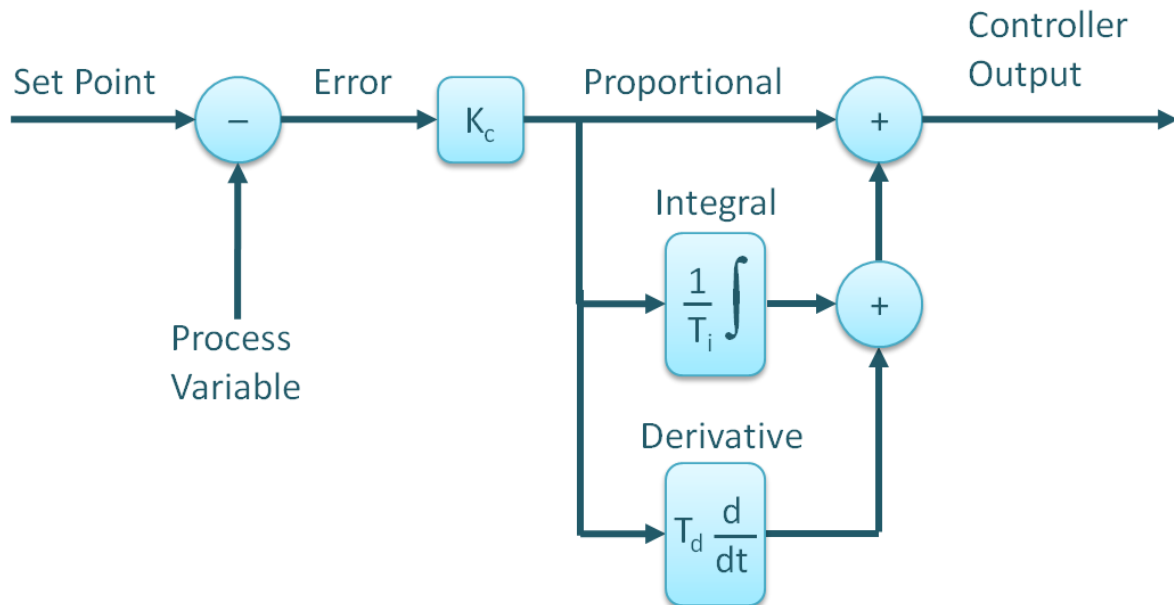


Implementacja regulatora PID:

Regulator PID składa się z trzech członów:

- Członu proporcjonalnego P o wzmacnieniu 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.



Fot.: Struktura regulatora PID

Źródło: <http://blog.opticontrols.com/archives/344>

Taką konfigurację po przekształceniu do dziedziny czasu dyskretnego zastosowano w implementacji. 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.

```
#ifndef PID_H
#define PID_H

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

#include <inttypes.h>

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, 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);

#endif
```

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  $K \cdot T_p \cdot (e_{past} + e) / 2 / T_i$ 
    u_i = pid->u_i_past + pid->K * pid->T_p * (pid->e_past + e) / 2 / pid->T_i;

    // 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->T_v > 0.0 )
        u_i += pid->T_p * (pid->u_w_past - pid->u_past) / pid->T_v;

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

    // 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

```
#ifndef PID_DATA_H
#define PID_DATA_H

#include <inttypes.h>

//Parametry regulatora PID

// okres próbkowania
#define PID_Tp (1/20.0)
// parametr anti-winding
#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 inżynierską
/*
#define PID_K 15.00f
#define PID_Ti 3.5f
#define PID_Td 0.040f
*/
////////////////////////////////////

#endif
```

Implementacja regulatora DMC:

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.

```
#ifndef DMC_H
#define DMC_H

/*
 * Plik:          DMC.h
 * Opis:          Biblioteka implementująca regulator DMC.
 *                Parametry regulatora wyznaczone 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);

#endif
```

Plik źródłowy DMC.c definiuje funkcje biblioteczne regulatora DMC:

- DMC_init()
- DMC_get_control()

Dokładne opisy funkcji zawarte w listingu kodu na limonkowym tle.

```
#include "mat_lib.h"
#include "DMC.h"
#include <stdlib.h>
#include <string.h>

/*
 * Plik:          DMC.c
 * Opis:          Biblioteka implementująca regulator DMC.
 *               Parametry regulatora wyznaczone przy
 *               pomocy dedykowanych skryptów 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
    //  $u(k) = u(k-1) + K_e \cdot e(k) - K_u \cdot \Delta u_p(k)$ 
    // Źródło: wzór  $u(k|k)$ , str. 90 skryptu do projektu 1 SMS

    // iloczyn wektorów współczynników  $K_u$  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;
}
```

Wektory wykorzystywane w implementacji regulatora DMC wyznaczone 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
%               przeznaczony 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 Mp regulatora zgodnie ze wzorami podanymi na początku tej sekcji.

```
% Plik:          DMC_init.m
% Opis:          Skrypt wyliczający 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. 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, predykcji 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 wartosci 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 wywoływania funkcji DMC_get_control() lub PID_get_control() ze stosownymi parametrami w celu sterowania obiektu

Poniżej skrócona implementacja bibliotek regulatorów.

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

. . .

#include "DMC_data.h"
#include "PID_data.h"
#include "DMC.h"
#include "PID.h"

. . .

// 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)
    {
        . . .
    }
}

. . .
```

```

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();
        }
    }
    . . .
}
. . .

```