Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Systemy mikroprocesorowe w sterowaniu projekt 1

Sprawozdanie z projektu 1

Mikołaj Wewiór, Filip Szczygielski

Warszawa, 2023

Spis treści

1. Algorytm PID

1.1. Implementacja

Algorytm PID został zaimplementowany bezpośrednio w kodzie mikrokontrolera w pliku `main.c` w funkcji przerwania zegara `TIM2`. Do utworzenia kodu skorzystaliśmy z zmiennych globalnych. W funkcji main zmieniamy jedynie trajektorię zadanej wartości wyjścia.

```
//zmienne globalne:
__IO uint32_t input = 0;
__IO uint32_t output = 0;
float T          = 0.1f;
float yzad       = 0.0f;
float prev_e     = 0.0f;
float prev_ui    = 0.0f;
float prev_uw    = 0.0f;
float prev_u     = 0.0f;

double sterowanie[4000] = { 0 };
int iter = 0;

...


void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
{
    static int i=0;
    static uint32_t tmpval= 0;
    for(i=0,tmpval=0;i<ADC_BUFFER_LENGTH; ++i){
        tmpval += uhADCxConvertedValue[i];
    }
    input = tmpval / ADC_BUFFER_LENGTH;
}


void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance == TIM2){

        static float y = 0.0f;
        static float u = 0.0f;

        y = (input-2048.0f);    // przejscie z 0 - 4095 do -2048 - 2047

        float e = yzad - y;

        static float up, ui, ud;
        static float K  = 2.8f;      // 10.82f
        static float Ti = 1.5f;      // 1.5f;
        static float Td = 0.2f;      // 1.25f;
        static float Tv = 10.0f;
```

```
        // PID:
        up = K * e;
        ui = prev_ui + (K / Ti) * T * (prev_e + e)/2 +
            (T/Tv)*(prev_uw - prev_u);
        ud = K * Td * (e - prev_e)/T;
        u = up + ui + ud;


        prev_e = e;
        prev_ui = ui;

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

        prev_u = u;
        prev_uw = u;
    }
}

...


int main(void)
{
    int i = 0;

    while (1)
    {
        if (i < 150)         yzad = 0;
        if (i == 150)        yzad = 500;
        else if(i == 300)    yzad = 1900;
        else if(i == 450)    yzad = -300;
        else if(i == 600)    yzad = -1500;
        else if(i == 800)    yzad = 100;

        if(TS_State.touchDetected > 0){
            sprintf(text2,"X=%d;Y=%d;\n",
                TS_State.touchX[0],
                TS_State.touchY[0]); // 22 znaki
        } else {
            sprintf(text2,"NO TOUCH\n"); // 22 znaki
        }
        HAL_Delay(100);
        i++;
    }
}
```

1.2. Metoda Zieglera-Nicholsa

W pierwszej kolejności wyznaczyliśmy nastawy regułami Ziglera-Nicholsa wykorzystując metodę przekaźnikową. Polegała ona na zastosowaniu regulatora z histerezą gdzie wartości sterowania zmieniały się między skrajnymi wartościami u_{max} i u_{min}. Otrzymaliśmy w ten sposób oscylacje na wyjściu i zmierzyliśmy ich amplitudę oraz okres. Następnie wyznaczyliśmy wzmocnienie krytyczne ze wzoru:

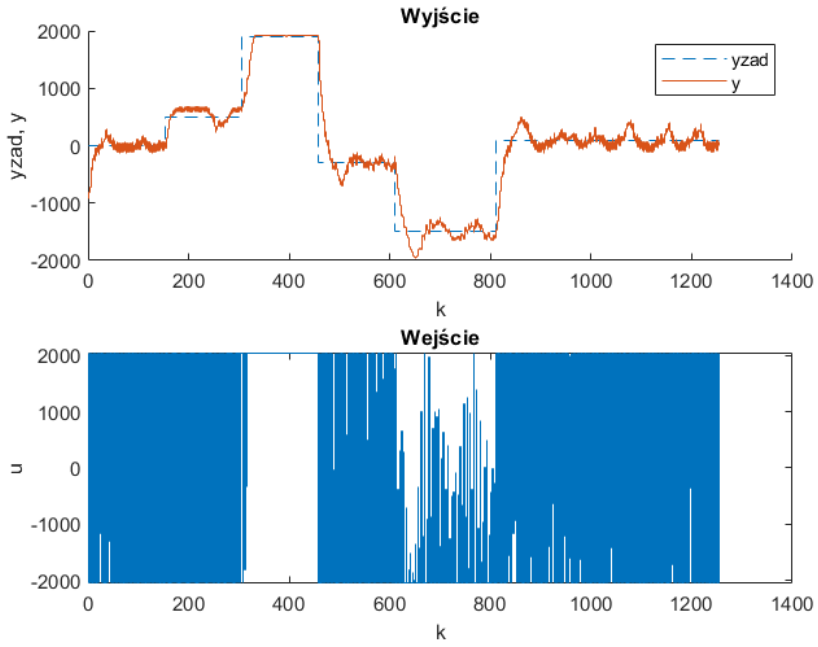
$$K_{kryt} = \frac{4b}{\pi\sqrt{a^2 - \epsilon^2}}$$

Gdzie $b = \frac{u_{max}-u_{min}}{2}$, a to amplituda sygnału wyjściowego, a ϵ to szerokość obszaru histerezy. W naszym przypadku $b = 2047$, $a = 333$, $\epsilon = 300$ oraz okres oscylacji $T_k = 10$. Po wyznaczeniu wzmocnienia krytycznego resztę nastaw wyznaczyliśmy z tabeli:

Regulator	K	T_I	T_D
P	$0{,}5K_{kryt}$	-	-
PI	$0{,}45K_{kryt}$	$T_k/1{,}2$	-
PID	$0{,}6K_{kryt}$	$T_k/2$	$T_k/8$

Tab. 1.1. Tabela Ziglera-Nicholsa

Uzyskane wartości nastaw to: $K = 10{,}8$, $T_I = 5$, $T_D = 1{,}25$ Przestestowaliśmy wyznaczone nastawy dla kolejnych skoków wartości zadanej: 500, 1900, -300, -1500, 100.

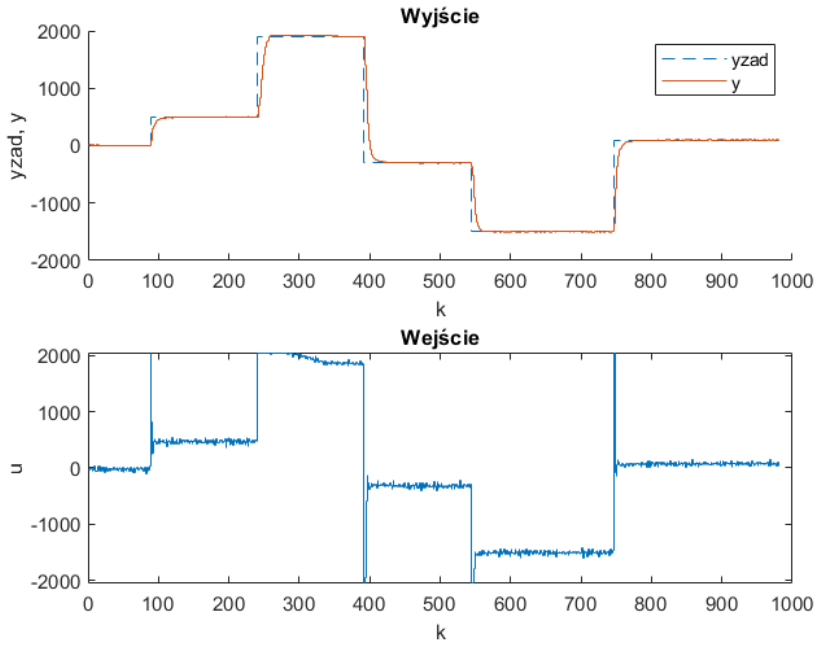


Rys. 1.1. Reakcja obiektu na zmiany wartości zadanej

Jak widać po przebiegu sygnałów regulator nadal wymaga wiele pracy, ponieważ jego działanie nie jest satysfakcjonujące a wartości sterowania w dużej mierze skaczą między u_{min} i u_{max}.

1.3. Metoda inżynierska

Kolejnym krokiem było poprawienie nastaw korzystając z metody inżynierskiej, czyli ustawienie wzmocnienia na połowę wartości wzmocnienia krytycznego, a następnie wyznaczenie T_I i T_D metodą prób i błędów. W trakcie eksperymentów jednak doszliśmy do wniosków, że wartość wzmocnienia także należy zmniejszyć i w taki sposób uzyskaliśmy parametry równe: $K = 2{,}8$, $T_I = 1{,}5$, $T_D = 0{,}2$. Ponownie przestestowaliśmy regulator dla takich samych wrtości zadanych.



Rys. 1.2. Reakcja obiektu na zmiany wartości zadanej

Na pierwszy rzut oka od razu widać, że wyznaczenie nastaw metodą prób i błędów dało znacznie lepsze wyniki. Metoda Zieglera-Nicholsa, cytując Doktora Patryka Chabera, nie gwarantuje optymalności rozwiązania, lecz dla klasycznych obiektów regulacji przemysłowej sprawdza się bardzo dobrze jako punkt startowy. Po dopracowaniu nastaw regulator zaczął działać znacznie lepiej, przestały występować oscylacje, szybciej dochodził do wartości zadanej a skoki sterowania stały się mniejsze.

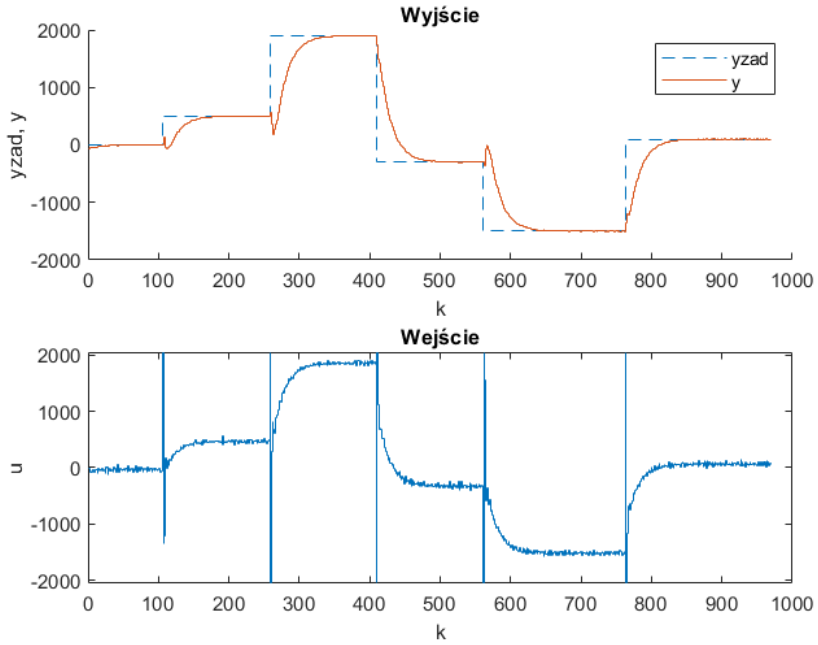
1.4. Anti-Windup

Anti-windup to technika stosowana w systemach regulacji, mająca na celu zapobieganie nadmiernemu całkowaniu. Gdy regulator osiąga swoje ograniczenia, może dojść do sytuacji, w której wyjście regulatora jest blokowane na skutek ograniczeń fizycznych (np. ograniczenia zakresu sterowania). W rezultacie regulator nie może wykonać zamierzonej korekcji, co prowadzi do niestabilności systemu lub pogorszenia jakości regulacji. Implementacja anti-windup'u polega na rozbudowaniu członu całkującego o dodatkowy element.

$$u_I(k) = ... + \frac{T}{T_v}(u_w(k-1) - u(k-1))$$

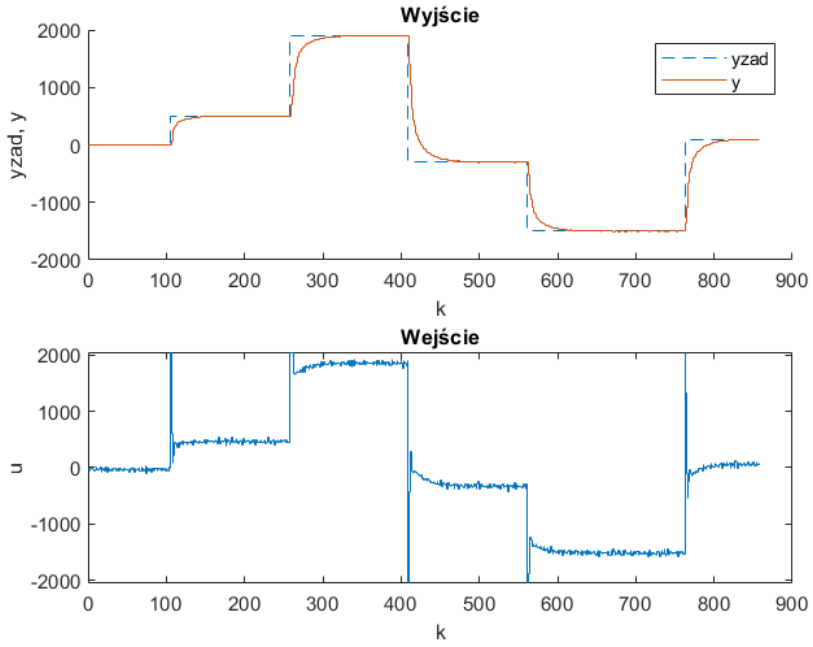
Gdzie u_w to sygnał który faktycznie został zaaplikowany do procesu. Porównanie działania regulatora dla różnych wartości Tv.

$Tv = 0{,}1$



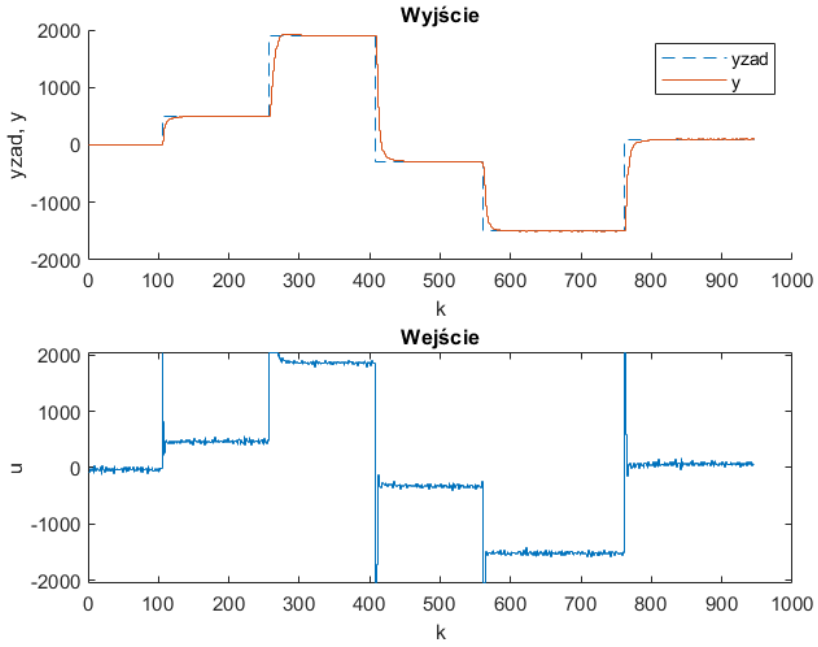
Rys. 1.3. Obiekt z regulatorem z $Tv = 0{,}1$

$Tv = 1$



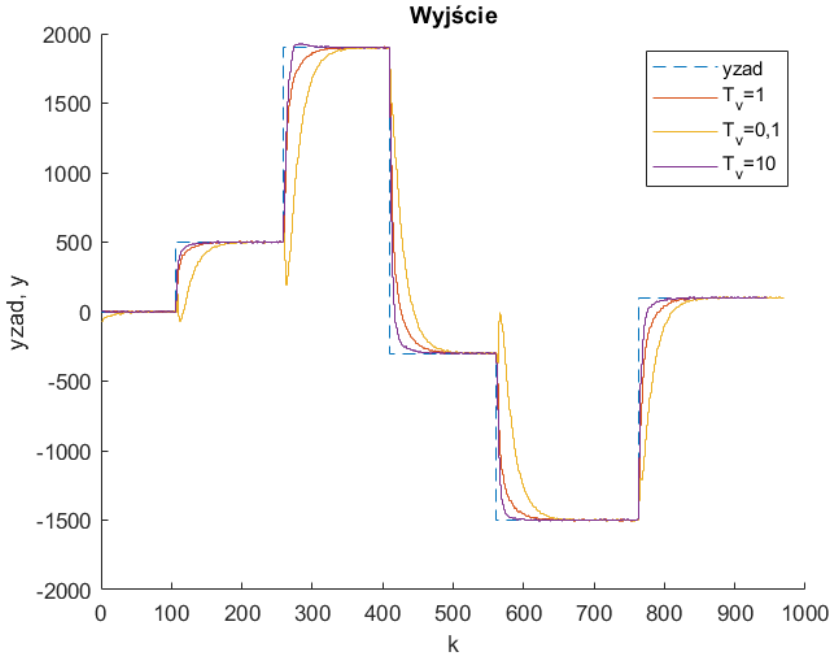
Rys. 1.4. Obiekt z regulatorem z $Tv = 1$

$Tv = 10$



Rys. 1.5. Obiekt z regulatorem z $Tv = 10$

Porównanie wpływu T_v.



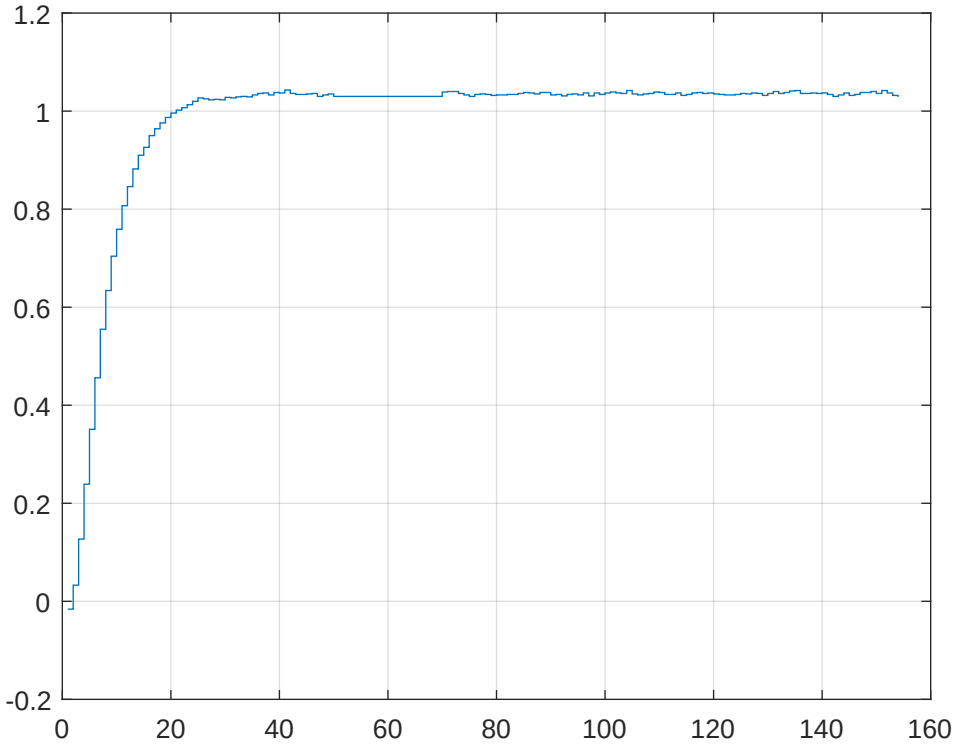
Rys. 1.6. Porównanie obiektów

Jak widać na wykresie im mniejsza wartość T_v tym mocniej regulator stara się zapobiec przesterowaniu i nawijaniu. Najlepiej to widać przy $T_v = 0{,}1$ gdzie przez chwilę wartość sterowania idzie w przeciwną stronę do zmiany wartości zadanej.

2. Algorytm DMC

2.1. Implementacja

Aby móc skorzystać z algorytmu DMC musieliśmy najpierw zebrać odpowiedź skokową obiektu. Wymusiliśmy zmianę sterowania z 0 na 1000 i sprawdziliśmy jak odpowiada obiekt. Następnie każdą z wartości s_k odpowiedzi skokowej przeskalowaliśmy o wartość zmiany sterowania żeby uzyskać skok jednostkowy. Otrzymane wartości zczytaliśmy z mikrokontrolera i zarchiwizowaliśmy w `Matlabie`. Uzyskana odpowiedź skokowa prezentuje się następujący sposób.



Rys. 2.1. odpowiedz skokowa

Korzystając z odpowiedzi skokowych, zapisanych pod zmienną `s` (wektor kolejnych wartości s_k), wykonaliśmy w środowisku `Matlab` konieczne obliczenia, czyli wyznaczenie macierzy M, M^P i wektorów K, K_j^u.

```
D = 50;
N = 20;
Nu = 10;
lambda = 1;

M = zeros(N, Nu);

for c = 1:Nu % kolumny
    for r = 1:N % wiersze
```

```
        if (r-c+1) <= 0
            M(r, c) = 0;
        elseif (r-c+1) > D
            M(r, c) = s(D);
        else
            M(r, c) = s(r-c+1);
        end

    end
end


Mp = zeros(N, D-1);

for c = 1:D-1 % kolumny
    for r = 1:N % wiersze

        if r+c > D
            s(c+r) = s(D);
        elseif c > D
            s(c) = s(D);
        end

        Mp(r, c) = s(c+r) - s(c);

    end
end


K = (M'*M + lambda*eye(Nu, Nu)) \ M';

K_1 = K(1,:);


Ke = 0;

for i = 1:size(K_1, 2)
    Ke = Ke + K_1(i);
end


Kuj = zeros(1, D-1);

for j = 1:D-1
    ku_j = K_1 * Mp(:,j);
    Kuj(j) = ku_j;
end
```

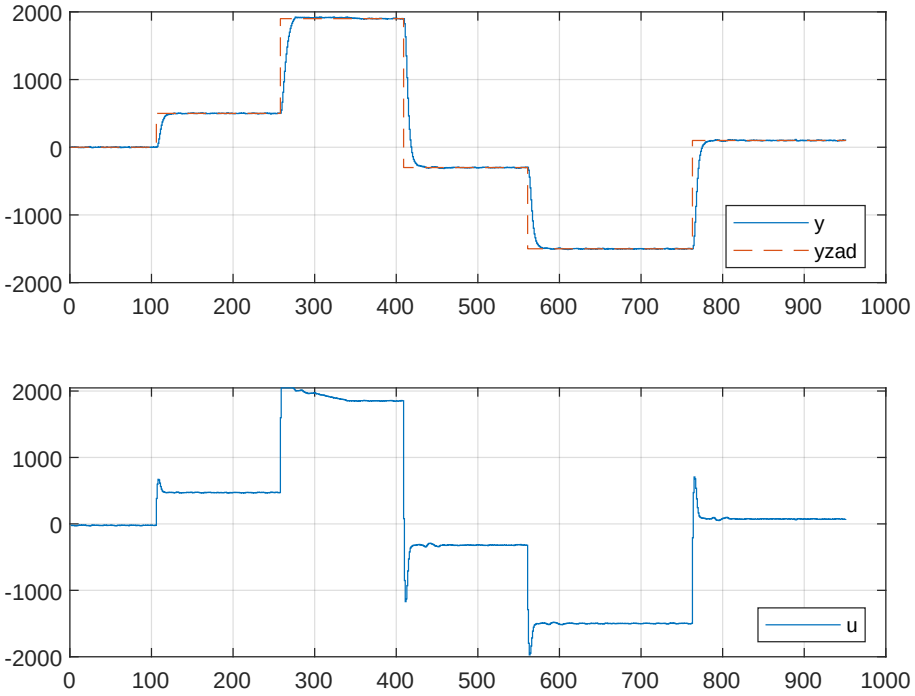
Następnie do mikrokontrolera wprowadzaliśmy wektor K_j^u. Został on zapisany jako tablica zmiennych typu `double`. Sterowanie dla bieżącej chwili dyskretnej było liczone już na mikrokontrolerze. Rola `Matlaba` sprowadziła się więc do jednorazowego wykonania koniecznych obliczeń

macierzowych offline - przed działaniem całego cyklu algorytmu, a kwestia wyliczania sterowania pozostawiona była mikrokontrolerowi. Przy każdorazowych zmianach parametrów regulatora tj. któregoś z horyzontów lub paramtetru λ, należało wykorzystać skrypt `Matlaba`. Implementacja algorytmu regulacji na mikrokontrolerze (kod odpowiadający za regulator PID został zakomentowany, a pod nim znalazła się implementacja DMC).

```
int D = 50, N = 20, Nu = 10;
float lambda = 1.0f;

// dane wprowadzane z Matlaba
static float Ke = 0.8060f;
double Kuj[49]  = { ... }

float elem = 0.0f;
float du_kmj;

for(int j = 0; j <= D-2; j++){
    du_kmj = sterowanie[max(0, iter-j)] - sterowanie[max(0, iter-j-1)];
    elem = elem + Kuj[j]*du_kmj;
}

float du = Ke * e - elem;

u = sterowanie[iter] + du;

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

sterowanie[iter+1] = u;
```

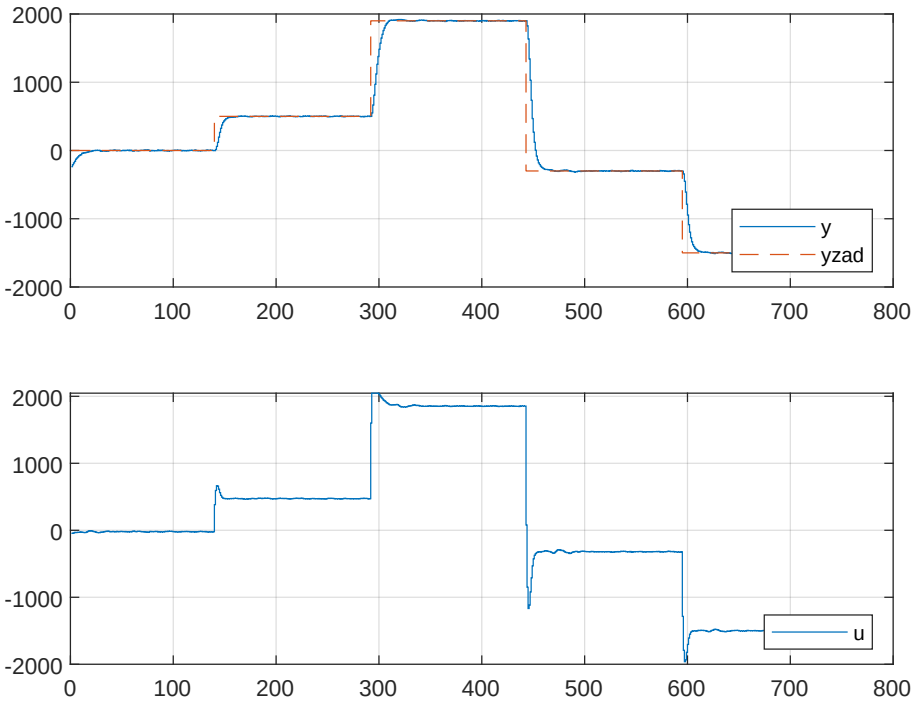
Nastawy, które wybraliśmy metodą inżynierską:

$$D = 50,\ N = 20,\ N_u = 10,\ \lambda = 1$$

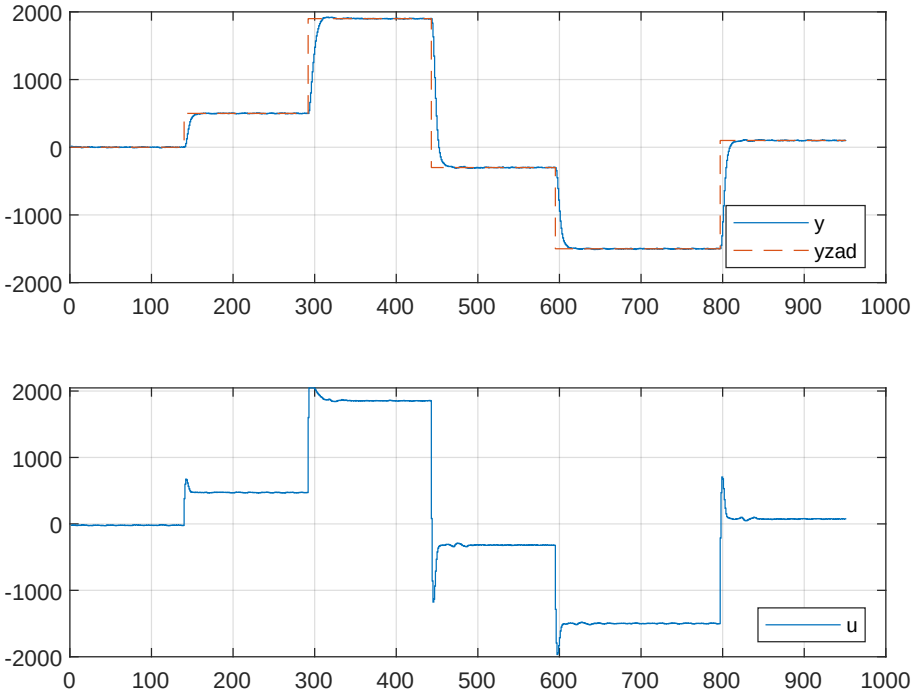


Rys. 2.2. działanie regulatora DMC dla podstawowych parametrów

2.2. Badanie działania zmian horyzontu predykcji i sterowania



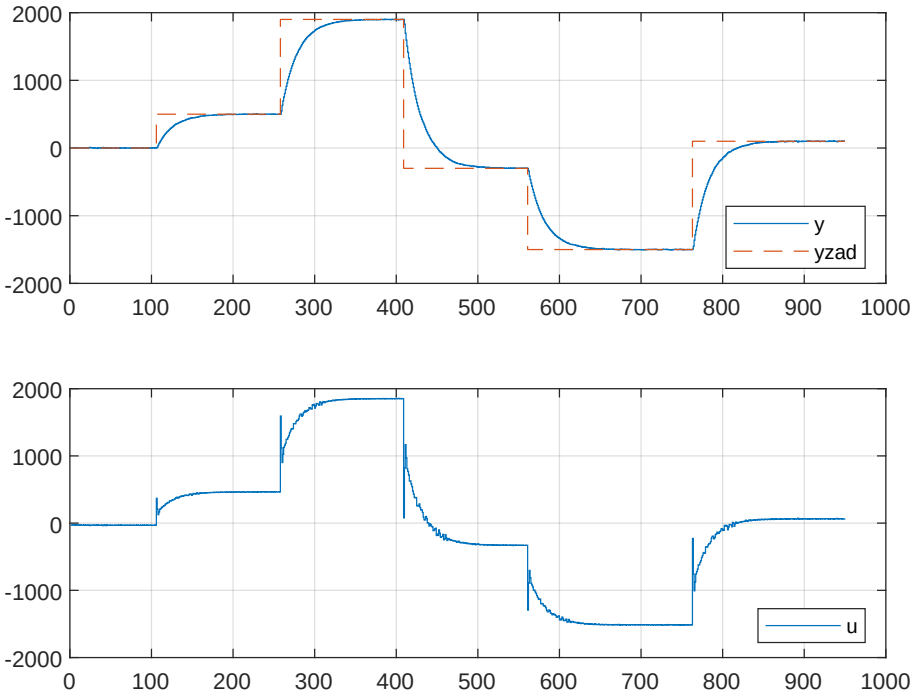
Rys. 2.3. Regulator przy N=50



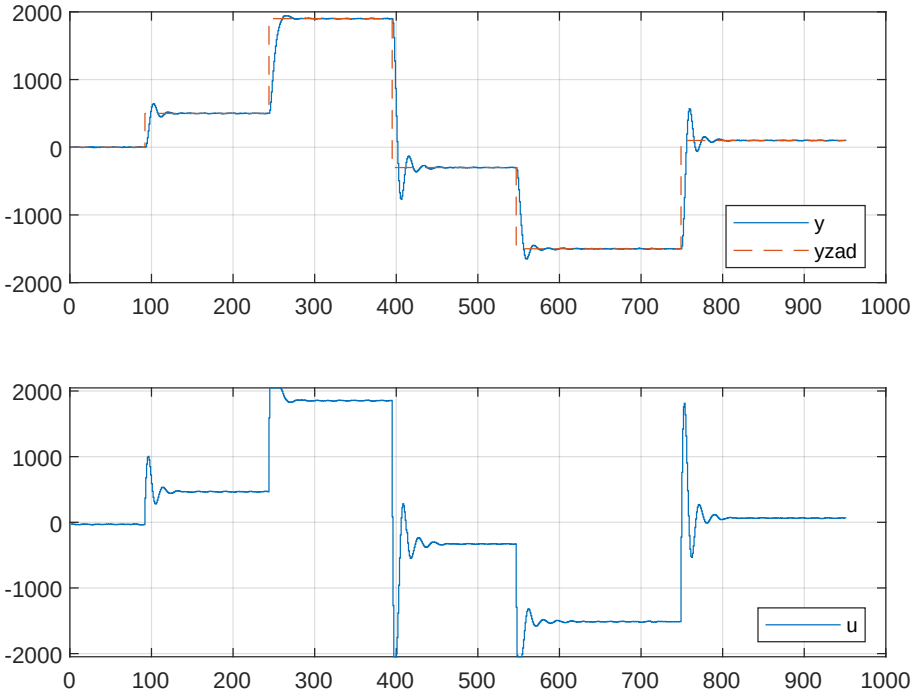
Rys. 2.4. Regulator przy N_u=50

Zarówno zwiększenie horyzontów N i N_u nie za dużo zmieniło. Mogliśmy spodziewać się nieco lepszych przebiegów, jednak nie zauważyliśmy znaczącej zmiany. Nieudało nam się jednak wykonać testów dla niższych wartości tych horyzontów, co spowodowałoby zapewne w pewnym momencie pogorszeniem jakości regulacji.

2.3. Badanie działania współczynnika λ



Rys. 2.5. Regulator przy λ=10



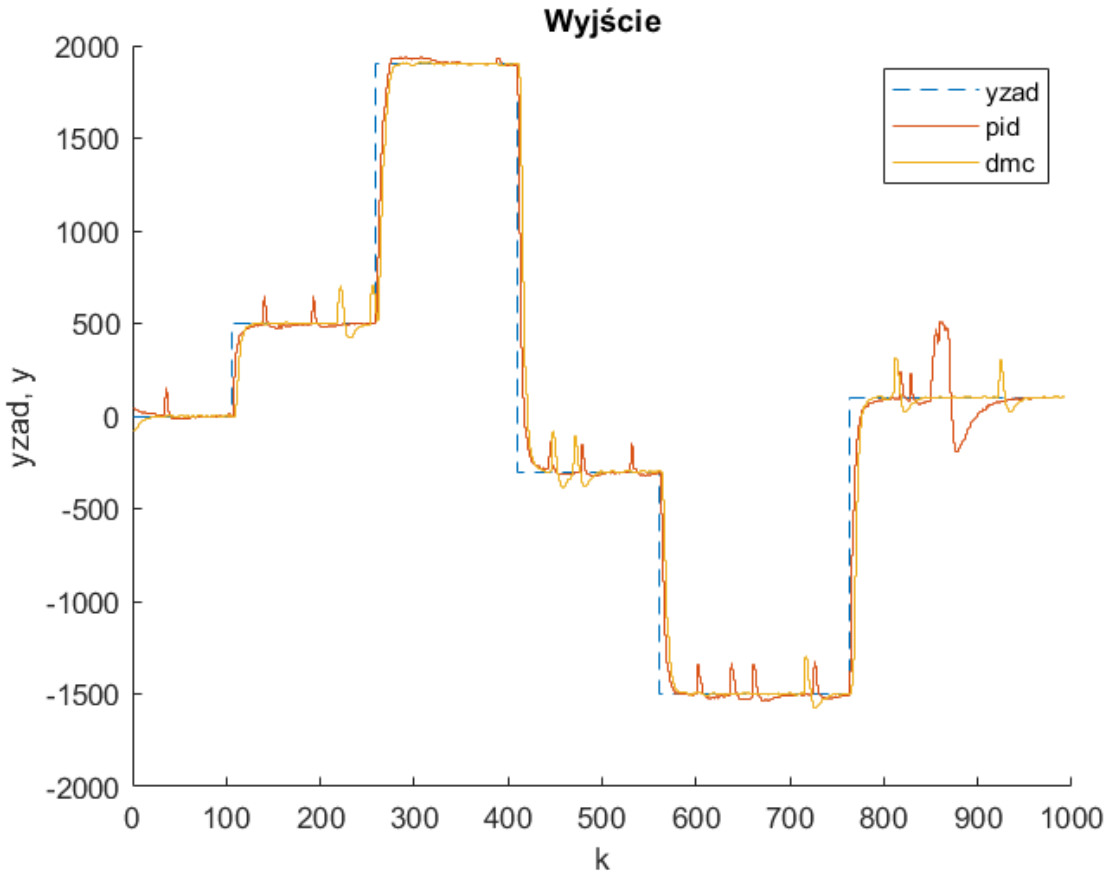
Rys. 2.6. Regulator przy λ=0.01

Wpływ lambdy na przebiegi sygnałów sterującego i wyjściowego są znaczące. Zgodnie z oczekiwaniami za mała wartość lambdy powoduje bardziej gwałtowne zmiany sterowania a przez to więszke chwilowe przeregulowania i oscylacje. Zbyt duża z drugiej strony powoduj bardzo długie ustalenie obiektu, ale bez przesterowania. Dlatego λ o wartości 1 okazała się najlepszym z badanych kandydatów - obiekt szybko dochodzi do wartości zadanej, na wyjściu praktycznie nie występuje przeregulowania, a sygnał sterujący również nie ma bardzo wysokich pików.

3. Porównanie algorytmów

3.1. Odporność na zakłócenia

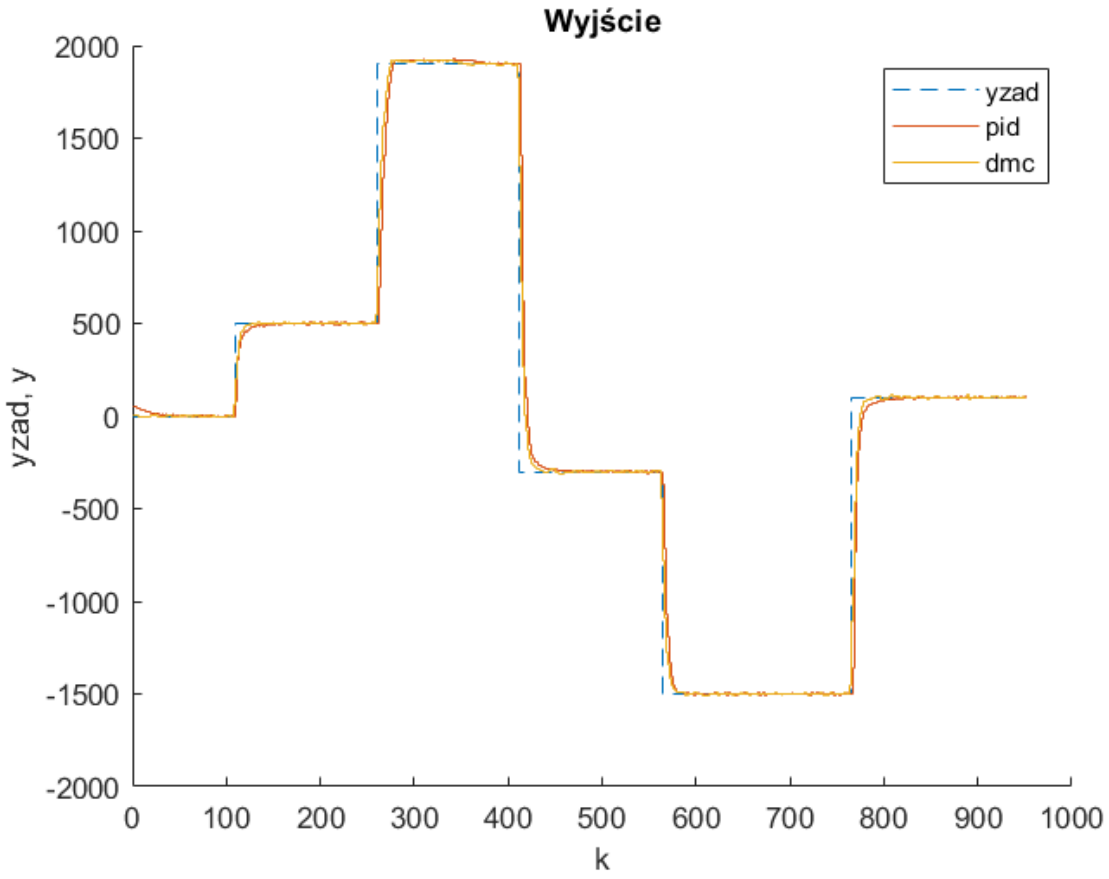
Porównanie odporności regulatorów na zakłócenia.



Rys. 3.1. Porównanie regulatorów

W naszym przypadku regulator PID lepiej radził sobie z zakłóceniami nie mierzalnymi, po zadaniu wartości zakłóceń występowało mniejsze przeregulowanie i szybciej wracał do wartości zadanej. Może to wynikać z faktu, że regulaor PID wartośc sterowania wylicza z uchybu i lepiej jest w stanie zareagować na jego chwilową zmianę. W przypadku DMC, który jest zbudowany na analitycznym modelu, jest mniej elastyczny jeśli chodzi o nagłe zmiany sterowania, których nie mierzy i nie jest w stanie przewidzieć.

3.2. Porówananie wyjść procesu



Rys. 3.2. Porównanie procesów

3.2.1. Przesterowanie

W przypadku obu regulatorów przesterowanie nie występuje - jest to zamierzony przez nas efekt.

3.2.2. Czas ustalenia

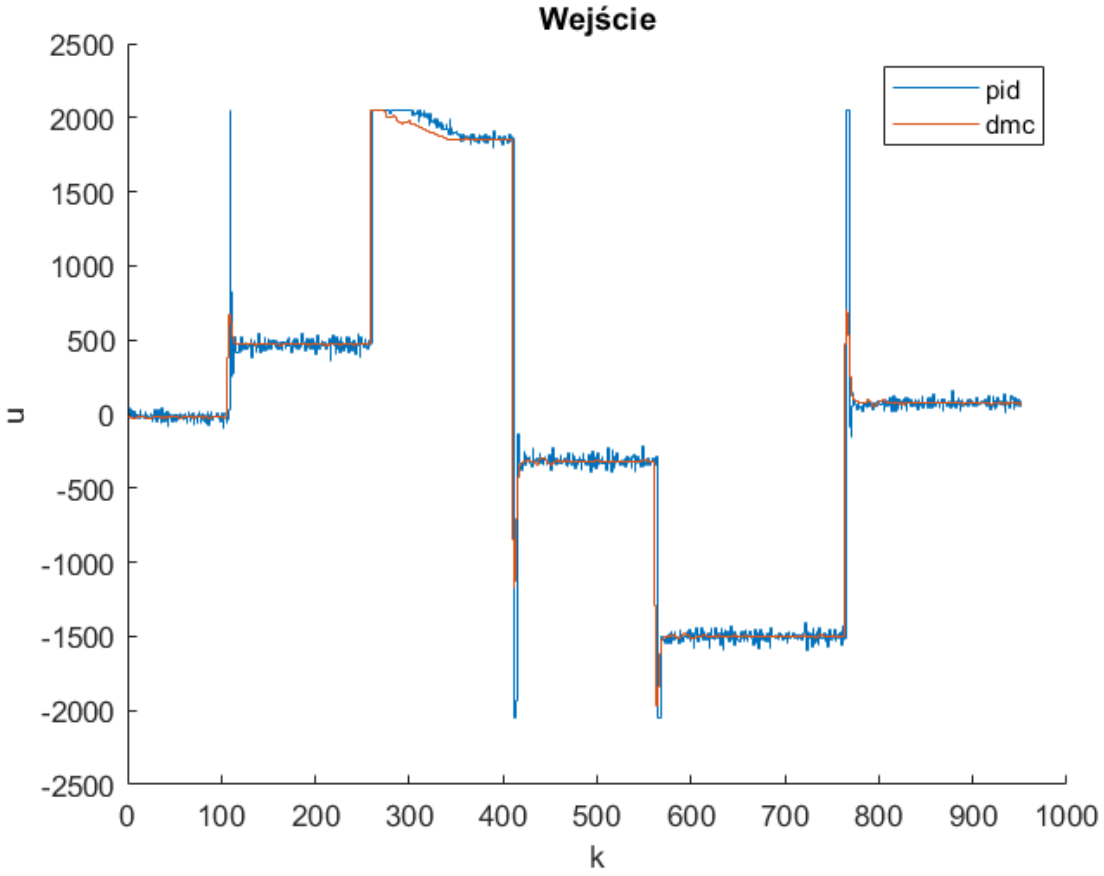
Regulator DMC jest minimalnie szybszy od regulatora PID.

3.2.3. Oscylacje

W przypadku obu regulatorów oscylacje nie występują. Oczywiście można zauważyć delikatne szumy w sygnałach ale jest to nieuniknione. Są to szumy pomiarowe związane z ADC oraz samym mikrokontrolerem.

3.3. Podsumowanie

Regulatory funkcjonują bardzo podobnie i jedyną dużą różnicą w ich działaniu są wartości sterowania.



Rys. 3.3. Porównanie obiektów

W przypadku regulatora PID wartości sterowania mają znacznie większe przyrosty i są mniej stabilne po osiągnięciu wartości zadanej. Wejście regulatora DMC jest bardziej stabilne i zmiany wartości mają mniejsze amplitudy.

Nie można określić, który z regulatorów jest lepszy. Jeżeli zależy nam na delikatniejszym sterowaniu to w tej kwestii powinniśmy zastosować DMC. Jeśli jednak bardziej zależy nam na szybszej kompensacji zakłóceń to w takim przypadku lepszy okaże się regulator PID.