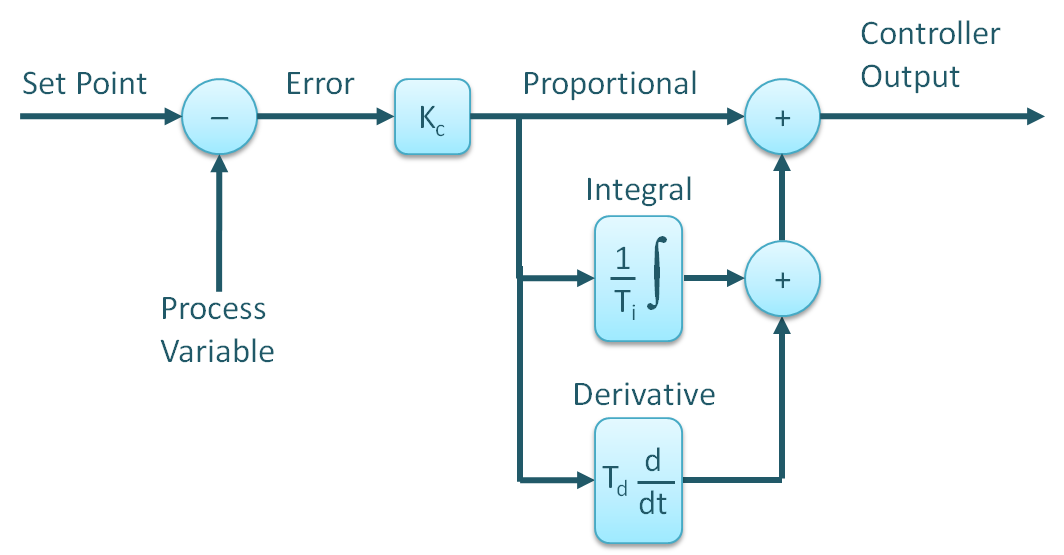
Implementacja regulatora PID:

Regulator PID składa się z trzech członów:

* Członu proporcjonalnego P o wzmocnieniu (na rysunku oznaczone ),
* Członu całkującego I o czasie zdwojenia ,
* Członu różniczkowego D o czasie wyprzedzenia

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 🡪 - wartość zadana
* Process Variable 🡪 - wyjście obiektu
* Error 🡪 - uchyb sterowania
* Proportional 🡪 - składowa proporcjonalna sterowania
* Integral 🡪 - składowa całkowa sterowania
* Derivative 🡪 - składowa różniczkowa sterowania
* Controller Output 🡪 - wyjście regulatora

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

Składowa wyznaczana ze wzoru:

Składowa regulatora bez funkcjonalności anti wind-up wyznaczana ze wzoru:

Składowa regulatora z funkcjonalnością anti wind-up wyznaczana ze wzoru:

Składowa wyznaczana ze wzoru:

Gdzie:

,

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**;** // wzmocnienie 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 - wzmocnienie członu proporcjonalnego P

\* \_PID\_Ti - parametr członu całkującego I

\* \_PID\_Td - parametr członu róźniczkowego 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 implementuąca 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 - wzmocnienie członu proporcjonalnego P

\* \_PID\_Ti - czas zdwojenia członu całkującego I

\* \_PID\_Td - czas wyprzedzenia członu róźniczkowego 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**;** // skladowa sterowania od P

float u\_i **=** 0**;** // skladowa sterowania od I

float u\_d **=** 0**;** // skladowa 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\*Tp\*(e\_past+e)/2/Ti

u\_i **=** pid**->**u\_i\_past **+** pid**->**K**\***pid**->**Tp**\*(**pid**->**e\_past **+** e**)/**2**/**pid**->**Ti**;**

// anti wind-up, aktywny jeśli Tv>0;

// składowa I powiększana dodatkowo co krok o Tp\*(u\_w\_past-u\_past)/Tv

// 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\*Td\*(e-e\_past)/Tp

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

#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 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**);**

#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 wyznaczane 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 wartosci zmian sterowawnia

**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**;**

**}**

Wektory wykorzystywane 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 Mp regulatora zgodnie ze wzorami podanymi na początku tej sekcji.

% Plik: DMC\_init.m

% Opis: Skrypt wylicząjacy 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**();**

**}**

**}**

. . .

**}**

. . .