
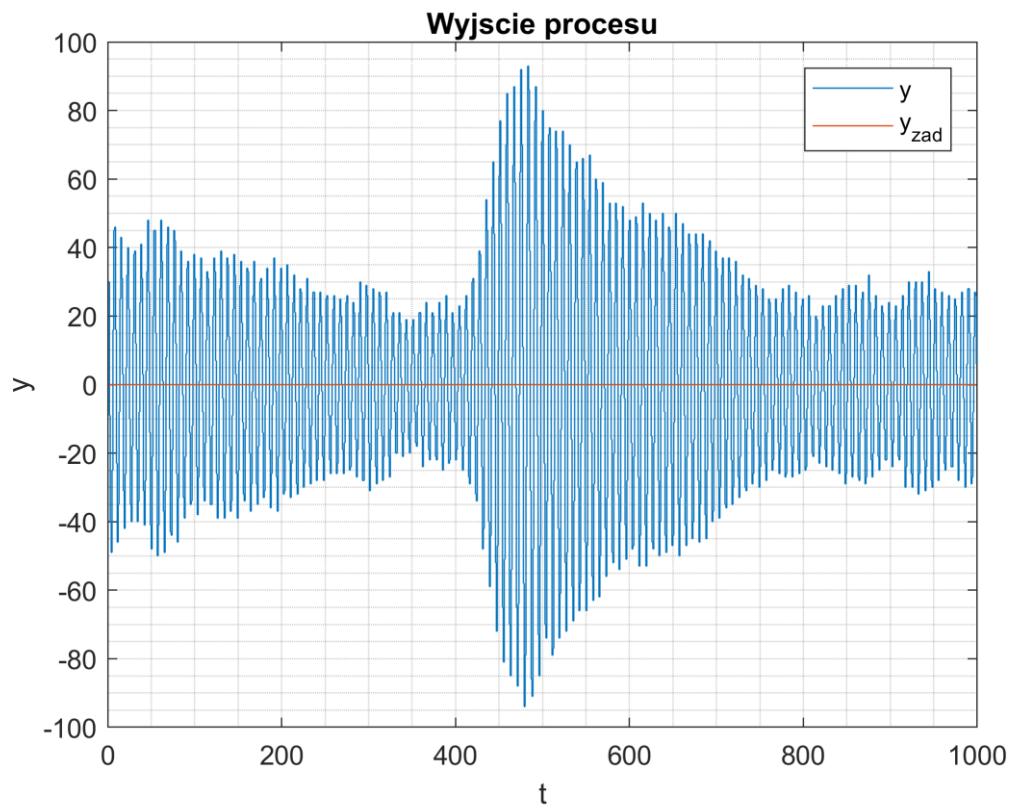


Po wykonaniu skoku wartości zadanej, dało się zauważyc, że niektóre przebiegi wyglądają bardziej obiecująco od innych. Z tego powodu kolejne wykresy przedstawić będą stan ustalony tzn. taki, w którym sygnał oscyluje i zmiany jego amplitudy również występują co pewien okres czasu.

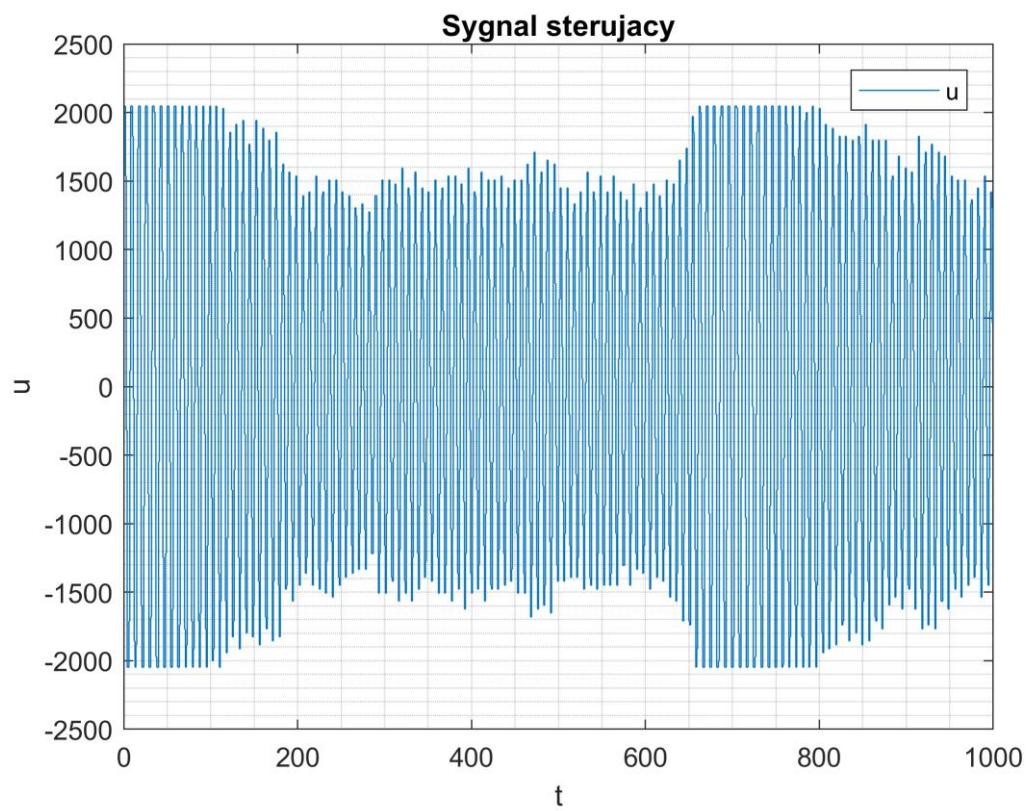
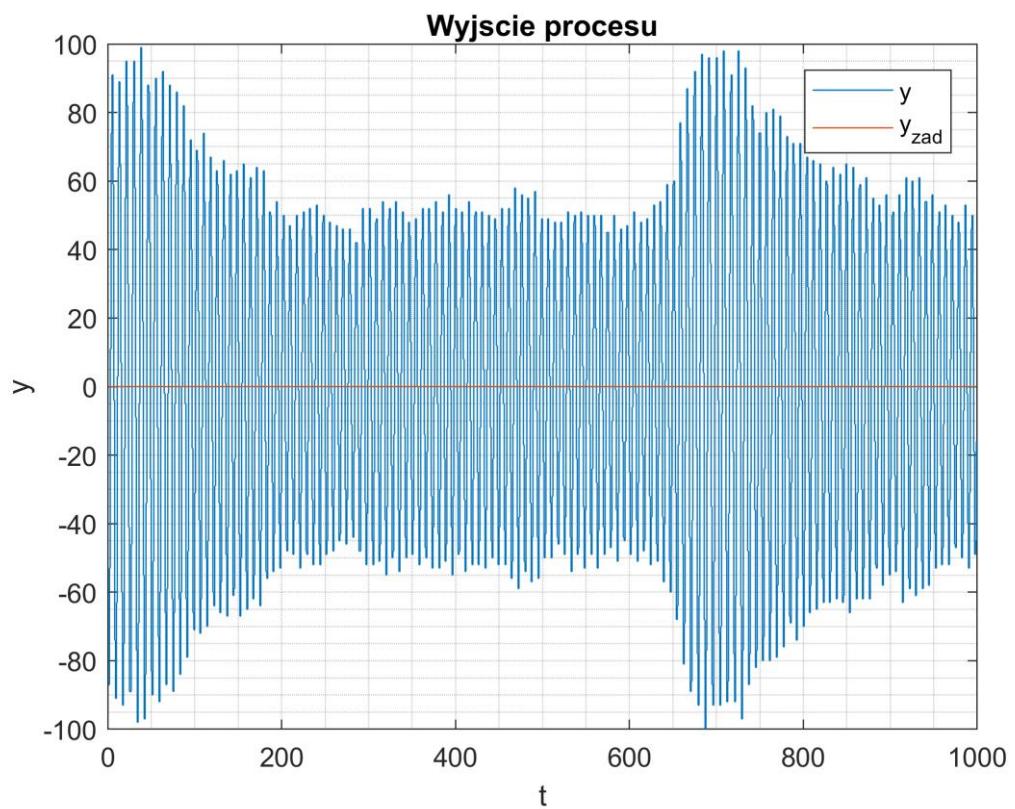
$$K = 28.15$$



$$K = 28.5$$



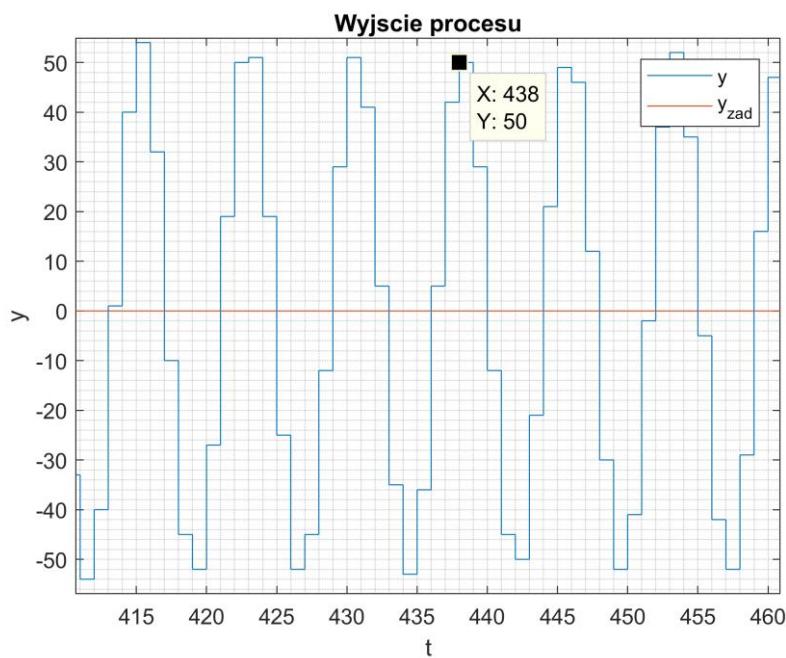
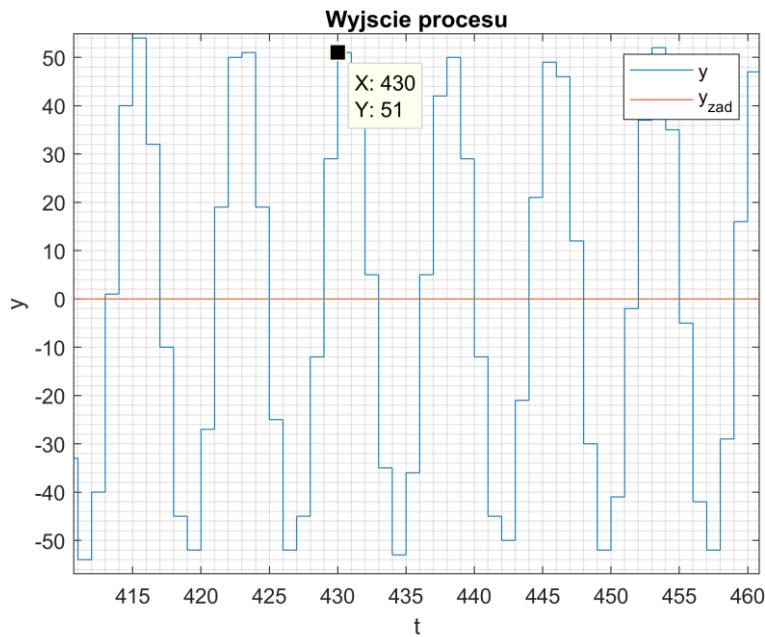
$K = 30$



Na powyższych wykresach dobrze widać opisane wcześniej zjawiska. Sygnał sterujący faktycznie uderza przez chwilę w ograniczenia, jednak we wszystkich przypadkach obserwujemy, że z czasem stabilizuje się i mieści w dozwolonych granicach. Kolejną rzeczą godną obserwacji jest zmiana amplitudy sygnałów, szczególnie dobrze widoczna przy demonstracji stanów ustalonych.

Po wykonaniu eksperymentów dla różnego K, za wzmocnienie krytyczne postanowiliśmy uznać $K = 30$. W tym przypadku, przez większość czasu obserwujemy drgania zbliżone do stałych, o podobnej amplitudzie, co więcej przez zdecydowaną większość czasu sygnał sterujący mieści się w dopuszczanych granicach, uderzając w ograniczenia przez chwilę, co pewien okres czasu. Uznaliśmy, że są to drgania pożądane i potrzebne do wyznaczenia dalszych nastaw regulatora.

Po wybraniu wzmocnienia krytycznego należało sprawdzić okres drgań w stanie skrajnej stabilności:



Z wykresów wynika, że okres drgań do 8 próbek co przy częstotliwości 20Hz daje 0.4 sekundy.

Otrzymane wartości wzmacnienia krytycznego oraz okresu drgań posłużyły do wyznaczenia nastaw regulatora PID, zgodnie z wyżej zacytowaną tabelką.

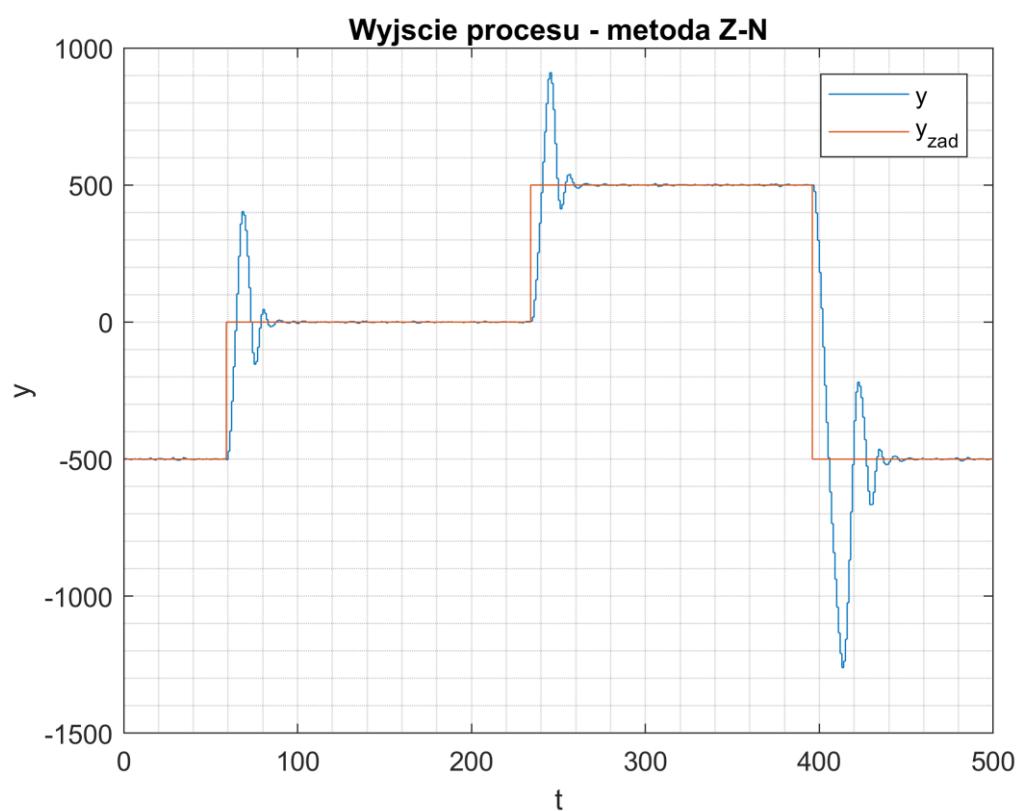
$$K_u = 30.0$$

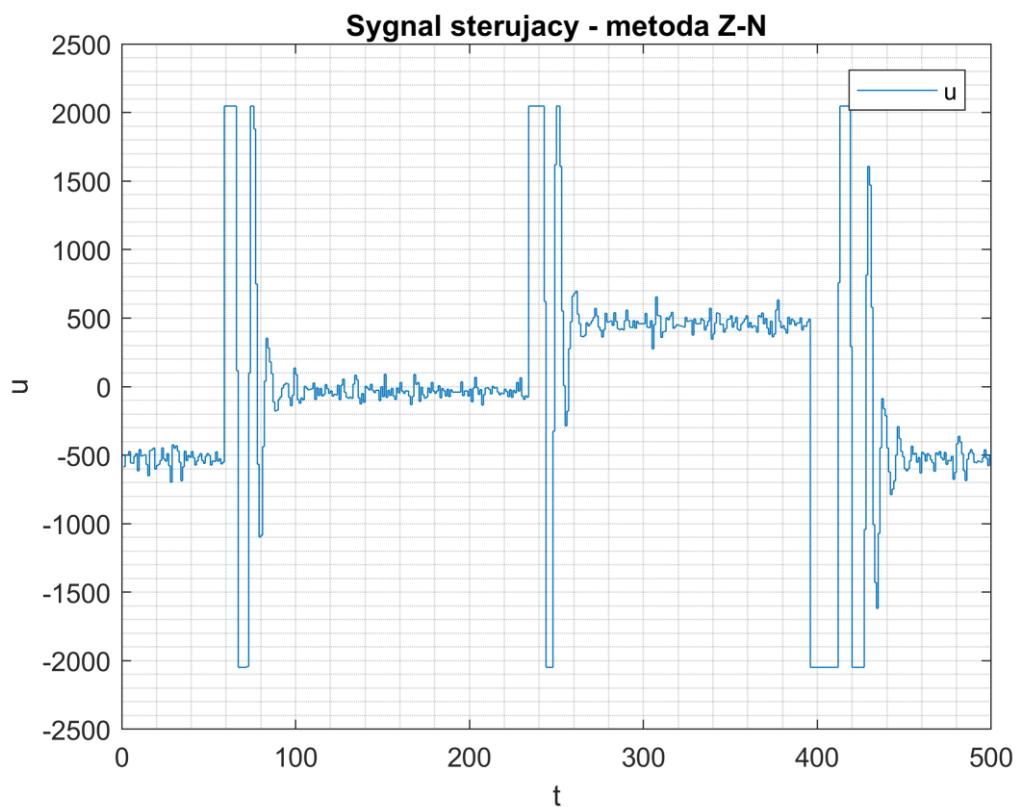
$$T_u = 0.4$$

$$K_p = 18.0$$

$$T_i = 0.2$$

$$T_d = 0.05$$





Otrzymany regulator daleki jest od ideału. „Na oko” widać duże przeregulowanie oraz sygnał sterujący, który nie mieści się w ograniczeniach przez dłuższą chwilę po skoku wartości zadanej.

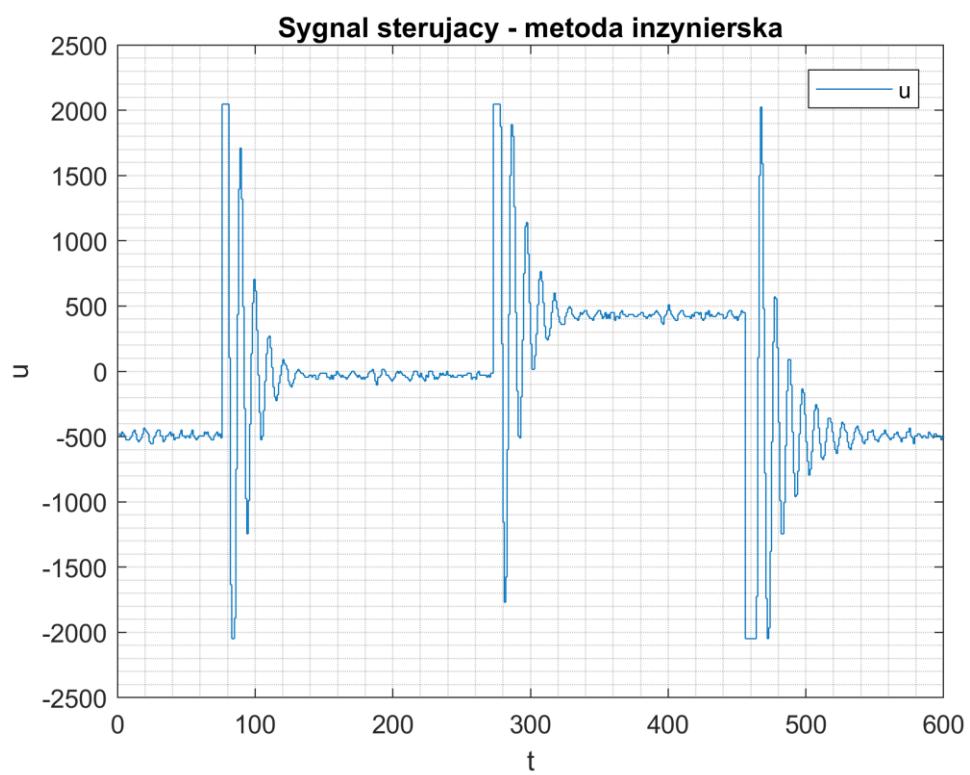
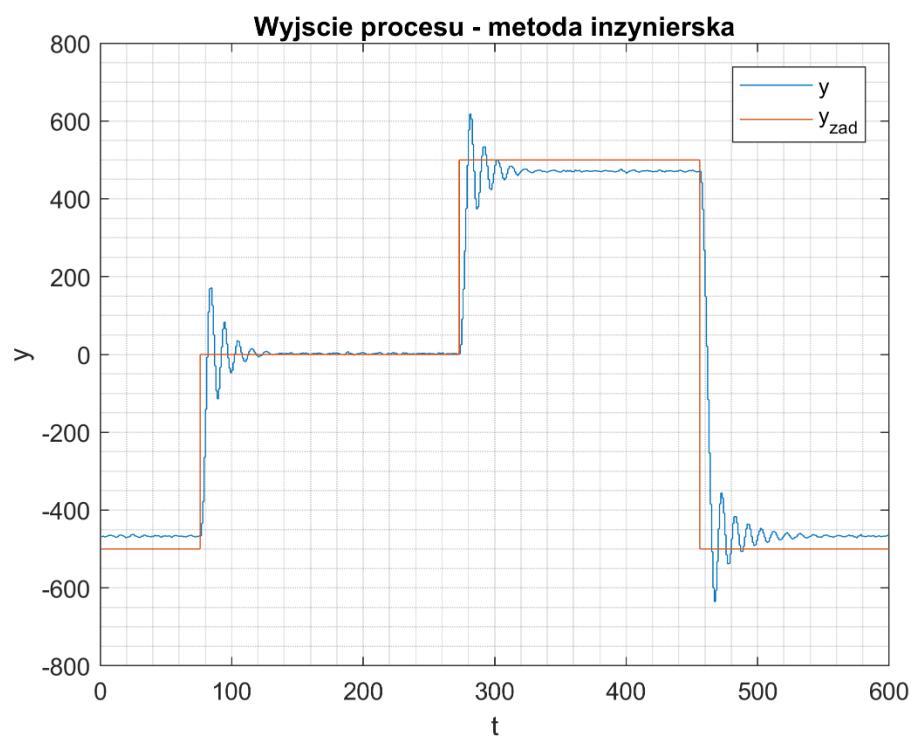
Metoda Zieglera-Nicholsa jest dobrą metodą, jest prosta i stosunkowo szybko pozwala uzyskać dobre efekty. Powinna być ona jednak stosowana jako wstęp do strojenia, gdyż trzymanie się schematu nie zawsze da świetne rezultaty. W naszym projekcie uznaliśmy powyższy wynik za końcowy i nie staraliśmy się go w żaden sposób poprawiać ponieważ następnym zadaniem było wyznaczenie nastaw metodą inżynierską. Zostawiając powyższy wynik, porównanie metod dobrze uwidocznii konieczność jego poprawy.

Dobór nastaw regulatora metodą „inżynierską”

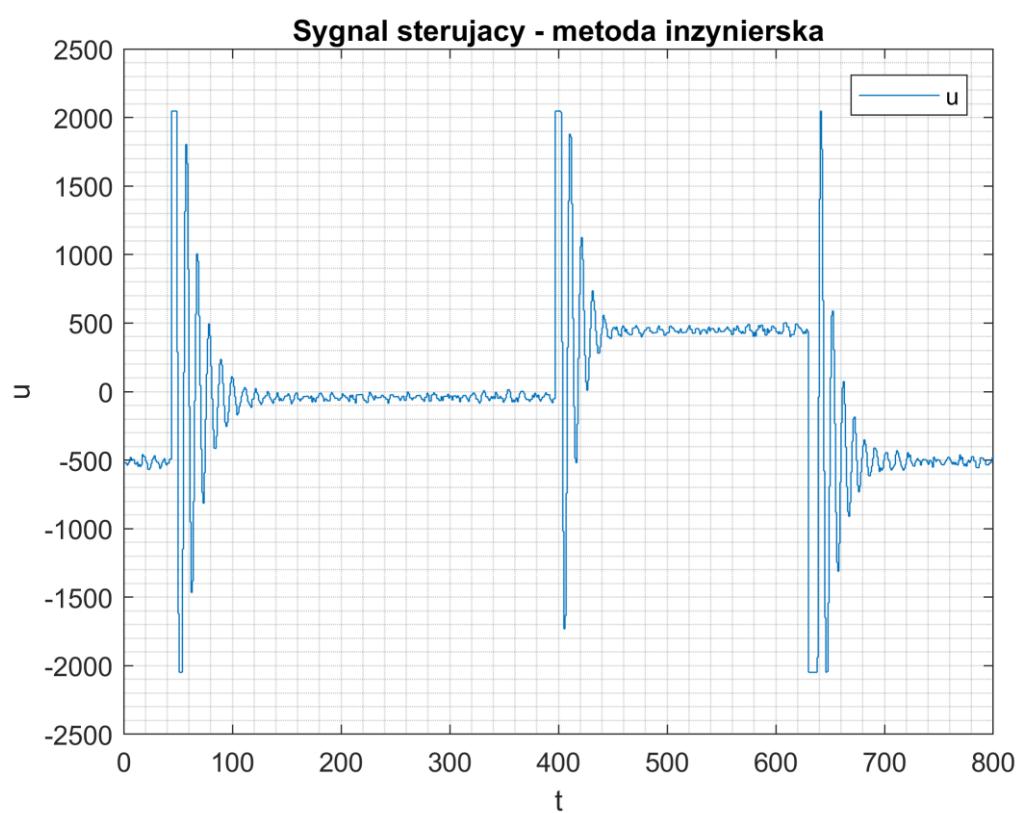
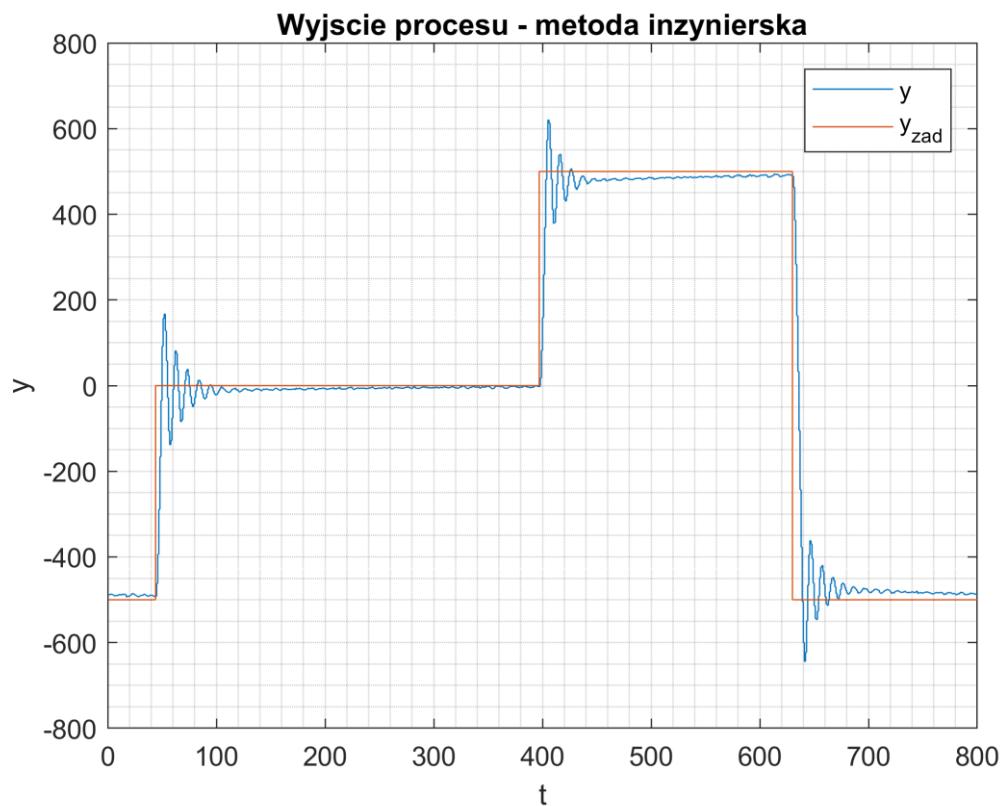
Metoda ta jest podzielona na trzy etapy – dobór parametru K , dobór parametru T_I oraz dobór parametru T_D . Ponieważ w poprzednim punkcie znaleźliśmy już dobre wzmacnienie krytyczne, postanowiliśmy go nie zmieniać i przyjęliśmy za wzmacnienie naszego regulatora wartość 15, czyli połowę wzmacnienia krytycznego. Następnym krokiem był dobór parametrów regulatora T_I oraz T_D . Odbywało się to metodą prób i błędów w celu uzyskania jak najlepszych rezultatów.

Wyniki

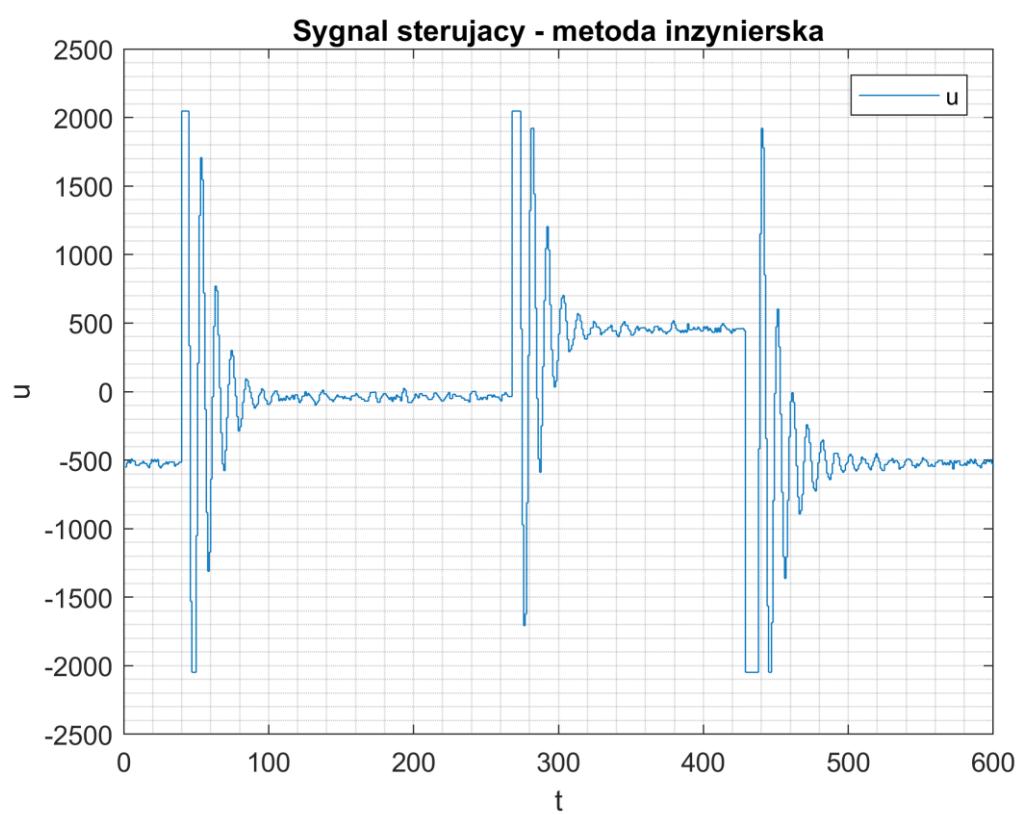
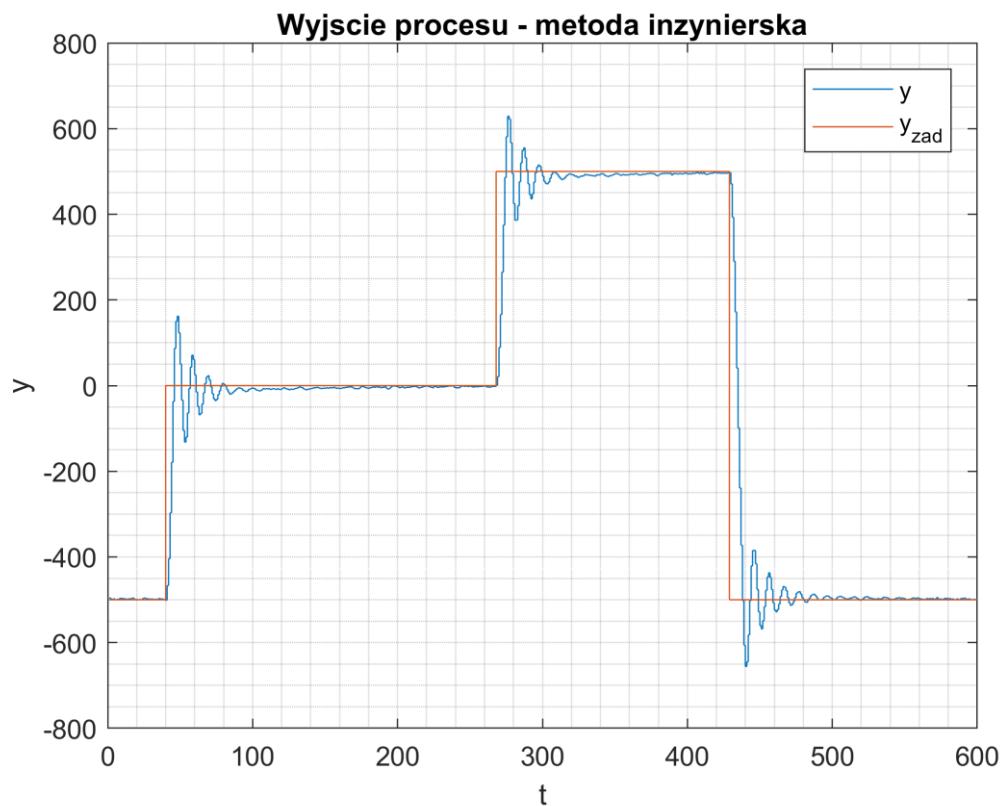
Regulator P: K = 15



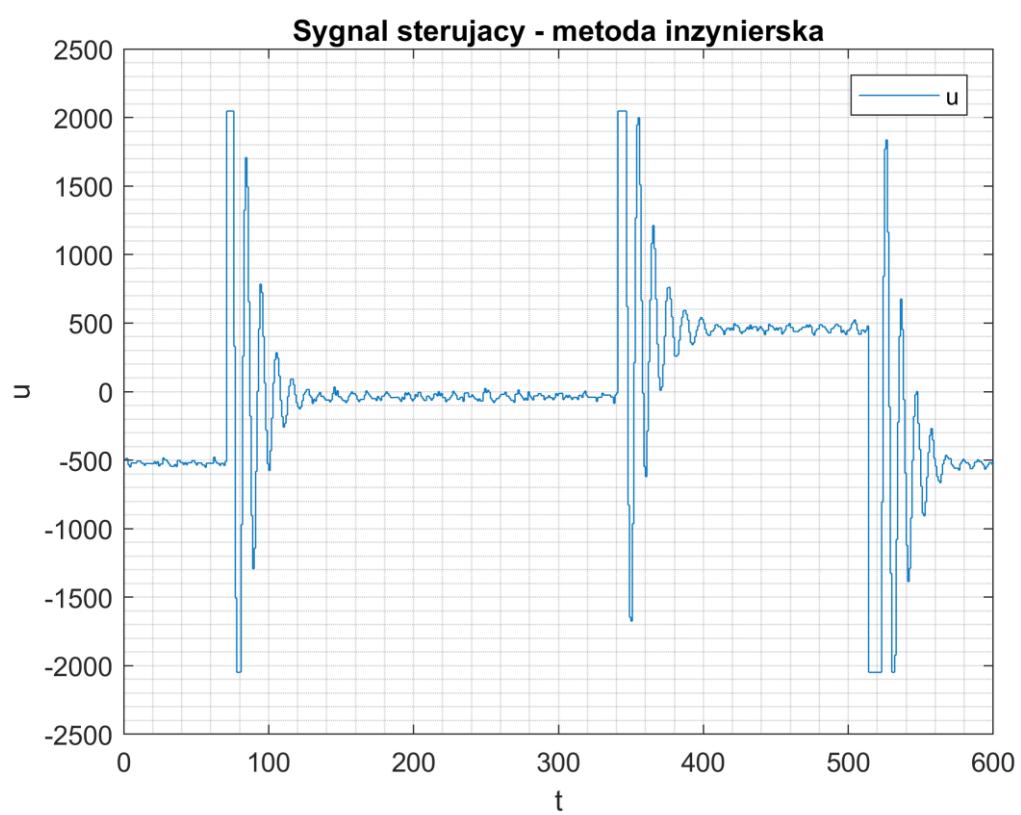
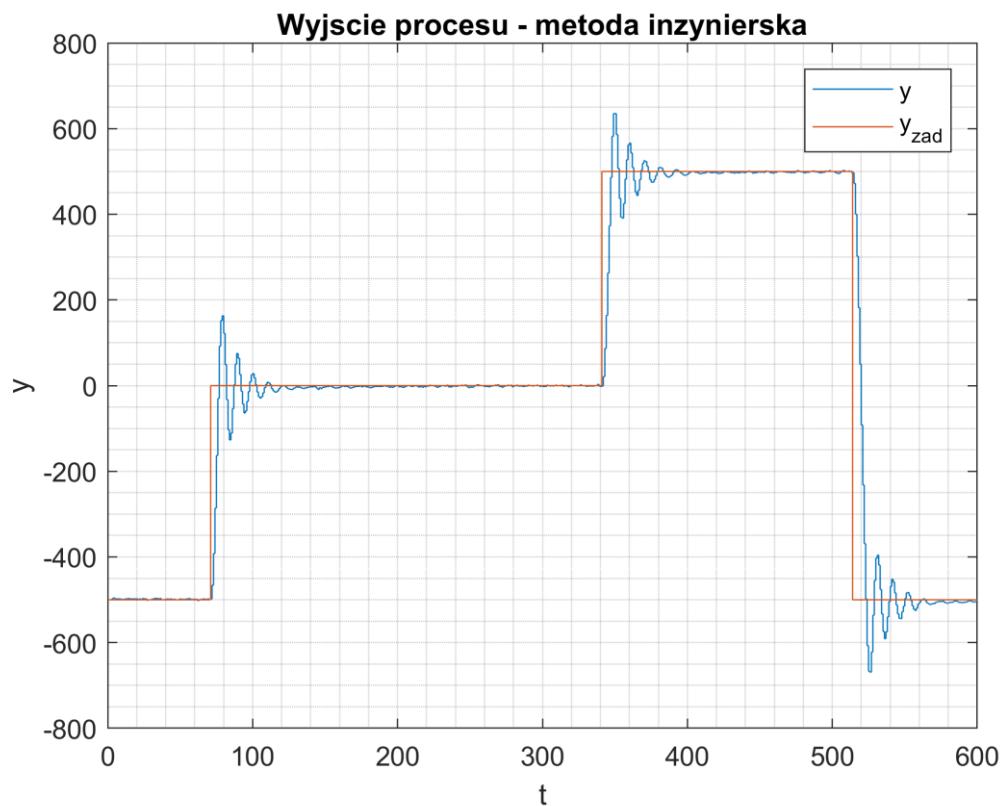
Regulator PI: $K = 15$, $T_i = 10$



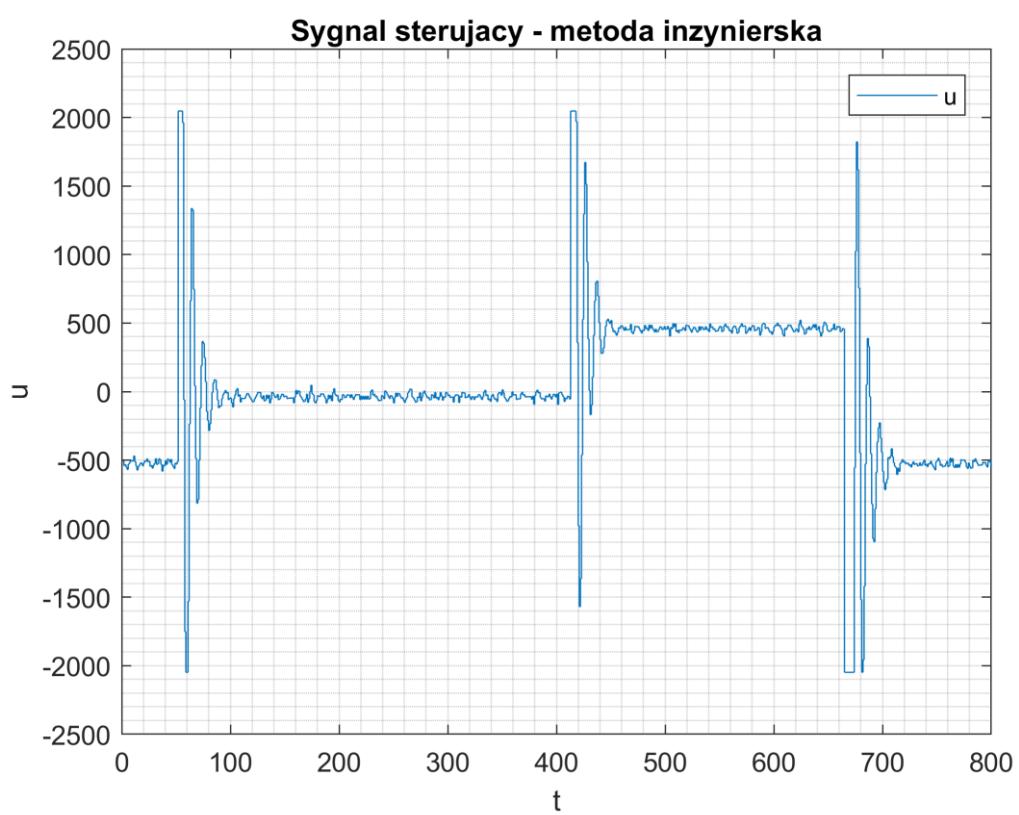
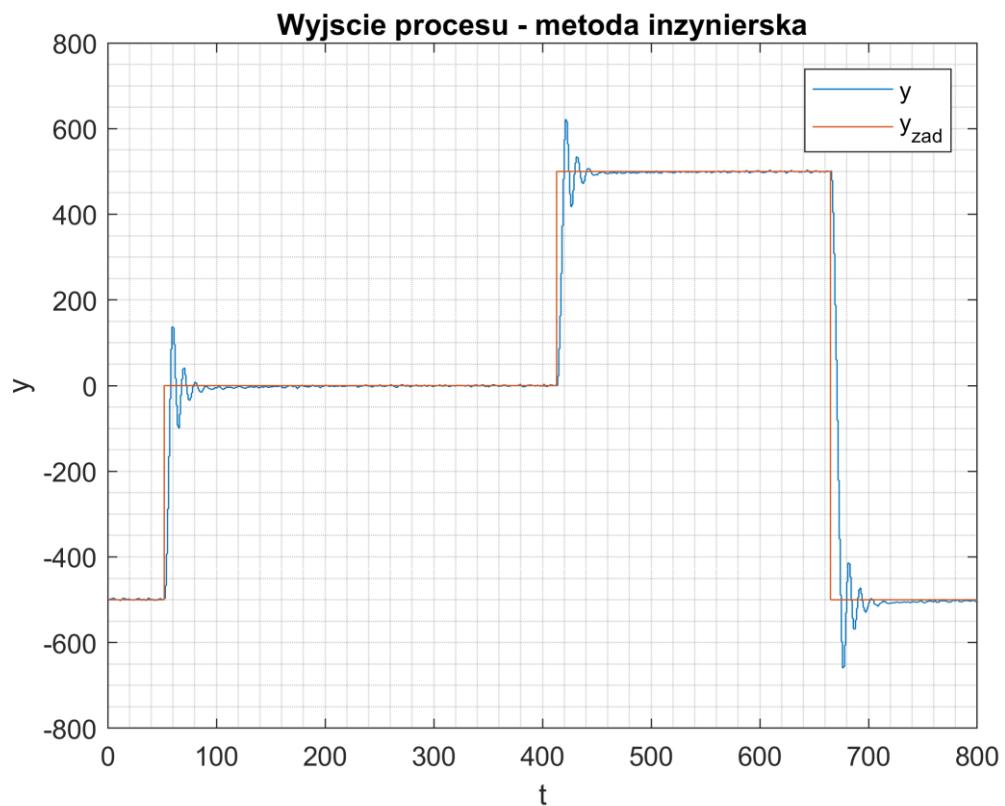
Regulator PI: $K = 15$, $T_i = 5$



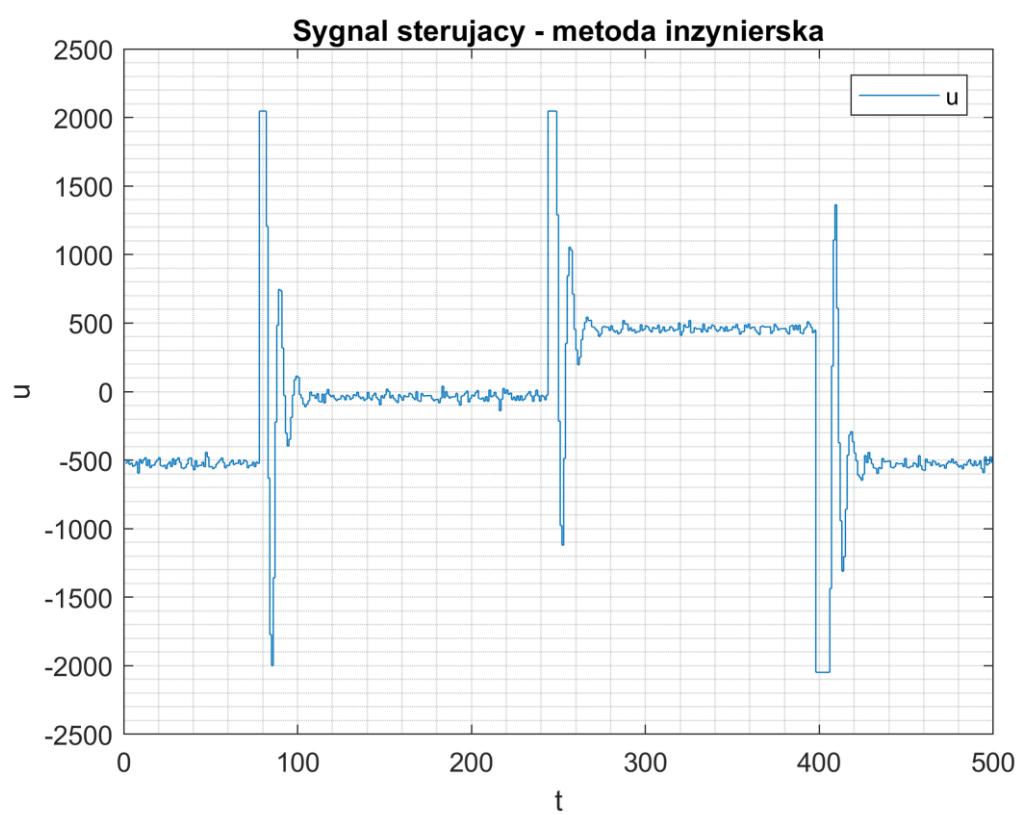
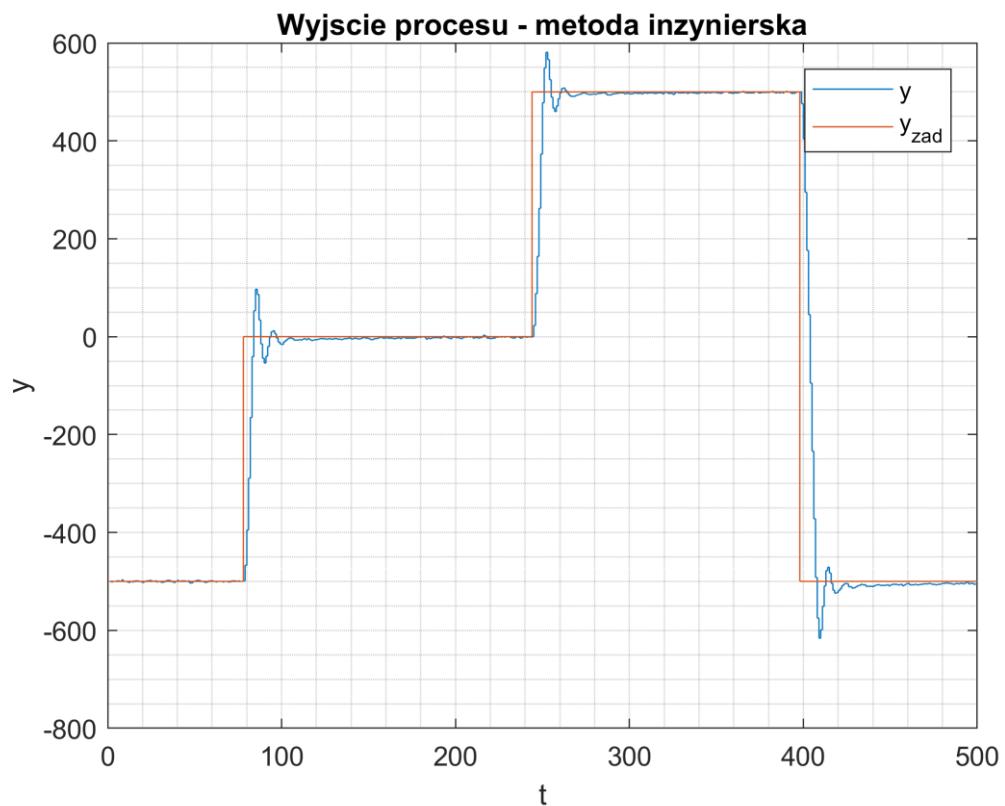
Regulator PI: $K = 15$, $T_i = 3.5$



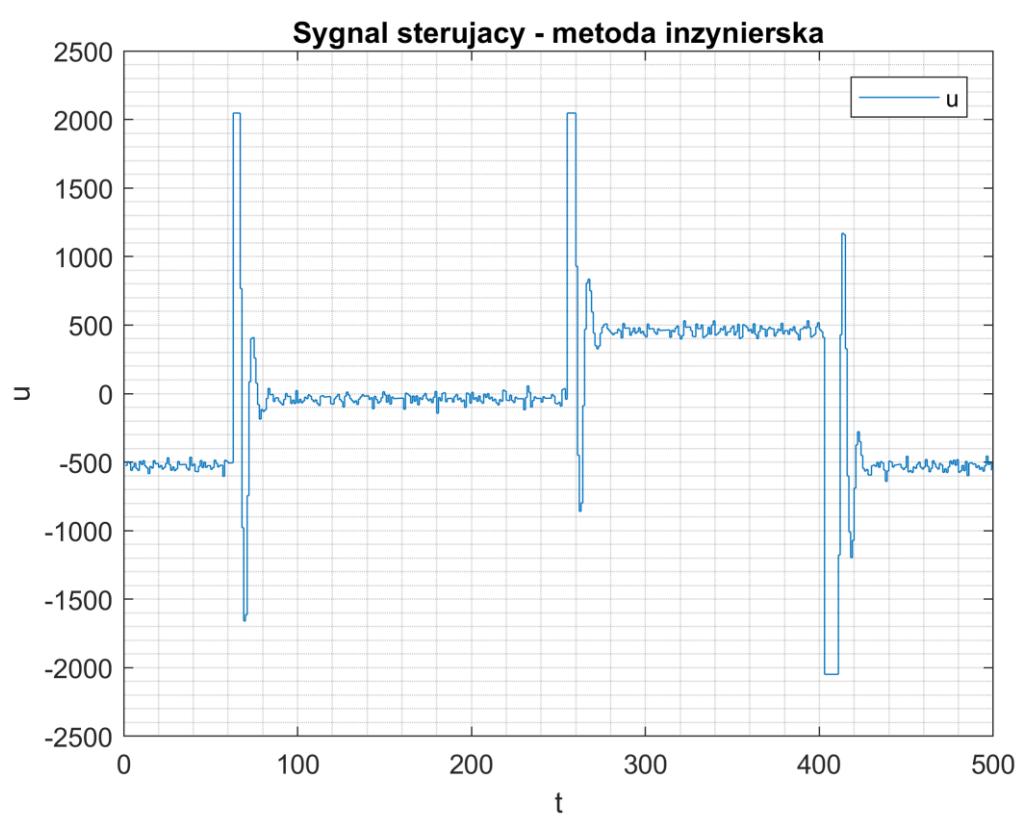
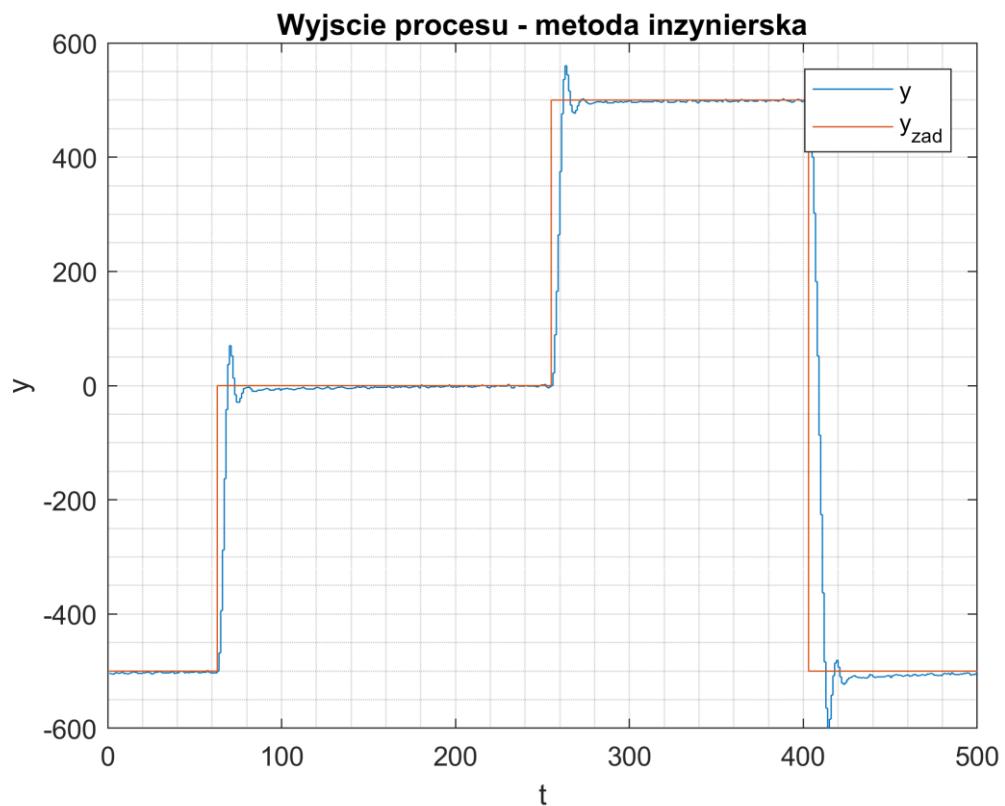
Regulator PID: $K = 15$, $T_i = 3.5$, $T_d = 0.01$



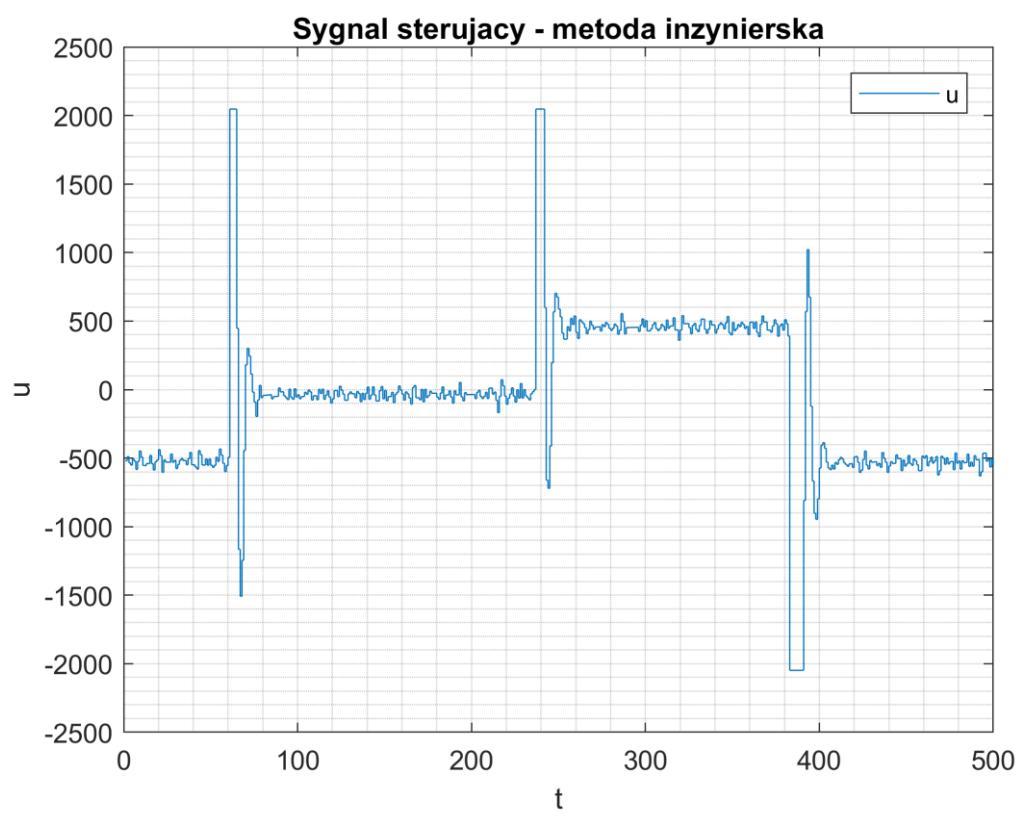
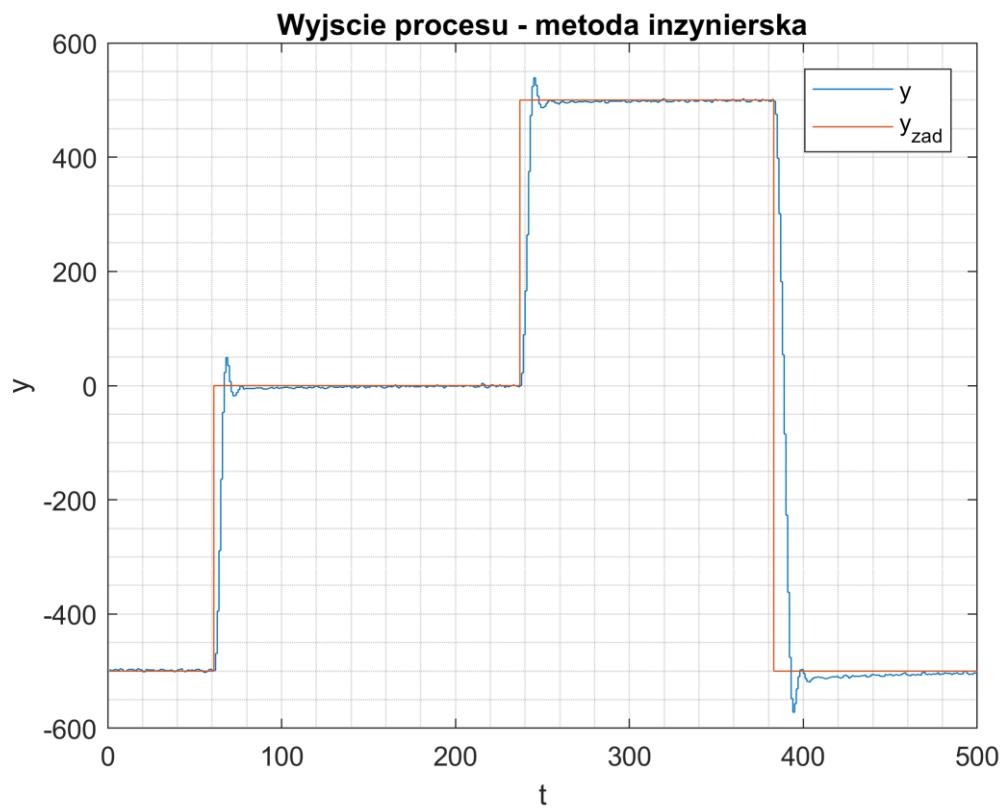
Regulator PID: $K = 15$, $T_i = 3.5$, $T_d = 0.03$



Regulator PID: $K = 15$, $T_i = 3.5$, $T_d = 0.04$



Regulator PID: $K = 15$, $T_i = 3.5$, $T_d = 0.05$



Eksperymenty z parametrem T_i wykonywane były od stosunkowo wysokiej wartości równej 10. Wyższa wartość czasu zdwojenia powoduje mniejszy wpływ całkowania na regulację. Można zauważyc, że nie dostajemy zadowalającego efektu, jednak dodanie członu całkującego poprawia jakość regulacji.

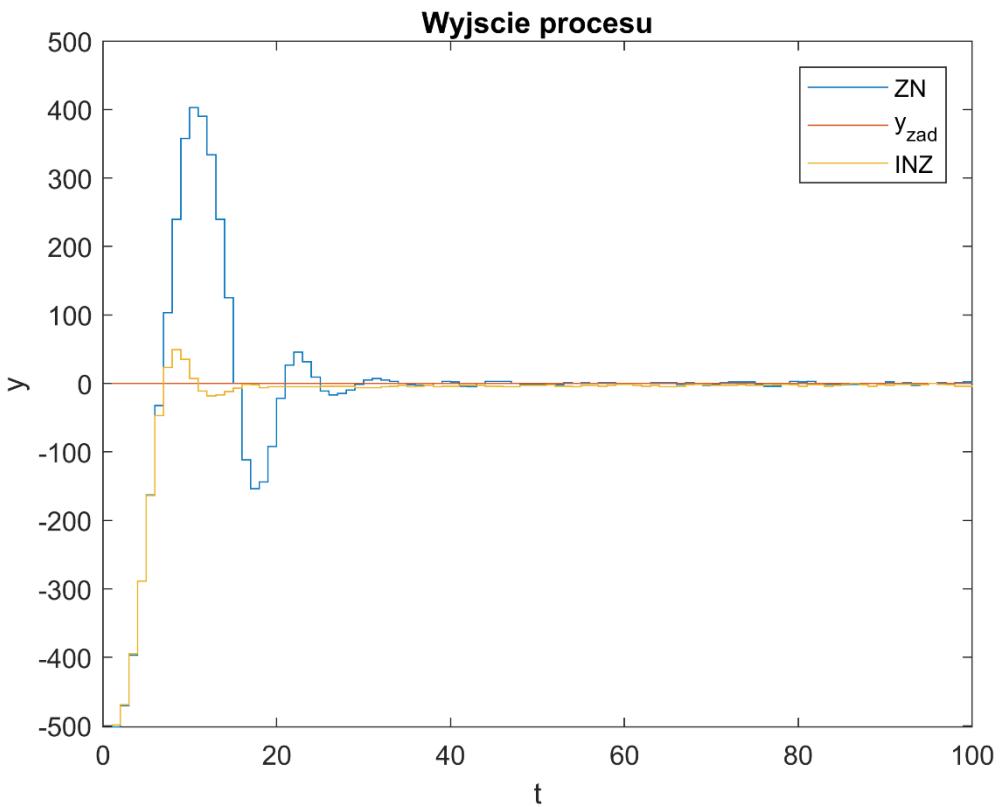
Ostatecznie wybrałem $T_i = 3.5$.

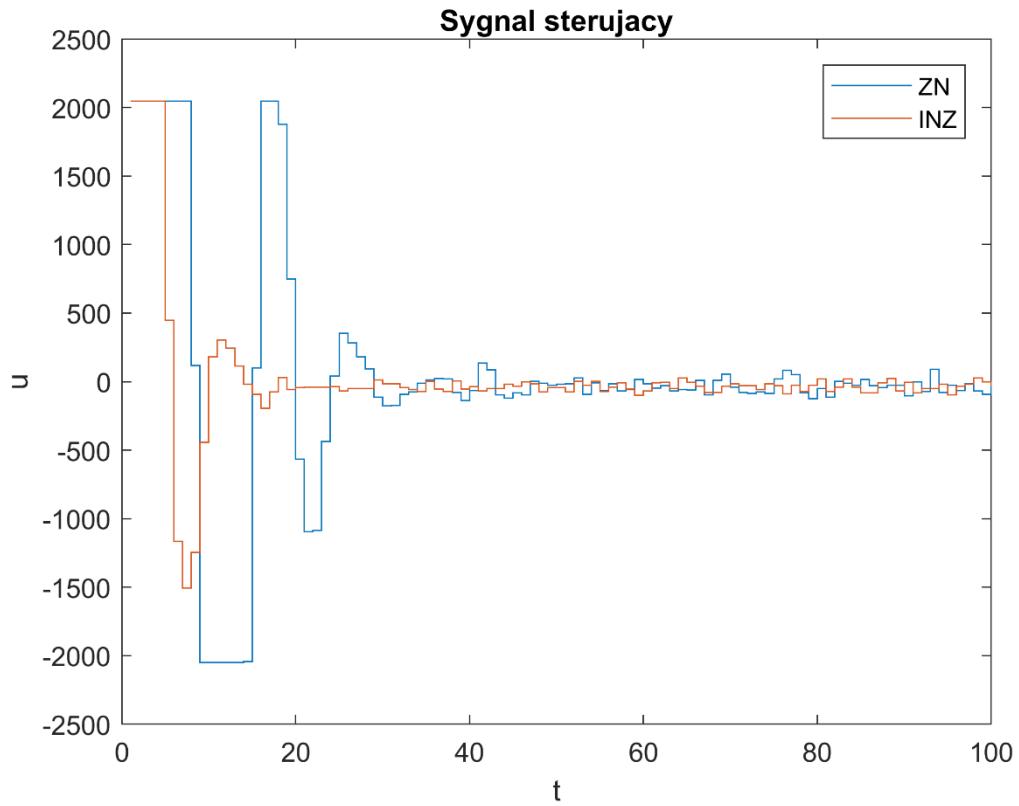
Następnie rozpoczęliśmy badanie wpływu parametru T_d na jakość regulacji, zaczynając od wartości zapewniającej mały udział członu D oraz stopniowo ten parametr zwiększając. Widać, że nawet niewielkie zmiany stałej różniczkowania dają wyraźny efekt w przełożeniu na regulację. Wraz ze zwiększaniem parametru uzyskiwaliśmy mniejsze przeregulowanie oraz szybszy czas ustalenia.

Finalnie nasz regulator posiadał parametr T_d równy 0.05.

Regulator PID: $K = 15$, $T_i = 3.5$, $T_d = 0.05$

Porównanie obu metod





Można stwierdzić, że wynik porównania jest zgodny z przewidywaniami. Regulator otrzymany przy pomocy metody inżynierskiej jest wyraźnie lepszy. Potwierdza to zarówno sygnał wyjściowy cechujący się niewielkim przeregulowaniem oraz krótkim czasem ustalenia, jak i wykres sterowania który szybciej „odbija się” od ograniczeń i do końca pozostaje w dozwolonym zakresie.

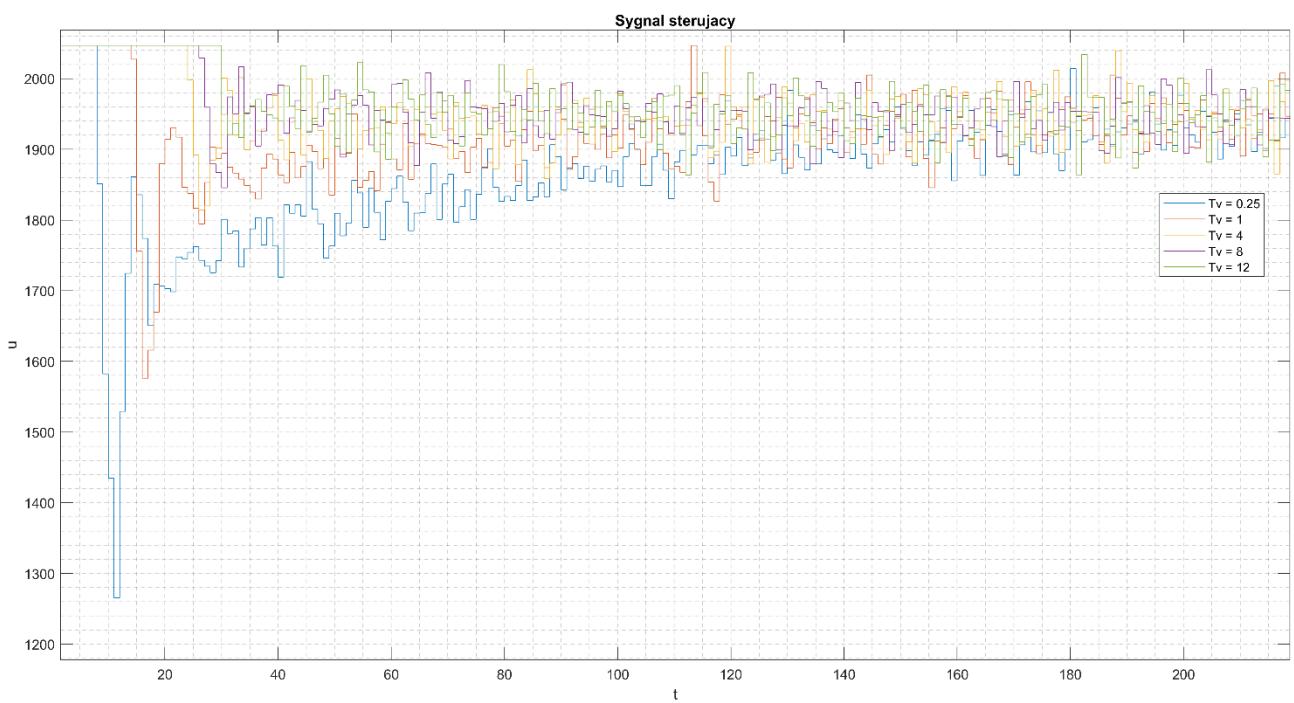
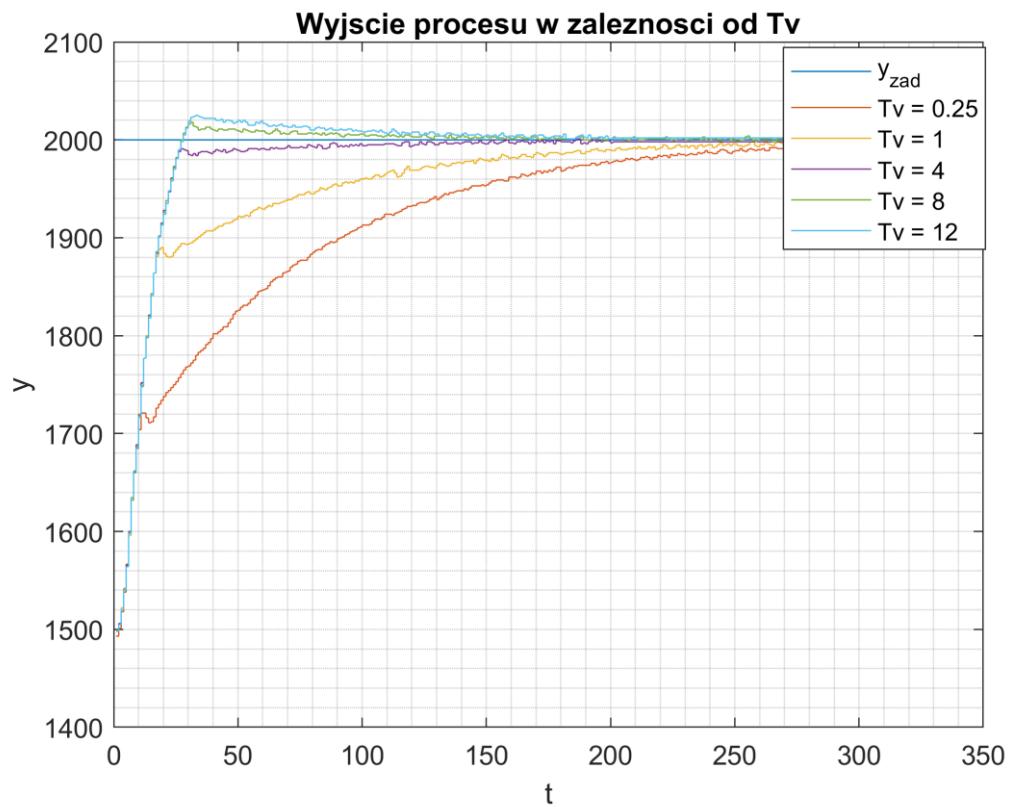
Przeprowadzone porównanie jasno pokazuje, że metoda Zieglera-Nicholsa jest metodą dobrą, ale tylko jeśli mówimy o pierwszej fazie strojenia. Pozwala ona na rozpoczęcie pracy w pewnym punkcie jednak dalsze próby manipulacji nastawami dają lepszy efekt.

Anti-windup

W prawie sterowania regulatora PID nie jest uwzględniane ograniczenie na wartości sterowania, w związku z tym należy zrealizować je osobno. Brak uwzględnienia ograniczeń powoduje, że gdy zostanie ono osiągnięte, a błąd będzie niezerowy, realizowane będzie niepotrzebne całkowanie, tj. wyznaczona wartość sygnału sterującego będzie rosła mimo, że faktyczna maksymalna wartość sygnału sterującego już została osiągnięta. Efekty tego zjawiska widoczne są szczególnie w momencie, gdy sygnał wyjściowy przekroczy wartość zadaną – wtedy przez długi czas dokonywane jest „odcałkowywanie”, tj. sygnał sterujący jest nieustannie pomniejszany (przez składową całkującą), aż jego wartości będą mniejsze niż ograniczenie i regulator będzie ponownie pracować zgodnie z oczekiwaniemi (pod warunkiem, że w tym momencie jeszcze obiekt nie wpadł w oscylacje). Zjawisko przesadnego całkowania jest nazywane nawijaniem (windup). Jednym z prostszych algorytmów przeciwdziałających nawijaniu jest pomniejszanie składowej całkującej wartości sterowania o pewną stałą przemnożoną przez różnicę między nasioną wartością sygnału sterującego, a wartością wyznaczoną przez regulator. Nosi ono nazwę anti-windup. Realizowane

jest to poprzez wprowadzenie do członu całkującego dodatkowego elementu proporcjonalnego do różnicy między faktycznym sygnałem sterującym a oczekiwany.

Zadaniem do wykonania było sprawdzenie wpływu stałej T_v , będącej parametrem wspomnianego wyżej dodatkowego elementu. Wyniki obserwowaliśmy przy skoku na tyle wysokim, żeby powodował odcięcie sygnału sterującego na ograniczeniu górnym.



Wszystkie eksperymenty dotyczące parametru T_v realizowane były dla regulatora wyznaczonego metodą inżynierską. Badanym skokiem wartości zadanej był skok od 1500 do 2000. Górnne ograniczenie sterowania wynosiło 2047. Na wykresach sterowania widoczne jest odcięcie sygnału sterującego. Duża wartość T_v odpowiada małemu wpływowi anti-windupu dlatego dla $T_v = 12$ obserwujemy długie przesterowanie wynikające z przekroczenia przez sygnał sterujący wartości ograniczenia. Ze względu na kumulację członu całkującego obserwujemy długie zbieganie do wartości zadanej. Można zauważyć, że dla małego T_v odcięcie sygnału sterującego nie jest tak wyraźne, jednak wyjście zbiega do wartości zadanej bardzo powoli, co również nie jest zadowalające.

Korzystając z mechanizmu anti-windup można skrócić oba czasy odpowiednio modyfikując parametr T_v , co oznacza jednocześnie szybszą reakcję algorytmu regulacji w przypadku osiągnięcia ograniczeń i najczęściej również ograniczenie przesterowania.

Naszym zdaniem najlepiej radził sobie regulator z T_v równym 8. Reagował zadowalająco szybko, przesterowanie było niewielkie, tak samo jak i czas ustalenia.

Regulator DMC

Zadania do wykonania

W tej części należało dokonać nastrojenia regulatora DMC oraz porównanie trajektorii sygnału wyjściowego procesu w zależności od różnych parametrów algorytmu.

//Wrzucę tutaj sobie implementację DMC skoro jest tu trochę wolnego miejsca

- Implementacja regulatora DMC:

Algorytm DMC jest jednym z najpopularniejszych algorytmów regulacji predykcyjnej. W poniższym opisie założono, że czytelnikowi znane jest zagadnienie regulacji predykcyjnej zrealizowanej na przykładzie regulatora DMC, toteż pominięto opis teoretyczny, co pozwala przejść bezpośrednio do przydatnych w implementacji równań.

Poszczególne macierze tworzone są na podstawie wektora s odpowiedzi skokowej obiektu:

- Macierz współczynników odpowiedzi skokowej M :

$$M = \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ s_2 & s_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ s_N & s_{N-1} & \cdots & s_{N-N_u+1} \end{bmatrix}_{NxN_U}$$

- Macierz służąca do wyznaczenia trajektorii swobodnej $Y_0(k)$ - M^P :

$$M^P = \begin{bmatrix} s_2 - s_1 & s_3 - s_2 & \cdots & s_D - s_{D-1} \\ s_3 - s_1 & s_4 - s_2 & \cdots & s_{D+1} - s_{D-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N+1} - s_1 & s_{N+2} - s_2 & \cdots & s_{N+D-1} - s_{D-1} \end{bmatrix}_{N \times (D-1)}$$

Pozostałe macierze:

- Wektor przeszłych zmian sterowania ΔU^P :

$$\Delta U^P(k) = \begin{bmatrix} \Delta u(k-1) \\ \vdots \\ \Delta u(k-(D-1)) \end{bmatrix}_{(D-1) \times 1}$$

- Macierz współczynników zmian sterowania K :

$$K = [M^T M + \lambda I]^{-1} M^T = \begin{bmatrix} \bar{K}_1 \\ \bar{K}_2 \\ \vdots \\ \bar{K}_{N_U} \end{bmatrix} = \begin{bmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,N} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N_U,1} & K_{N_U,2} & \cdots & K_{N_U,N} \end{bmatrix}_{N_U \times N}$$

- Aby móc wyznaczać kolejne wartości sterowania potrzebne są następujące przekształcenia:

$$\begin{aligned} u(k|k) &= u(k-1) + \Delta u(k|k) = u(k-1) + \bar{K}_1[Y^{ZAD}(k) - Y^0(k)] \\ &= u(k-1) + \bar{K}_1[Y^{ZAD}(k) - Y(k) - M^P \Delta U^P(k)] \\ &= u(k-1) + \bar{K}_1[Y^{ZAD}(k) - Y(k)] - \bar{K}_1 M^P \Delta U^P(k) \\ &= u(k-1) + \sum_{l=1}^N K_{1,l} (y^{ZAD}(k) - y(k)) - \bar{K}_1 M^P \Delta U^P(k) \end{aligned}$$

- Po podstawieniu:

$$e(k) = y^{ZAD}(k) - y(k),$$

$$K_e = \overline{\sum_{l=1}^N K_{1,l}},$$

$$K_U = \bar{K}_1 M^P$$

- otrzymano równanie:

$$u(k|k) = u(k-1) + K_e e(k) - K_u \Delta U^P(k)$$

- Po uwzględnieniu ograniczeń wartości sterowania otrzymano finalnie:

$$u(k) = u(k|k) = u(k-1) + K_e e(k) - K_u \Delta U^P(k)$$

Gdzie:

K_e to suma pierwszego wiersza macierzy K ,

K_u to iloczyn pierwszego wiersza macierzy K i M^P

Taka postać równania przy wykorzystaniu biblioteki obsługującej operacje na wektorach pozwala na implementację regulatora DMC w języku C.

Plik nagłówkowy DMC.h deklaruje bibliotekę zawierającą:

- Strukturę parametrów regulatora *DMC_type*
- Funkcję *DMC_init()* inicjującą strukturę
- Funkcję *DMC_get_control()* wyznaczającą kolejne wartości sterowania

Więcej informacji zawarto w komentarzach w listingu kodu na limonkowym tle poniżej.

```
/*
* Plik:          DMC.h
* Opis:          Biblioteka implementująca regulator DMC.
*
* Parametry regulatora wyznaczane przy pomocy dedykowanych
* skryptów Matlaba.
*
* Do wyznaczenia parametrów wymagana jest znana odpowiedź
* skokowa.
*
* Skrypty: "DMC_init.m", "DMC_script.m", "exporter.m"
*/
#include <inttypes.h>

typedef struct
{
    uint8_t D;                      // długość horyzontu dynamiki
    float Ke;                       // suma pierwszego wiersza macierzy K
    float * Ku;                     // iloczyn macierzowy pierwszego wiersza K i Mp
    float * delta_u_past;           // wektor przeszłych zmian sterowania
    float u;                         // wartość sterowania
} DMC_type;

/*
* Inicjacja struktury regulatora DMC
* dmc      - wskaźnik na strukturę regulatora
* _D       - długość horyzontu dynamiki
* _Ke     - suma pierwszego wiersza macierzy K
* _Ku     - iloczyn macierzowy pierwszego wiersza K i Mp
* u_initial - początkowa wartość wyjścia regulatora DMC
*/
void DMC_init(DMC_type*, uint8_t, float, float*, float);

/*
* Wyznaczenie nowej wartości sterowania
* dmc      - wskaźnik na strukturę regulatora
* e        - bieżący uchyb sterowania
* u_max   - maksymalna wartość sterowania
* u_min   - minimalna wartość sterowania
*/
float DMC_get_control(DMC_type*, float, float, float);
```

Plik źródłowy DMC.c definiuje funkcje biblioteczne regulatora DMC:

- *DMC_init()*
- *DMC_get_control()*

Dokładne opisy funkcji zawarte w listingu kodu na limonkowym tle.

```
/*
 * Plik:          DMC.c
 * Opis:          Biblioteka implementujaca regulator DMC.
 *                 Parametry regulatora wyznaczane przy
 *                 pomocy dedykowanych skryptow Matlaba.
 *                 Do wyznaczenia parametrów wymagana jest
 *                 znana odpowiedź skokowa.
 *                 Skrypty: "DMC_init.m", "DMC_script.m", "exporter.m"
 */

/*
 *   Inicjacja struktury regulatora DMC
 *   dmc      - wskaźnik na strukturę regulatora
 *   _D       - długość horyzontu dynamiki
 *   _Ke     - suma pierwszego wiersza macierzy K
 *   _Ku     - iloczyn macierzowy pierwszego wiersza K i Mp
 *   u_initial - początkowa wartość wyjścia regulatora DMC
 */
void DMC_init(DMC_type* dmc, uint8_t _D, float _Ke, float* _Ku, float
u_initial)
{
    uint8_t n=0;

    dmc->D = _D;
    dmc->Ke = _Ke;

    // alokacja pamięci dla wektora współczynników regulatora DMC
    dmc->Ku = malloc(sizeof(float)*(dmc->D-1));

    // alokacja pamięci dla wektora przeszłych zmian sterowania
    dmc->delta_u_past = malloc(sizeof(float)*(dmc->D-1));

    dmc->u = u_initial;

    // przekopiowanie wektora do zaalokowanego obszaru pamięci
    memcpy(dmc->Ku, _Ku, sizeof(float)*(dmc->D-1) );

    // wyzerowanie poprzednich wartości zmian sterowania
    for(n=0;n<(dmc->D-1);n++)
        dmc->delta_u_past[n] = 0.0;
}
```

Funkcja DMC_get_control() implementuje regulator DMC oparty na wzorach podanych na początku tej sekcji.

```
/*
 *   Wyznaczenie nowej wartości sterowania
 *   dmc          - wskaźnik na strukturę regulatora
 *   e            - bieżący uchyb sterowania
 *   u_max        - maksymalna wartość sterowania
 *   u_min        - minimalna wartość sterowania
 */
float DMC_get_control(DMC_type* dmc, float e, float u_max, float u_min)
{
    float delta_u;
    float tmp;
    float * new_delta_u_past;
    new_delta_u_past = malloc(sizeof(float)*(dmc->D-1));

    // Równanie regulatora DMC
    // u(k) = u(k-1) + Ke*e(k) - Ku*deltaUp(k)
    // Źródło: wzór u(k|k), str. 90 skryptu do projektu 1 SMS

    // iloczyn wektorów współczynników Ku i
    // przeszłych zmian sterowania delta_u_past
    mat_mul(dmc->Ku, 1, dmc->D-1, dmc->delta_u_past, dmc->D-1, 1, &tmp);

    // wyznaczenie nowej zmiany sterowania
    delta_u = dmc->Ke*e - tmp;
    // wyznaczenie nowej wartości sterowania
    tmp = dmc->u + delta_u;

    // nałożenie ograniczeń na sterowanie
    if(tmp > u_max)
        tmp = u_max;
    else if(tmp < u_min)
        tmp = u_min;

    // przekazanie do regulatora ograniczonej zmiany sterowania
    delta_u = tmp - dmc->u;
    dmc->u = tmp;

    // przesunięcie wektora przeszłych zmian sterowania o
    // jeden krok w tył i
    // wstawienie bieżącej zmiany sterowania na początek
    mat_move_down(dmc->delta_u_past, dmc->D-1, 1, delta_u,
new_delta_u_past);

    // zastąpienie przeszłego wektora nowym
    free(dmc->delta_u_past);
    dmc->delta_u_past = new_delta_u_past;

    return dmc->u;
}
```

Wartości wektorów wykorzystywanych w implementacji regulatora DMC wyznaczono przy pomocy poniższych skryptów Matlaba.

Skrypt DMC_script.m:

- *ładuje do środowiska wyznaczony wcześniej znormalizowany wektor odpowiedzi skokowej,*
- *definiuje zmienne określające parametry regulatora (D , N , Nu , $lambda$),*
- *wywołuje skrypt $DMC_init.m$ wyznaczający macierze regulatora DMC,*
- *wylicza wektor Ku oraz skalar Ke ,*
- *finalnie wywołuje skrypt $exporter.m$ generujący plik nagłówkowy $DMC_data.h$ załączany do projektu*

```
% Plik:          DMC_script.m
% Opis:          Skrypt wyliczający parametry regulatora DMC
%                 przeznaczonego do uruchomienia w systemie wbudowanym

% Załadowanie odpowiedzi skokowej obiektu
load('s_D44.mat')

% Założone parametry regulatora
D = length(s);           % horyzont dynamiki
N=5;                      % horyzont predykcji
Nu=1;                      % horyzont sterowania
lambda = 0.1              % kara za zmienność sterowania
run('DMC_init.m');

Ke = sum(K(1,:));
Ku = K(1,:)*Mp;

% wyeksportowanie wyznaczonych parametrów do
%pliku nagłówkowego zgodnego ze standardem języka C
run('exporter.m');
```

Skrypt DMC_init.m wyznacza macierze M i M_P regulatora zgodnie ze wzorami podanymi na początku tej sekcji.

```
% Plik:          DMC_init.m
% Opis:          Skrypt wyliczajacy parametry regulatora DMC

% Wyznaczenie macierzy M
M = zeros(D,D);
for kNu=1:Nu
    M(kNu:N,kNu) = s(1:(N+1-kNu));
end

% Wyznaczenie macierzy Mp
Mp = ones(D,D-1)*s(end);
for kD=1:D-1
    Mp(1:(N-kD),kD) = s((kD+1):(N))';
end
Mp = Mp - ones(D,1)*s(1:end-1);

fi = eye(D);
LAMBDA = lambda*eye(D);
% Wyznaczenie macierzy K
K = inv((M')*M+LAMBDA)*(M');
```

Skrypt eksporter.m generuje plik nagłówkowy DMC_data.h zgodny ze standardem języka C.

Skrypt eksporter.m umieszcza w pliku:

- *Wartości horyzontów(informacyjnie) – typ uint8_t,*
- *Parametr lambda(informacyjnie) – typ float,*
- *Skalar Ke – typ float,*
- *Wektor Ku – tablica float*

```
% Plik:           exporter.m
% Opis:          Skrypt eksportujący wyliczone parametry regulatora DMC do
%                  postaci zgodnej ze standardem języka C

% powstanie plik "DMC_data.h" w folderze Inc
fileID = fopen('../..\\Inc/DMC_data.h','w');

fprintf(fileID,'#ifndef DMC_DATA_H\n#define DMC_DATA_H\n\n');
fprintf(fileID,'#include <inttypes.h>\n\n', D);

fprintf(fileID,'//Parametry regulatora DMC\n', D);
fprintf(fileID,'uint8_t DMC_D = %d;\n', D);
fprintf(fileID,'uint8_t DMC_N = %d;\n', N);
fprintf(fileID,'uint8_t DMC_Nu = %d;\n', Nu);
fprintf(fileID,'float DMC_lambda = %f;\n\n', lambda);

fprintf(fileID,'// Przeliczone wartosci do sterowania DMC\n', D);
fprintf(fileID,'float DMC_Ke = %f;\n\n', Ke);

fprintf(fileID,'float DMC_Ku[] =\n{\n');
fprintf(fileID,'    %f,\n', Ku)
fprintf(fileID,'};\n\n');

fprintf(fileID,'#endif\n');
fclose(fileID);
```

Przykładowy plik nagłówkowy DMC_data.h wygenerowany skryptem exporter.m dla horyzontów dynamiki równym 44, predykci 5, a sterowania 1 oraz lambda równym 0.1 .

```
#ifndef DMC_DATA_H
#define DMC_DATA_H

#include <inttypes.h>

//Parametry regulatora DMC
uint8_t DMC_D = 44;
uint8_t DMC_N = 5;
uint8_t DMC_Nu = 1;
float DMC_lambda = 0.100000;

// Przeliczone wartości do sterowania DMC
float DMC_Ke = 2.085205;

float DMC_Ku[] =
{
    1.135635,      1.675895,      1.893366,      1.883774,
    1.789106,      1.672334,      1.559733,      1.459643,
    1.347042,      1.238612,      1.126011,      1.025921,
    0.934172,      0.850764,      0.763185,      0.696458,
    0.633902,      0.571346,      0.517131,      0.458745,
    0.417041,      0.379507,      0.337803,      0.296099,
    0.262736,      0.237713,      0.212691,      0.187668,
    0.154305,      0.145964,      0.129283,      0.112601,
    0.100090,      0.083408,      0.079238,      0.062556,
    0.050045,      0.041704,      0.033363,      0.020852,
    0.020852,      0.016682,      0.016682,
};

#endif
```

Zastosowana implementacja regulatora DMC z wykorzystaniem skryptów Matlaba automatycznie generującymi plik z konfiguracją regulatora pozwala na sprawne wyznaczenie nastaw regulatora DMC spełniających założenia jakości regulacji.

Wykorzystanie bibliotek regulatorów PID i DMC sprowadza się do:

- zdefiniowania struktury *DMC_type* lub *PID_type* jako globalne,
- wywołania funkcji *DMC_init()* lub *PID_init()* ze stosownymi parametrami,
- cyklicznego wywoływanego funkcji *DMC_get_control()* lub *PID_get_control()* ze stosownymi parametrami w celu sterowania obiektu

Poniżej znajduje się przykładowa skrócona implementacja bibliotek regulatorów.

```
* File Name          : main.c
* Description       : Main program body

. . .

#include "DMC_data.h"    /* parametry regulatora PID */
#include "PID_data.h"    /* parametry regulatora DMC */
#include "DMC.h"          /* biblioteka DMC */
#include "PID.h"          /* biblioteka PID */

. . .

// DMC structure
DMC_type dmc;
// PID structure
PID_type pid;

. . .

int main(void)
{
    . . .

    // inicjacja struktury regulatora DMC
    DMC_init(&dmc, DMC_D, DMC_Ke, DMC_Ku, y_zad);
    // inicjacja struktury regulatora PID
    PID_init(&pid, PID_Tp, PID_K, PID_Ti, PID_Td, PID_Tv);

    . . .

    while(1)
    {
        . . .
    }
}

. . .
```

Cyklicznie wywoływana procedura przerwania:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    . . .

    if(htim->Instance == TIM2) // TIM2 set to 20Hz
    {
        static float y = 0.0f;
        static float u = 0.0f;
        float e = 0;

        y = (input-2048.0f); // przejście z 0 - 4095 do -2048 - 2047
        e = y_zad-y;           // uchyb sterowania

        u = PID_get_control(&pid, e, 2047, -2048); // nowe sterowanie PID
        //u = DMC_get_control(&dmc, e, 2047, -2048); // nowe sterowanie DMC

        // ograniczenia sterowania
        if(u < -2048.0f) u = -2048.0f;
        if(u > 2047.0f) u = 2047.0f;

        output = u+2048.0f; // przejście z -2048 - 2047 do 0 - 4095

        updateControlSignalValue(output);

        // synteza danych przesyłanych do komputera
        sprintf(text, "U=%+8.2f;Y=%+8.2f;Yzad=%+8.2f;\n\r", u, y, y_zad);

        //BSP_LCD_DisplayStringAtLine(4, (uint8_t*)text);

        while(HAL_UART_GetState(&huart) == HAL_UART_STATE_BUSY_TX);
        if(HAL_UART_Transmit_IT(&huart, (uint8_t*)text, 40) != HAL_OK)
        {
            Error_Handler();
        }
    }
    . . .
}
```