UNIVERSITAS INDONESIA

MATLAB / SIMULINK BASED MODELING AND SIMULATION OF FUZZY PI CONTROL FOR PMSM

UJIAN AKHIR SEMESTER

SISTEM KENDALI MOTOR LISTRIK

Raihan Muhammad Syahran

1906379301

FAKULTAS TEKNIK

PROGRAM STUDI TEKNIK ELEKTRO

BOGOR

JUNI 2022

DAFTAR ISI

BAB I
PENDAHULUAN

I.1. Latar Belakang

Sistem control servo dengan presisi tinggi tidak hanya membutuhkan respon yang cepat, akan tetapi harus kokoh dari beban eksternal. Ketika PMSM(Permanent Magnet Synchronous Motor) diberikan perubahan atau beban eksternal, Kontroler PI tidak dapat memeberikan solusi atas permasalahn kontradiksi antara performa dinamik dan static sehingga kurang memenuhi persyaratan dalam sistem servo dengan track yang cepat.

Untuk menanggulangi kekurangan kontroler PI, dalam makalah ini akan dibahas mengenai kontrol fuzzy(fuzzy controller) yang diaplikasikan dalam PMSM untuk mengatur parameter dari kontroler PI. Simulasi ini akan dilakukan dalam MATLAB dengan memanfaatkan SIMULINK. Setiap blok dalam SIMULINK akan menggunakan C-MEX, yaitu sebuah blok yang mampu menggunakan Bahasa C untuk mensimulasikan suatu blok dalam SIMULINK. Hasil yang ingin dicapai akan menunjukkan suatu sistem PMSM yang dapat mengatasi performa dinamik dari menggunakan kontroler PI ditambah dengan Fuzzy Logic Controller dalam suatu system.

I.2. Tujuan Penulisan

Tujuan dari laporan ini adalah untuk membandingkan performa yang dihasilkan oleh Fuzzy Logic Controller yang diaplikasikan dengan PI Controller dalam PMSM dengan performa PI Controller tanpa Fuzzy Logic Controller yang diaplikasikan dalam PMSM.

I.3. Rumusan Masalah

Rumusan masalah pada laporan ini meliputi:

- Bagaimana cara mengendalikan PMSM menggunakan Fuzzy Logic Controller?
- Bagaimana cara membandingkan performa PMSM sebelum dan sesudah diberikan Fuzzy Logic Controller?

I.4. Sistematika Penulisan

Dalam makalah ini, terdapat beberapa bagian yang masing-masing bagiannya:

- Bab I : Pendahuluan

  Bab ini berisi mengenai latar belakang, tujuan penulisan, rumusan masalah, dan sistematika penulisan.
- Bab II : Landasan Teori

Bab ini berisi mengenai landasan teori yang digunakan saat melakukan simulasi dan penelitian.

- BAB III: Simulasi dan Analisis

  Bab ini berisi hasil dan analisis dari simulasi yang dihasilkan dari proses penelitian.

- BAB IV : Kesimpulan

  Bab ini berisi kesimpulan dari penelitian yang sudah dibuat.

BAB II
LANDASAN TEORI

II.1. Permanent Magnet Synchronous Motor

Permanent Magnet Synchronous Motor (PMSM) merupakan motor AC sinkron dimana medan eksitasi berasal dari magnet permanen dan memiliki gelombang back EMF sinusoidal. PMSM merupakan gabungan dari suatu motor induksi dan brushless DC motor. Hal tersebut dikarenakan PMSM memiliki magnet permanen dalam rotor dan kumparan dalam stator yang memiliki kemiripan dengan brushless DC motor. Akan tetapi, struktur dari stator dengan kumparan dibentuk untuk menghasilkan suatu sinusoidal flux density yang memiliki kemiripan dengan motor induksi. PMSM memiliki daya yang lebih tinggi dibandingkan dengan motor induksi dikarenakan memiliki ratingyang sama dikarenakan tidak terdapat daya stator yang dialihkan untuk produksi medan magnet.

PMSM mampu menghasilkan torsi walau tidak terdapat kecepatan, namu memerlukan kontroler inverter digital untuk operasinya. PMSM sering digunakan untuk servo yang membutuhkan performa tinggi dengan efisiensi yang tinggi. Performa yang tinggi dari control motor dapat terlihat dari rotasi yang halus dari kecepatan motor tersebut, dengan akselerasi dan deselerasi yang cukup cepat.

Untuk membuat suatu control tersebut, teknik vector control digunakan dalam PMSM. Teknik vector control yang biasa digunakan dalam PMSM adalah field-oriented control(FOC). Sistem penggerak PMSM dapat dimodelkan secara matematis untuk mempermudah perhitungan parameter. Persamaan arus stator PMSM pada koordinat putar d-q dapat dijabarkan dengan sebagai berikut:

$$\frac{di_{sd}}{dt} = \frac{v_{sd} - Ri_{sd} - e_d}{L}$$

$$\frac{di_{sq}}{dt} = \frac{v_{sq} - Ri_{sq} - e_q}{L}$$

Dimana v_{sd}, v_{sq}, dan i_{sd}, i_{sq} merupakan tegangan dan arus pada sumbu d dan q. R merupakan resistansi stator dan L merupakan induktansi stator. e_d dan e_q merupakan back EMF(Electromagnetive Force), dimana:

$$e_{sd} = p\varphi_{Fu} = -\omega_e L_{sq} i_{sq}$$

$$e_{sq} = p\varphi_{Fu} = \omega_e (L_{sd} i_{sd} + \Psi_F)$$

$$\omega_e = N\omega_r$$

Dengan L_{sd} dan L_{sq} merupakan induktansi pada sumbu d dan q. ω_e merupakan mechanical angular velocity, dan N merupakan jumlah kutub pada PMSM. Vektor-vektor dalam sumbu d-q lalu dikonversikan menjadi bentuk 3 fasa, dengan rumus:

$$\begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos\theta & \cos\theta & \frac{1}{\sqrt{2}} \\ \cos\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta - \frac{2\pi}{3}\right) & \frac{1}{\sqrt{2}} \\ \cos\left(\theta + \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \\ i_0 \end{bmatrix}$$

Dari persamaan diatas, akan didapatkan persamaan gerak mekanik dari PMSM:

$$J\frac{d\omega_r}{dt} = T_e - B\omega_r - T_L$$

Dengan J merupakan momen inersia rotor dan beban, T_e merupakan torsi elektromagnetik, B merupakan koefisien gesek viskositas, dan T_L merupakan torsi beban. Torsi elektromagnetik dapat didapatkan dengan persamaan:

$$T_e = \frac{3}{2}\varphi_F i_{sq}$$

II.2. PI Controller

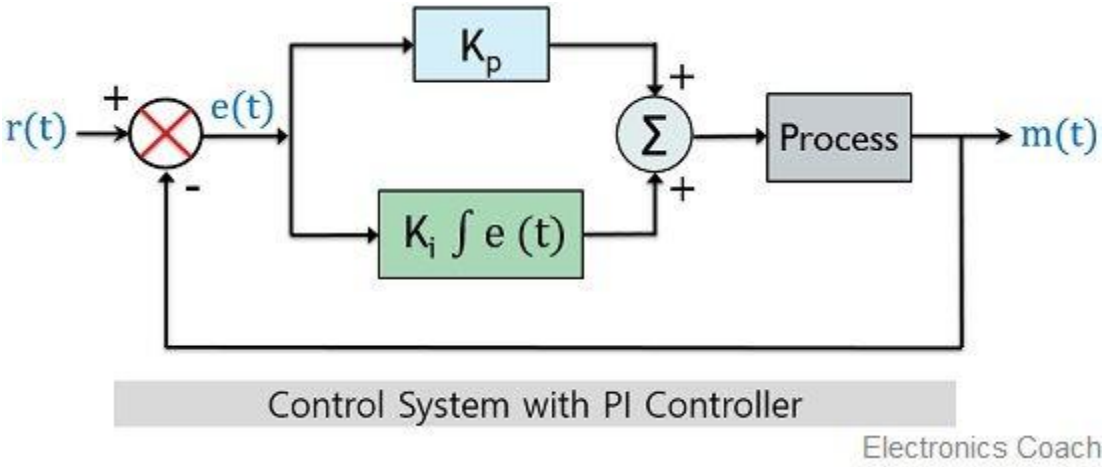
PI controller atau Proportional plus Integral(PI) Controller merupakan tipe kontroler yang dibentuk dari kombinasi kontrol proporsional dan kontrol integral. Dalam PI Controller, sistem kontrol dari proporsional dan integral diutilisasikan sehingga membentuk suatu feedback controller yang lebih efisien dan mampu menanggulangi kekurangan dari masing-masing kontroler tersebut.

Representasi matematika dari PI controller adalah sebagai berikut:

$$m(t) = K_p e(t) + K_i \int e(t)$$

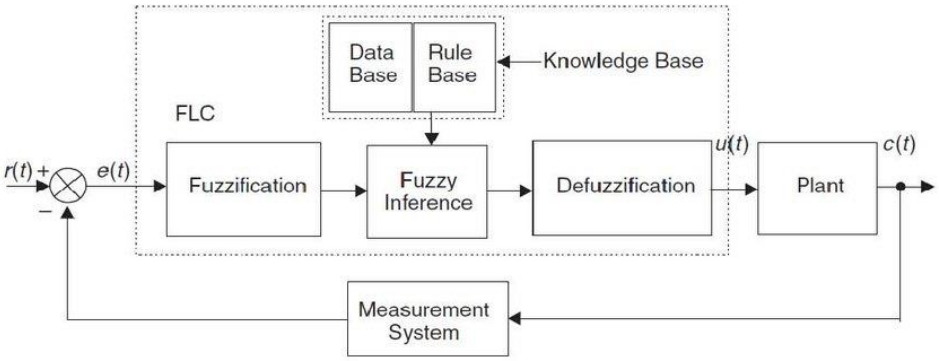
Dimana K_p merupakan konstanta proposional, K_i merupakan konstanta integral, dan e(t) merupakan error.

Berikut merupakan blok diagram yang merepresentasikan suatu PI Controller:



II.3. Fuzzy Logic Controller

Fuzzy Logic Controller(FLC) merupakan system control yang mengimplementasikan pemikiran manusia dalam sistem control nonlinear. Konsiderasi kualitatif dan heuristik yang tidak dapat dikontrol dengan teori control konvensional, dapat dikontrol dengan menambahkan logika fuzzy dalam sistem. Kontrol logika fuzzy tidak memerlukan model matematika yang akurat, dikarenakan logika fuzzy dapat mengambil input crisp, mengambil input yang tidak linear, dan tahan terhadap gangguan yang lebih baik dibandingkan dengan kontroler nonlinear lainnya. Logika fuzzy unggul dalam bidang sistem yang kompleks maupun sistem nonlinear.



Pada dasarnya, logika fuzzy mampu menghasilkan tidak hanya nilai keluaran yang berupa pembulatan (0 dan 1), namun dapat juga mengeluarkan nilai dalam rentang 0-1. Dari kemampuan untuk mengeluarkan input nilai dalam rentang tersebut, dapat dibilang logika fuzzy memiliki sistem yang memiliki kemiripan dengan logika berpikir manusia. Terdapat beberapa tahapan untuk membentuk suatu Fuzzy Logic Controller, yaitu:

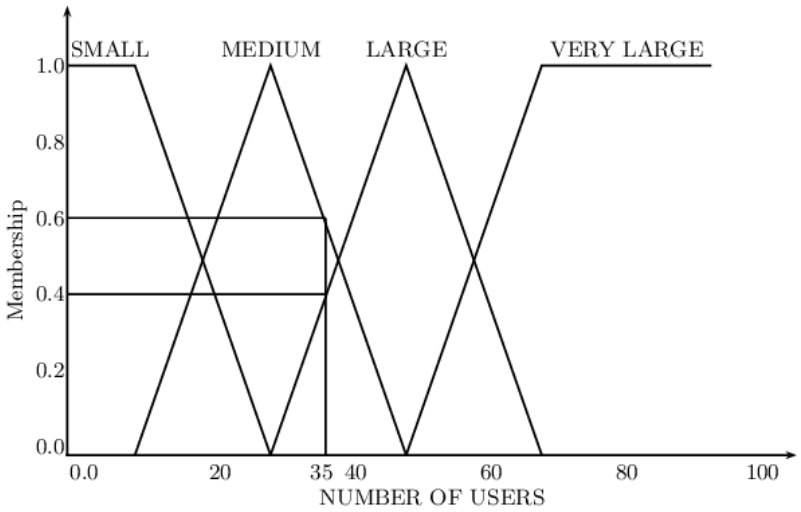
II.3.1. Fuzzification (Pre Processing)

Fuzzification merupakan langkah pertama dalam FLC, yaitu tahapan untuk mengambil nilai input crisp dan dikonversikan menjadi variabel fuzzy. Proses pengubahan tersebut dapat diubah menggunakan fungsi membership dan Degree of Membership(DOM).

Derajat Keanggotaan atau Degree of Membership merupakan suatu derajat yang dapat merepresentasikan nilai input crisp menjadi suatu nilai fuzzy. Nilai DOM sendiri berkisar dari nilai 0 dan 1.

Fungsi keanggotaan atau membership function merupakan fungsi yang mengklasifikasikan nilai-nilai DOM menjadi kelompok anggotanya masing-masing. Terdapat beberapa fungsi keanggotaan yang dapat digunakan, yaitu fungsi triangular, eksponensial, maupun gaussian.

Masing-masing nilai input yang diubah dari DOM akan dikelompokkan menjadi kelompok yang ada dengan fungsi keanggotaan. Dengan begitu, suatu nilai input tersebut akan memiliki 2 nilai, yaitu DOM dan kelompoknya masing-masing.



II.3.2. Inference Engine – Rule Base (Processing)

Rule Base merupakan aturan logika fuzzy yang disajikan dalam bentuk IF-ELSE. Biasanya, suatu rule base direpresentasikan dalam bentuk tabel dimana menggambarkan hubungan antara beberapa parameter input.

Interference engine merupakan sistem yang dapat menentukan nilai keluaran yang output dari penalaran nilai input dengan menggunakan metode AND/OR. Dengan inference engine, nilai fuzzy dapat ditentukan bagian mana saja yang akan digunakan.

Deviation	Acceleration						
	NB	NM	NS	ZE	PS	PM	PB
NB	NB	NB	NB	NB	NM	NM	NS
NM	NB	NM	NM	NM	NS	NS	ZE
NS	NM	NM	NS	NS	ZE	ZE	PS
ZE	NM	NS	NS	ZE	PS	PS	PM
PS	NS	ZE	ZE	PS	PS	PM	PM
PM	ZE	PS	PS	PM	PM	PM	PB
PB	PS	PM	PM	PB	PB	PB	PB

II.3.3. Defuzzification (Post Processing)

Defuzzification merupakan tahapan untuk mengubah nilai fuzzy yang sudah didapatkan dari tahapan sebelumnya menjadi crisp output. Defuzzyfication merupakan decision-making algorithm untuk menentukan nilai terbaik dari kelompok-kelompok fuzzy. Terdapat beberapa metode defuzzificatoin, yaitu center of gravity(COG), mean of maximum(MOM), dan center average methods(CAM). Nilai dari defuzzification dapat digunakan oleh plant sebagai hasil feedback control untuk memperbaiki error plant tersebut.

II.4. Sinusoidal Pulse Width Modulation(SPWM)

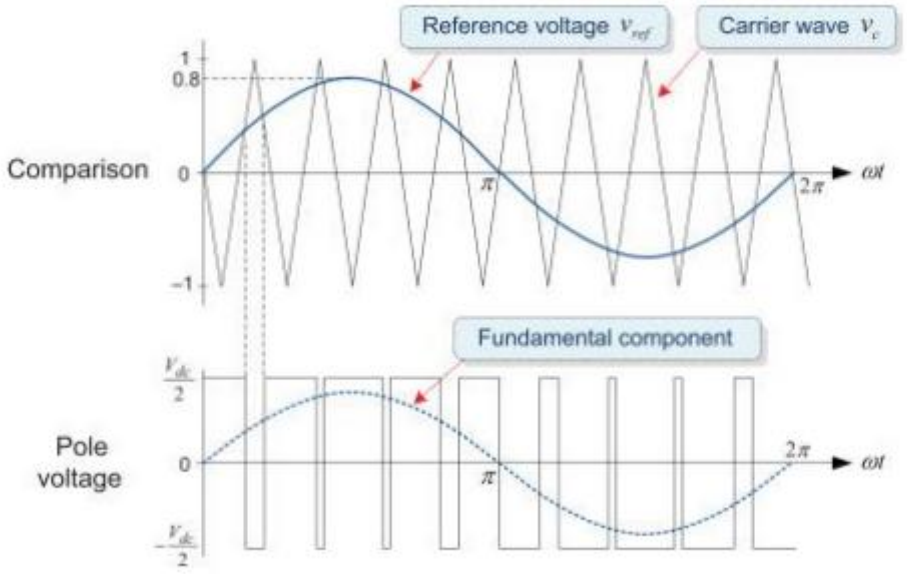
Sinusoidal Pulse Width Modulation(SPWM) merupakan sebuah teknik PWM yang umum digunakan dalam inverter. Nilai output dari inverter, yaitu sebuah tegangan AC lalu dibandingkan dengan gelombang modulasi berupa gelombang sinusoidal, dengan puncak sinyal modulasi selalu lebih kecil dari puncak sinyal pembawa.

Dalam teknik SPWM, referensi tegangan AC sinusoidal dibandingkan dengan gelombang carrier segitiga frekuensi tinggi. Perbandingan dilakuakn secara real time untuk menentukan status switching pad asetiap kutub dalam inverter. Status switching untuk setiap kutubnya dapat ditentukan dari aturan berikut:

- Tegangan referensi > Segitiga carrier : Tegangan Output = Vdc/2
- Tegangan referensi < Segitiga carrier : Tegangan Output = -Vdc/2

Output dari perbandingan yang dilakukan merupakan setengah tegangan DC input yang diberikan, dengan nilai positif untuk switch atas dan nilai negatif untuk switch bawah.

Berikut adalah ilustrasi dari Sinusoidal Pulse Width Modulation:



BAB III

SIMULASI DAN ANALISIS

Laporan ini menggunakan Simulink pada software MATLAB dengan CMEX yang memanfaatkan C language untuk membentuk blok s-function

III.1. Parameter PMSM

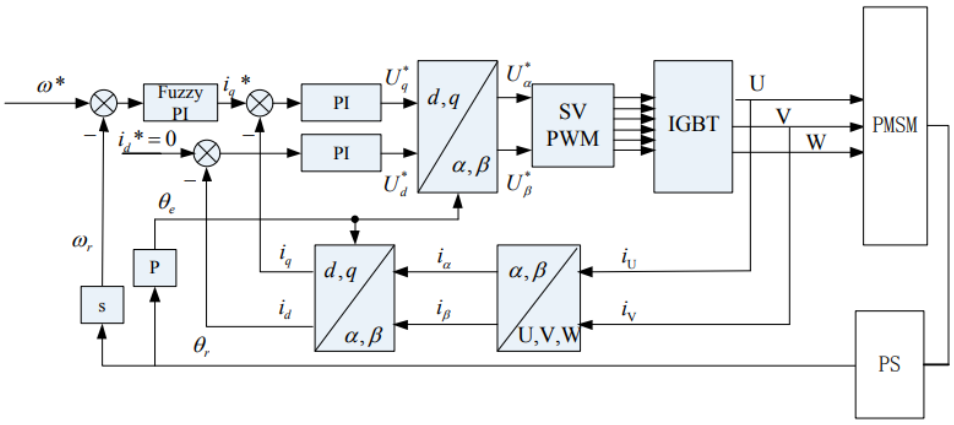
Parameter PMSM yang digunakan dalam rangkaian adalah:

Parameter	Symbol	Value	Unit
Hambatan Stator	R_s	0.55	Ω
Induktans Stator sumbu *d*	L_{sd}	0.01661	H
Induktans Stator sumbu *q*	L_{sq}	0.01622	H
Jumlah Pasangan Kutub	N	4	-
Momen Intersia Motor	J	0.01	$kgm2$

Persamaan model motor menggunakan persamaan yang terdapat pada landasan teori.

III.2. Simulasi Fuzzy Logic Controller

Fuzzy Logic Controller digunakan dalam processing setpoint nilai Kp dan Ki yang diberikan ke blok RFOC untuk memberikan nilai kecepatan arus *q*. Berikut merupakan blok diagram untuk penempatan Fuzzy Logic Controller dalam PMSM:

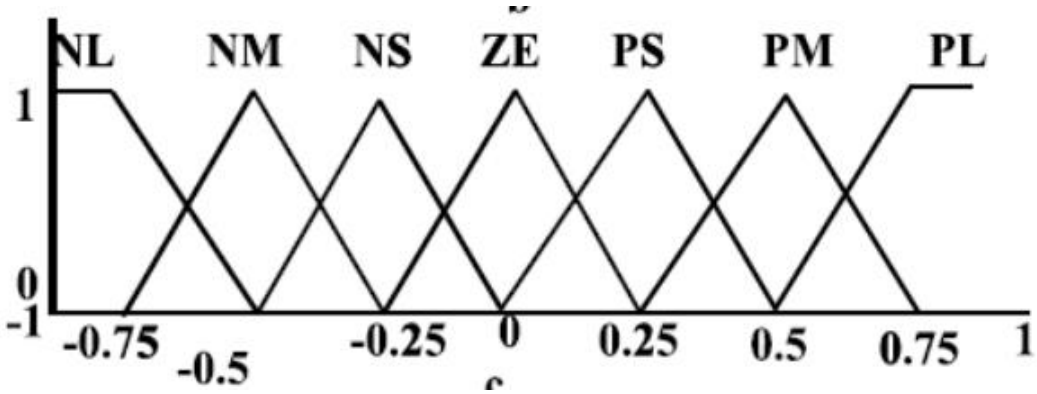


Nilai error dari Fuzzy Logic Controller didapatkan dari perbandingan antara setpoint kecepatan PMSM dan kecepatan aktual PMSM. Hasil perbandingan kemudian di proses didalam blok Fuzzy PI, untuk kemudian memberikan nilai Kp dan Ki baru yang akan ditambahkan kedalam setpoint Kp dan Ki.

Sebelum input diproses didalam blok Fuzzy, terlebih dahulu semua input di normalisasikan untuk mendapatkan performa yang terbaik. Terdapat 2 bentuk normalisasi yang digunakan, yaitu normalisasi untuk error dan normalisasi untuk change of error.

III.2.1. Degree of Membership FLC

Terdapat 1 buah fungsi keanggotaan yang digunakan dalam simulasi.yaitu 7 buah variabel linguistik, Normal Low(NL), Normal Medium(NM), Normal Small(NS), Zero(ZE), Positive Small(PS), Positive Medium(PM), dan Positive Large(PL). Membership yang digunakan dalam simulasi ini adalah memebrship function dengan ruile base 7x7.



Input yang masuk dalam blok Fuzzy, akan di preprocess terlebih dahulu menjadi variabel fuzzy. Preprocess tersebut dapat dilakukan dengan membentuk terlebih dahulu kelompok-kelompok dari variabel linguistik fuzzy, yaitu dengan menentukan daerah kelompok masing-masing. Dalam simulasi, masing-masing kelompok tersebut memiliki daerah:

NL : -0.75 x Scaler NS : -0.25 x Scaler PS : 0.25 x Scaler PL : 0.75 x Scaler

NM : -0.5 x Scaler Z : 0 x Scaler PM : 0.5 x Scaler

Untuk error, scaler tersebut diubah menjadi 10, sedangkan untuk change of error, scaler diubah menjadi 5. Setelah mendapat area masing-masing kelompok tersebut, proses fuzzyfication dapat diteruskan. Nilai feedback dari sistem sebagai input fuzzy di cek daerah kelompoknya dengan logika If-Else. Setelah itu, dihitung seberapa besar rasio input fuzzy di kelompok tersebut. Hasil tersebut akan dibandingkan dengan kelompok yang bersebrangan dengan kelompok tersebut. Proses tersebut berlaku untuk variabel change of error, sehingga akan didapatkan 2 variabel dengan kelompok dan besar rasio input di kelompok tersebut.

III.2.2. Rule Base FLC

Metode Pengendali Fuzzy Logic akan membuat suatu aturan untuk masing-masing variabel fuzzy. Aturan tersebut dibuat berdasarkan sistem yang berkaitan. Rule base untuk PMSM dibuat dengan beberapa konsiderasi, yaitu stabilitas dari sistem dan respon kecepatan PMSM.

Table.1 The Rulers of Δk_p

	3N	2N	1N	0Z	1P	2P	3P
3N	3P	3P	2P	2P	1P	0Z	0Z
2N	3P	3P	2P	1P	1P	0Z	1N
1N	2P	2P	2P	1P	0Z	1N	1N
0Z	2P	2P	1P	0Z	1N	2N	2N
1P	1P	1P	0Z	1N	2N	2N	2N
2P	1P	0Z	1N	2N	2N	2N	3N
3P	0Z	0Z	2N	2N	2N	3N	3N

Table.2 The Rulers of Δk_i

	3N	2N	1N	0Z	1P	2P	3P
3N	3N	3N	2N	2N	1N	0Z	0Z
2N	3N	3N	2N	1N	1N	0Z	0Z
1N	3N	2N	1N	1N	0Z	1P	1P
0Z	2N	2N	1N	0Z	1P	2P	2P
1P	2N	1N	0Z	1P	1P	2P	3P
2P	0Z	0Z	1P	1P	2P	3P	3P
3P	0Z	0Z	1P	2P	2P	3P	3P

Secara berurut, 3N, 2N, 1N, 0Z, 1P, 2P, dan 3P adalah NL, NM ,NS ,ZE, PS, PM, dan PL. Apabila nilai error berada dalam nilai yang tinggi, gain Kp dan Ki akan ditambakan kedalam sistem. Apabila nilai error berada dalam nilai menengah, maka parameter penentu adalah change of error, ketika arah sistem menjauh dari setpoint, gain Kp dan Ki

akan dikurangi, apabila sebaliknya, gain Kp dan Ki akan ditambahkan. Ketika nilai error cukup rendah, Gain Kp akan dikurangi sedikit dan gain Ki akan ditambahkan dengan nilai maksimum gain Ki dari Fuzzy Control. Hasil yang didapatkan kemudian akan secara berurut akan mengubah nilai setpoint gain Kp dan Ki, dengan berurut 3N, 2N, 1N, 0 , 1P, 2P, dan 3P menjadi -3,-2,-1,0,1,2,3. Nilai tersebut akan ditambahkan dengan setpoint gain Kp dan Ki dalam blok Speed Control(Spd_Ctrl).

III.3. Simulasi Sistem

Blok diagram pada SIMULINK dibuat berdasarkan desain pada paper referensi. Berikut merupakan blok diagram pada paper refferensi.

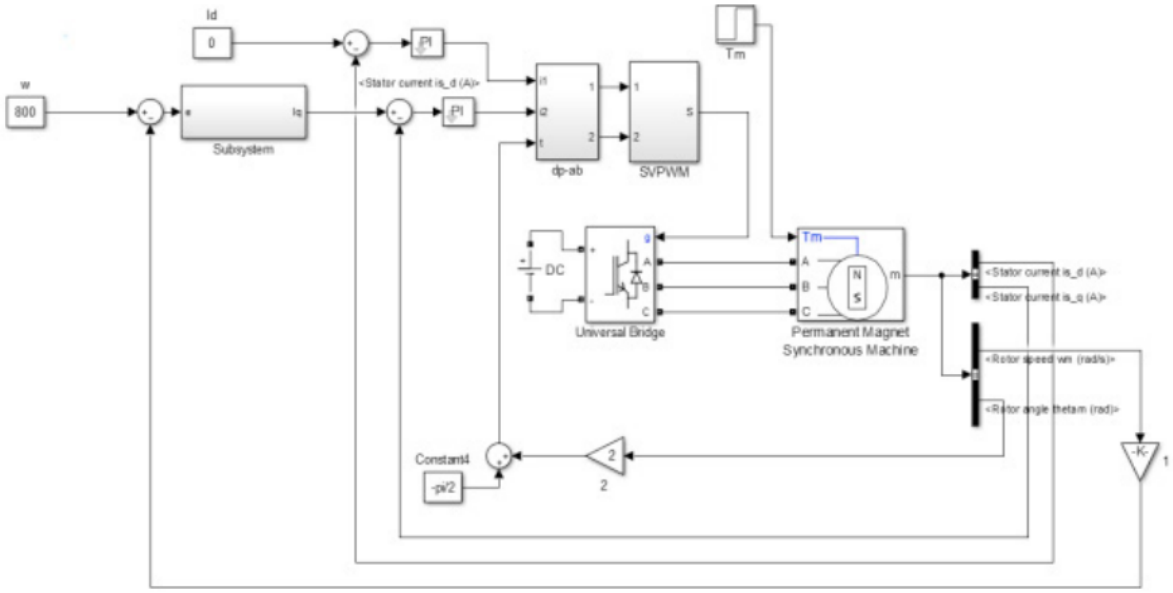
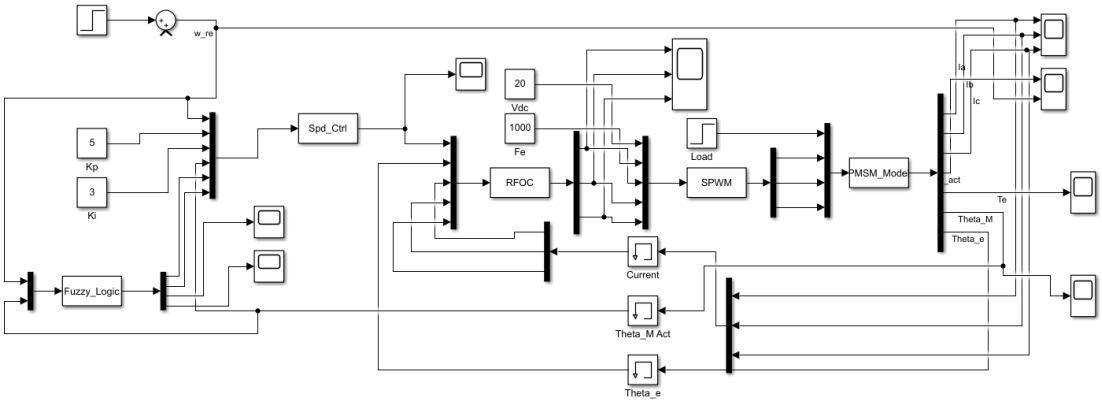


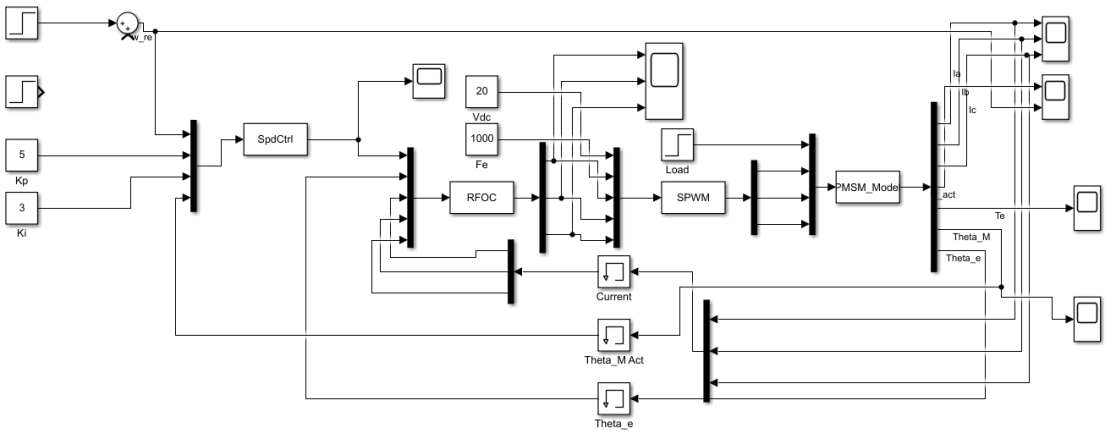
Figure.4 System Simulink Model

Blok diagram tersebut menggunakan input yaitu kecepatan referensi yang akan digunakan untuk memabndingkan nilai dengan kecepatan aktual untuk mendapatkan error kecepatan. Kecepatan aktual didapatkan dari model motor menggunakan simulasi. Hasil error tersebut dimasukkan kedalam FLC untuk menghasilkan tori sesuai dengan referensi yang diigninkan. Berikut merupakan model SIMULINK yang dibuat berdasarkan paper referensi.



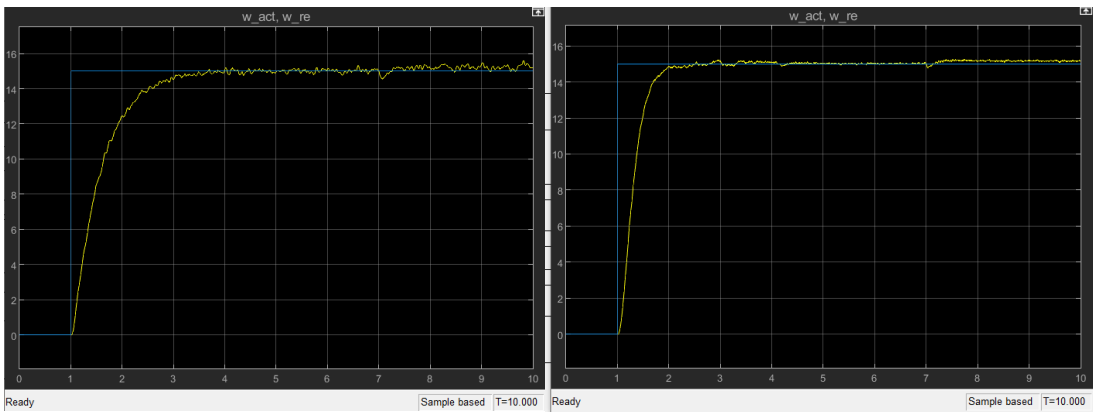
Dalam simulasi, digunakan gain Kp sebesar 5 dan gain KI sebesar 3. Dalam PMSM juga diberikan Vdc 20 Volt dengna frekuensi 1KHz. Diberikan Load sebesar 0.1 Nm dalam waktu 7 detik.

Berikut merupakan simulasi PMSM tanpa menggunakan Fuzzy Logic Controller:



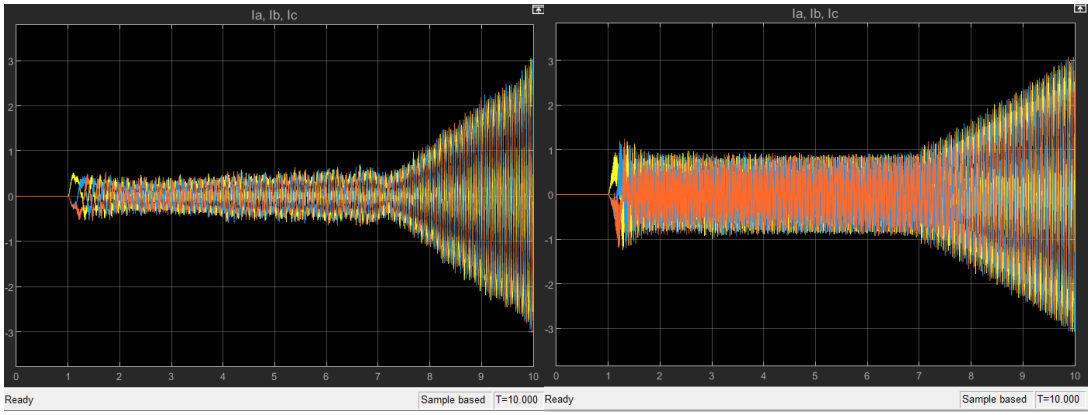
Hasil yang diperoleh dari masing-masing simulasi tersebut sebagai berikut.

- Kecepatan



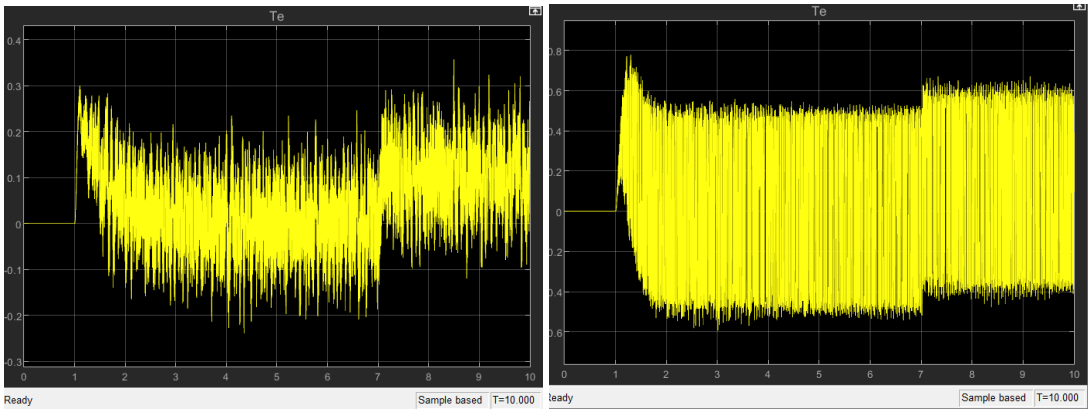
Tanpa Fuzzy Logic Dengan Fuzzy Logic

- Arus AC



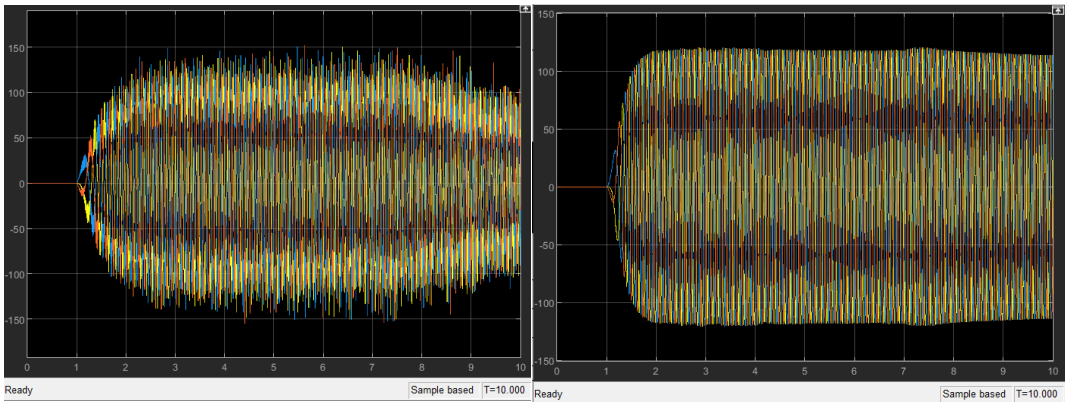
Tanpa Fuzzy Logic Dengan Fuzzy Logic

- Torsi Elektromagnetik



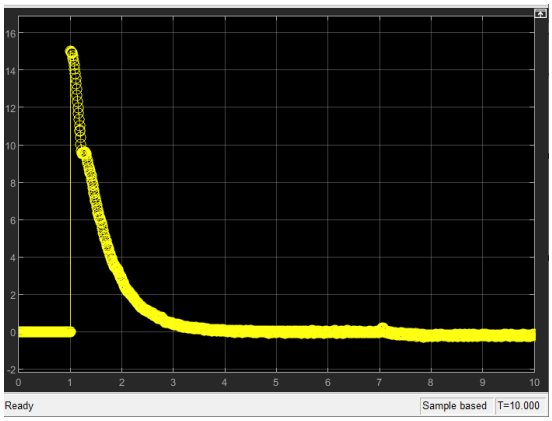
Tanpa Fuzzy Logic Dengan Fuzzy Logic

- Tegangan AC

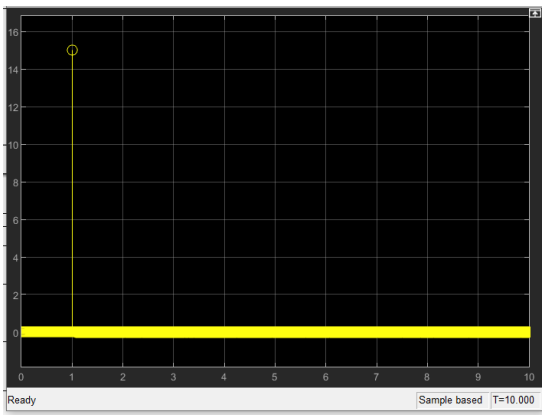


Tanpa Fuzzy Logic Dengan Fuzzy Logic

- Fuzzy Logic Error dan Fuzzy Logic Change of Error



Error



Change of Error

Secara garis besar, hasil yang didapat menggunakan Fuzzy Logic Controller sudah memenuhi kriteria dari paper referensi. Terlihat bahwa dengan menggunakan Fuzzy Logic Controller, sistem terlihat lebih stabil dan memiliki transient response yang lebih baik jika dibandingkan dengan tidak menggunakan Fuzzy Logic Controller. Terlihat dengan menggunakan Fuzzy Logic Controller, daya yang digunakan bertambah yang ditandai dengan arus dan tegangan yang lebih besar dibandingkan dengan hanya menggunakan kontroler PI tradisional saja. Namun, masih terdapat beberapa kekurangan dalam model yang dibuat. Kekurangan tersebut dapat ditimbulkan dari tidak disediakannya scaling factor untuk FLC dalam paper.

BAB IV

KESIMPULAN

Dari simulasi yang telah dilakukan, dapat disimpulkan bahwa Fuzzy Logic Controller merupakan suatu kontroler non konvensional yang mempunyai kemampuan untuk mengimplementasikan cara berfikir manusia kedalam suatu sistem, sehingga cocok diterapkan dalam suatu sistem non linear yang sulit untuk dikontrol dengan kontroler kovensional. Pada sistem servo PMSM, FLC dapat diimplementasikan sebagai pengenali keceaptan dengan memasukkan nilai selisih dari kecepatan referensi dengan nilai kecepatan aktual. Rule base dari FLC didapatkan dari menganalisa terlebih dahulu respon dari PMSM. Hasil simulasi yang didapatkan menunjukkan bahwa Fuzzy Logic Controller mampu memperbaiki sistem menjadi lebih baik dengan memperbaiki transient response dan kestabilitas sistem jika dibandingkan dengan menggunakan PI controller tradisional.

DAFTAR PUSTAKA

[1] Gu, D. W., Yao, Y., Zhang, D. M., Cui, Y. B., & Zeng, F. Q. (2020). MATLAB/simulink based modeling and simulation of Fuzzy Pi Control for PMSM. *Procedia Computer Science*, *166*, 195–199.

[2] Du Changqing, Zeng Hongxia, Wu Dongmei, et al. Contrastive simulation study on control strategy of permanent magnet synchronous motor [J].Digital Manufacturing Science, 2018(1): 34-41.

[3] Sun Fangchao, Du Xing, Wang WenMai, et al. Research on optimization strategy of PMSM servo system based on ADRC [J]. Manufacturing Automation, 2018, 40 (3): 37-38.

[4] Cui Jiarui, Li Qing, Zhang Bo, et al. Synthesis of Predictive Control for LPV system with Bounded Disturbances [J]. Chinese Journal of Electrical Engineering, 2013, 33 (z1).

[5] Li Hongwei, Jian Guihui, Shang Zeming. Design of Variable Universe Fuzzy Adap tive PID Control System for Brushless DC motor [J]. Journal of Jiangnan University: Natural Science Edition, 2009, 8 (1): 49-53.

[6] Yuan Yadeng, Meng Huilei, Feng Qianlong, et al. [J]. Simulation Research on Vec tor Control of Permanent Magnet Synchronous Motor Used in Electric Vehicle Based on Fuzzy PID, 2018, No. 263 (08): 14-17

[7] Song Shu, Gong Jianguo, Lin Wei, et al. Space Vector Control System Modeling and Simulation of Permanent Magnet Synchronous Motor for Pure Electric Vehicle [J]. Journal of Wuhan University of Technology, 2012, 34 (4)

[8] *Permanent magnet synchronous motor (PMSM)*. element14. (n.d.). Retrieved June 9, 2022, from https://my.element14.com/motor-control-permanent-magnet-sync-motor-pmsm-technology

[9] says, B. R. R. L., & Lingampalli, B. R. R. (2020, January 11). *What is proportional integral (PI) controller*. Electronics Coach. Retrieved June 9, 2022, from https://electronicscoach.com/proportional-integral-controller.html#:~:text=In%20the%20proportional%2Dintegral%20controller,with%20each%20one%20of%20them/.

[10] *Defuzzification*. Defuzzification - an overview | ScienceDirect Topics. (n.d.). Retrieved June 9, 2022, from https://www.sciencedirect.com/topics/engineering/defuzzification/

LAMPIRAN

Fuzzy_Logic.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME Fuzzy_Logic  // s-Function Block Name
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element]) /* Pointer to Input Port0 */

static void mdlInitializeSizes(SimStruct *S) {
    ssSetNumDiscStates(S, 5);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);   // Number of Inputs
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 4);  // Number of Outputs
    ssSetNumSampleTimes(S, 1);

    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE
    | SS_OPTION_DISCRETE_VALUED_OUTPUT));
}

static void mdlInitializeSampleTimes(SimStruct *S) {
    ssSetSampleTime(S, 0, 1e-4);    // Continuous: CONTINUOUS_SAMPLE_TIME
OR 1e-3|| Discrete: 0.001
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S) {
    real_T *X0 = ssGetRealDiscStates(S);
    int_T nXStates = ssGetNumDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    int_T i;

    /* Initialize the states to 0.0 */
    for (i=0; i < nXStates; i++) X0[i] = 0.0;
}

static void mdlOutputs(SimStruct *S, int_T tid) {
    real_T *Y = ssGetOutputPortRealSignal(S,0);
    real_T *X = ssGetRealDiscStates(S);

    real_T delta_Kp = X[1];
    real_T delta_Ki = X[2];
    Y[0] = delta_Kp;
    Y[1] = delta_Ki;
    Y[2] = X[0];
    Y[3] = X[4];

}

#define MDL_UPDATE
```

```
static void mdlUpdate(SimStruct *S, int_T tid) {
    real_T *X = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    real_T dt = 1e-3;

    // Inputs
    real_T w_ref = U(0);
    real_T theta_m = U(1);
    real_T theta_old = X[3];

    real_T w_act = (theta_m - theta_old)/dt;
    real_T error_prev = X[0];

    real_T error_scaling = 10;
    real_T change_error_scaling = 10;

    // Fuzzy Logic Variables
    real_T reg_NL, reg_NM, reg_NS, reg_Z, reg_PS, reg_PM, reg_PL;

    real_T err_MF1, err_MF1_DOM, err_MF2, err_MF2_DOM;
    real_T err_MF1_1, err_MF1_2, err_MF2_1, err_MF2_2;

    real_T ce_MF1, ce_MF1_DOM, ce_MF2, ce_MF2_DOM;
    real_T ce_MF1_1, ce_MF1_2, ce_MF2_1, ce_MF2_2;

    real_T delta_Kp ,delta_Ki;

    // Error
    real_T error = w_ref - w_act;
    real_T error_out = error;

    // Change of error
    real_T change_error = (error - error_prev);
    real_T change_error_out = change_error;

    // Normalization
    error = (error - error_scaling)/error_scaling;
    change_error = (change_error -
change_error_scaling)/change_error_scaling;

    //DEGREE OF MEMBERSHIP

    // Degree Of Membership ERROR
    reg_NL  = -0.75*error_scaling;
    reg_NM = -0.5 * error_scaling;
    reg_NS =  -0.25*error_scaling;
    reg_Z   = 0.00 *error_scaling;
    reg_PS = 0.25*error_scaling;
    reg_PM = 0.5 * error_scaling;
    reg_PL  = 0.75*error_scaling;

    // FUZZYFICATION of Errors
    if (error <= reg_NL){
        err_MF1 = reg_NL;  // Lower NL Upper NL
        err_MF2 = reg_NL;  // Lower NL Upper NL
        err_MF1_DOM = 1;
        err_MF2_DOM = 1;

    }
    else if (error >= reg_PL){
```

```
    err_MF1 = reg_PL;  // Lower PL Upper PL
    err_MF2 = reg_PL;  // Lower PL Upper PL
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (error == reg_NM){
    err_MF1 = reg_NM;  // Lower NM Upper NM
    err_MF2 = reg_NM;  // Lower NM Upper NM
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (error == reg_NS){
    err_MF1 = reg_NS;  // Lower NS Upper NS
    err_MF2 = reg_NS;  // Lower NS Upper NS
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (error == reg_Z){
    err_MF1 = reg_Z;  // Lower Z Upper Z
    err_MF2 = reg_Z;  // Lower Z Upper Z
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (error == reg_PS){
    err_MF1 = reg_PS;  // Lower PS Upper PS
    err_MF2 = reg_PS;  // Lower PS Upper PS
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (error == reg_PM){
    err_MF1 = reg_PM;  // Lower PM Upper PM
    err_MF2 = reg_PM;  // Lower PM Upper PM
    err_MF1_DOM = 1;
    err_MF2_DOM = 1;
}
else if (reg_NL < error && error < reg_Z){
    if (reg_NL < error && error < reg_NM){
        err_MF1 = reg_NL; // Lower NL Upper NL
        err_MF2 = reg_NM; // Lower NL Upper NS
        err_MF1_DOM = (reg_NM - error)/(reg_NM - reg_NL);

        err_MF2_1 = (error-reg_NL)/(reg_NM - reg_NL);
        err_MF2_2 = (reg_NS-error)/(reg_NS-reg_NM);
        if (err_MF2_1 > err_MF2_2) err_MF2_DOM = err_MF2_2;
        else if (err_MF2_1 < err_MF2_2) err_MF2_DOM = err_MF2_1;
        if (err_MF2_DOM > 0) err_MF2_DOM = err_MF2_DOM;
        else if (err_MF2_DOM < 0) err_MF2_DOM = 0;
    }
    else if (reg_NM < error && error < reg_NS){
        err_MF1 = reg_NM;  // Lower NL Upper NS
        err_MF2 = reg_NS;  // Lower NM Upper Z

        err_MF1_1 = (error-reg_NL)/(reg_NM - reg_NL);
        err_MF1_2 = (reg_NS-reg_NL)/(reg_NS-reg_NM);

        err_MF2_1 = (error-reg_NM)/(reg_NS - reg_NM);
        err_MF2_2 = (reg_Z-error)/(reg_Z-reg_NS);

        if (err_MF1_1 > err_MF1_2) err_MF1_DOM = err_MF1_2;
        else if (err_MF1_1 < err_MF1_2) err_MF1_DOM = err_MF1_1;
        if (err_MF1_DOM > 0) err_MF1_DOM = err_MF1_DOM;
```

```
            else if (err_MF1_DOM < 0) err_MF1_DOM = 0;

            if (err_MF2_1 > err_MF2_2) err_MF2_DOM = err_MF2_2;
            else if (err_MF2_1 < err_MF2_2) err_MF2_DOM = err_MF2_1;
            if (err_MF2_DOM > 0) err_MF2_DOM = err_MF2_DOM;
            else if (err_MF2_DOM < 0) err_MF2 = 0;
        }
        else if (reg_NS < error && error < reg_Z){
            err_MF1 = reg_NS;  // Lower NM Upper Z
            err_MF2 = reg_Z;   // Lower NS Upper PS

            err_MF1_1 = (error-reg_NM)/(reg_NS - reg_NM);
            err_MF1_2 = (reg_Z-error)/(reg_Z-reg_NS);

            err_MF2_1 = (error-reg_NS)/(reg_Z - reg_NS);
            err_MF2_2 = (reg_PS-error)/(reg_PS-reg_Z);

            if (err_MF1_1 > err_MF1_2) err_MF1_DOM = err_MF1_2;
            else if (err_MF1_1 < err_MF1_2) err_MF1_DOM = err_MF1_1;
            if (err_MF1_DOM > 0) err_MF1_DOM = err_MF1_DOM;
            else if (err_MF1_DOM < 0) err_MF1_DOM = 0;

            if (err_MF2_1 > err_MF2_2) err_MF2_DOM = err_MF2_2;
            else if (err_MF2_1 < err_MF2_2) err_MF2_DOM = err_MF2_1;
            if (err_MF2_DOM > 0) err_MF2_DOM = err_MF2_DOM;
            else if (err_MF2_DOM < 0) err_MF2_DOM = 0;
        }
    }
    else if (reg_Z < error && error < reg_PL){
        if (reg_Z < error && error < reg_NS){
            err_MF1 = reg_Z;  // Lower NS Upper PS
            err_MF2 = reg_PS;   // Lower Z Upper PM

            err_MF1_1 = (error-reg_NS)/(reg_Z - reg_NS);
            err_MF1_2 = (reg_PS-error)/(reg_PS-reg_Z);

            err_MF2_1 = (error-reg_Z)/(reg_PS - reg_Z);
            err_MF2_2 = (reg_PM-error)/(reg_PM-reg_PS);


            if (err_MF1_1 > err_MF1_2) err_MF1_DOM = err_MF1_2;
            else if (err_MF1_1 < err_MF1_2) err_MF1_DOM = err_MF1_1;
            if (err_MF1_DOM > 0) err_MF1_DOM = err_MF1_DOM;
            else if (err_MF1_DOM < 0) err_MF1_DOM = 0;

            if (err_MF2_1 > err_MF2_2) err_MF2_DOM = err_MF2_2;
            else if (err_MF2_1 < err_MF2_2) err_MF2_DOM = err_MF2_1;
            if (err_MF2_DOM > 0) err_MF2_DOM = err_MF2_DOM;
            else if (err_MF2_DOM < 0) err_MF2_DOM = 0;
        }
        else if (reg_NS < error && error < reg_PM){
            err_MF1 = reg_PS;  // Lower Z Upper PM
            err_MF2 = reg_PM;   // Lower PS Upper PL

            err_MF1_1 = (error-reg_Z)/(reg_PS - reg_Z);
            err_MF1_2 = (reg_PM-error)/(reg_PM-reg_PS);

            err_MF2_1 = (error-reg_PS)/(reg_PM - reg_PS);
            err_MF2_2 = (reg_PL-error)/(reg_PL-reg_PM);
```

```
            if (err_MF1_1 > err_MF1_2) err_MF1_DOM = err_MF1_2;
            else if (err_MF1_1 < err_MF1_2) err_MF1_DOM = err_MF1_1;
            if (err_MF1_DOM > 0) err_MF1_DOM = err_MF1_DOM;
            else if (err_MF1_DOM < 0) err_MF1_DOM = 0;

            if (err_MF2_1 > err_MF2_2) err_MF2_DOM = err_MF2_2;
            else if (err_MF2_1 < err_MF2_2) err_MF2_DOM = err_MF2_1;
            if (err_MF2_DOM > 0) err_MF2_DOM = err_MF2_DOM;
            else if (err_MF2_DOM < 0) err_MF2_DOM = 0;
        }
        else if (reg_PM < error < reg_PL){
            err_MF1 = reg_PM;  // Lower PS Upper PL
            err_MF2 = reg_PL;  // Lower PL Upper PL

            err_MF1_1 = (error-reg_PS)/(reg_PM - reg_PS);
            err_MF1_2 = (reg_PL-error)/(reg_PL-reg_PM);

            err_MF2_DOM = (reg_PL - error)/(reg_PL - reg_PM);


            if (err_MF1_1 > err_MF1_2) err_MF1_DOM = err_MF1_2;
            else if (err_MF1_1 < err_MF1_2) err_MF1_DOM = err_MF1_1;
            if (err_MF1_DOM > 0) err_MF1_DOM = err_MF1_DOM;
            else if (err_MF1_DOM < 0) err_MF1_DOM = 0;
            }
    }


// FUZZYFICATION of Change of Errors

// Degree Of Membership CHANGE OF ERROR
    reg_NL  = -0.75*change_error_scaling;
    reg_NM = -0.5 * change_error_scaling;
    reg_NS =  -0.25*change_error_scaling;
    reg_Z   = 0.00 *change_error_scaling;
    reg_PS = 0.25*change_error_scaling;
    reg_PM = 0.5 * change_error_scaling;
    reg_PL  = 0.75*change_error_scaling;

    if (change_error <= reg_NL){
        ce_MF1 = reg_NL;  // Lower NL Upper NL
        ce_MF2 = reg_NL;  // Lower NL Upper NL
        ce_MF1_DOM = 1;
        ce_MF2_DOM = 1;
    }
    else if (change_error >= reg_PL){
        ce_MF1 = reg_PL;  // Lower PL Upper PL
        ce_MF2 = reg_PL;  // Lower PL Upper PL
        ce_MF1_DOM = 1;
        ce_MF2_DOM = 1;
    }
    else if (change_error == reg_NM){
        ce_MF1 = reg_NM;  // Lower NM Upper NM
        ce_MF2 = reg_NM;  // Lower NM Upper NM
        ce_MF1_DOM = 1;
        ce_MF2_DOM = 1;
    }
    else if (change_error == reg_NS){
        ce_MF1 = reg_NS;  // Lower NS Upper NS
        ce_MF2 = reg_NS;  // Lower NS Upper NS
        ce_MF1_DOM = 1;
```

```
    ce_MF2_DOM = 1;
}
else if (change_error == reg_Z){
    ce_MF1 = reg_Z;  // Lower Z Upper Z
    ce_MF2 = reg_Z;  // Lower Z Upper Z
    ce_MF1_DOM = 1;
    ce_MF2_DOM = 1;
}
else if (change_error == reg_PS){
    ce_MF1 = reg_PS;  // Lower PS Upper PS
    ce_MF2 = reg_PS;  // Lower PS Upper PS
    ce_MF1_DOM = 1;
    ce_MF2_DOM = 1;
}
else if (change_error == reg_PM){
    ce_MF1 = reg_PM;  // Lower PM Upper PM
    ce_MF2 = reg_PM;  // Lower PM Upper PM
    ce_MF1_DOM = 1;
    ce_MF2_DOM = 1;
}
else if (reg_NL < change_error && change_error < reg_Z){
    if (reg_NL < change_error && change_error < reg_NM){
        ce_MF1 = reg_NL; // Lower NL Upper NL
        ce_MF2 = reg_NM; // Lower NL Upper NS
        ce_MF1_DOM = (reg_NM - change_error)/(reg_NM - reg_NL);

        ce_MF2_1 = (change_error-reg_NL)/(reg_NM - reg_NL);
        ce_MF2_2 = (reg_NS-change_error)/(reg_NS-reg_NM);
        if (ce_MF2_1 > ce_MF2_2) ce_MF2_DOM = ce_MF2_2;
        else if (ce_MF2_1 < ce_MF2_2) ce_MF2_DOM = ce_MF2_1;
        if (ce_MF2_DOM > 0) ce_MF2_DOM = ce_MF2_DOM;
        else if (ce_MF2_DOM < 0) ce_MF2_DOM = 0;
    }
    else if (reg_NM < change_error && change_error < reg_NS){
        ce_MF1 = reg_NM;  // Lower NL Upper NS
        ce_MF2 = reg_NS;  // Lower NM Upper Z

        ce_MF1_1 = (change_error-reg_NL)/(reg_NM - reg_NL);
        ce_MF1_2 = (reg_NS-reg_NL)/(reg_NS-reg_NM);

        ce_MF2_1 = (change_error-reg_NM)/(reg_NS - reg_NM);
        ce_MF2_2 = (reg_Z-change_error)/(reg_Z-reg_NS);

        if (ce_MF1_1 > ce_MF1_2) ce_MF1_DOM = ce_MF1_2;
        else if (ce_MF1_1 < ce_MF1_2) ce_MF1_DOM = ce_MF1_1;
        if (ce_MF1_DOM > 0) ce_MF1_DOM = ce_MF1_DOM;
        else if (ce_MF1_DOM < 0) ce_MF1_DOM = 0;

        if (ce_MF2_1 > ce_MF2_2) ce_MF2_DOM = ce_MF2_2;
        else if (ce_MF2_1 < ce_MF2_2) ce_MF2_DOM = ce_MF2_1;
        if (ce_MF2_DOM > 0) ce_MF2_DOM = ce_MF2_DOM;
        else if (ce_MF2_DOM < 0) ce_MF2 = 0;
    }
    else if (reg_NS < change_error && change_error < reg_Z){
        ce_MF1 = reg_NS;  // Lower NM Upper Z
        ce_MF2 = reg_Z;   // Lower NS Upper PS

        ce_MF1_1 = (change_error-reg_NM)/(reg_NS - reg_NM);
        ce_MF1_2 = (reg_Z-change_error)/(reg_Z-reg_NS);

        ce_MF2_1 = (change_error-reg_NS)/(reg_Z - reg_NS);
```

```
            ce_MF2_2 = (reg_PS-change_error)/(reg_PS-reg_Z);

            if (ce_MF1_1 > ce_MF1_2) ce_MF1_DOM = ce_MF1_2;
            else if (ce_MF1_1 < ce_MF1_2) ce_MF1_DOM = ce_MF1_1;
            if (ce_MF1_DOM > 0) ce_MF1_DOM = ce_MF1_DOM;
            else if (ce_MF1_DOM < 0) ce_MF1_DOM = 0;

            if (ce_MF2_1 > ce_MF2_2) ce_MF2_DOM = ce_MF2_2;
            else if (ce_MF2_1 < ce_MF2_2) ce_MF2_DOM = ce_MF2_1;
            if (ce_MF2_DOM > 0) ce_MF2_DOM = ce_MF2_DOM;
            else if (ce_MF2_DOM < 0) ce_MF2_DOM = 0;
        }

    }
    else if (reg_Z < change_error && change_error < reg_PL){
        if (reg_Z < change_error && change_error < reg_NS){
            ce_MF1 = reg_Z;  // Lower NS Upper PS
            ce_MF2 = reg_PS;   // Lower Z Upper PM

            ce_MF1_1 = (change_error-reg_NS)/(reg_Z - reg_NS);
            ce_MF1_2 = (reg_PS-change_error)/(reg_PS-reg_Z);

            ce_MF2_1 = (change_error-reg_Z)/(reg_PS - reg_Z);
            ce_MF2_2 = (reg_PM-change_error)/(reg_PM-reg_PS);


            if (ce_MF1_1 > ce_MF1_2) ce_MF1_DOM = ce_MF1_2;
            else if (ce_MF1_1 < ce_MF1_2) ce_MF1_DOM = ce_MF1_1;
            if (ce_MF1_DOM > 0) ce_MF1_DOM = ce_MF1_DOM;
            else if (ce_MF1_DOM < 0) ce_MF1_DOM = 0;

            if (ce_MF2_1 > ce_MF2_2) ce_MF2_DOM = ce_MF2_2;
            else if (ce_MF2_1 < ce_MF2_2) ce_MF2_DOM = ce_MF2_1;
            if (ce_MF2_DOM > 0) ce_MF2_DOM = ce_MF2_DOM;
            else if (ce_MF2_DOM < 0) ce_MF2_DOM = 0;
        }
        else if (reg_NS < change_error && change_error < reg_PM){
            ce_MF1 = reg_PS;  // Lower Z Upper PM
            ce_MF2 = reg_PM;   // Lower PS Upper PL

            ce_MF1_1 = (change_error-reg_Z)/(reg_PS - reg_Z);
            ce_MF1_2 = (reg_PM-change_error)/(reg_PM-reg_PS);

            ce_MF2_1 = (change_error-reg_PS)/(reg_PM - reg_PS);
            ce_MF2_2 = (reg_PL-change_error)/(reg_PL-reg_PM);


            if (ce_MF1_1 > ce_MF1_2) ce_MF1_DOM = ce_MF1_2;
            else if (ce_MF1_1 < ce_MF1_2) ce_MF1_DOM = ce_MF1_1;
            if (ce_MF1_DOM > 0) ce_MF1_DOM = ce_MF1_DOM;
            else if (ce_MF1_DOM < 0) ce_MF1_DOM = 0;

            if (ce_MF2_1 > ce_MF2_2) ce_MF2_DOM = ce_MF2_2;
            else if (ce_MF2_1 < ce_MF2_2) ce_MF2_DOM = ce_MF2_1;
            if (ce_MF2_DOM > 0) ce_MF2_DOM = ce_MF2_DOM;
            else if (ce_MF2_DOM < 0) ce_MF2_DOM = 0;
        }
        else if (reg_PM < change_error && change_error < reg_PL){
            ce_MF1 = reg_PM;  // Lower PS Upper PL
            ce_MF2 = reg_PL;  // Lower PL Upper PL
```

```
            ce_MF1_1 = (change_error-reg_PS)/(reg_PM - reg_PS);
            ce_MF1_2 = (reg_PL-change_error)/(reg_PL-reg_PM);

            ce_MF2_DOM = (reg_PL - change_error)/(reg_PL - reg_PM);

            if (ce_MF1_1 > ce_MF1_2) ce_MF1_DOM = ce_MF1_2;
            else if (ce_MF1_1 < ce_MF1_2) ce_MF1_DOM = ce_MF1_1;
            if (ce_MF1_DOM > 0) ce_MF1_DOM = ce_MF1_DOM;
            else if (ce_MF1_DOM < 0) ce_MF1_DOM = 0;
            }
    }

    real_T OMF_KP, OMF_KI, err_OMF_DOM, err_OMF, ce_OMF_DOM, ce_OMF;

    // Inference Engine
    // Max Algorithm
    if (err_MF1_DOM > err_MF2_DOM){err_OMF = err_MF1; err_OMF_DOM =
err_MF1_DOM;}
    else if (err_MF1_DOM < err_MF2_DOM){err_OMF = err_MF2; err_OMF_DOM =
err_MF2_DOM;}
    else if (err_MF1_DOM == err_MF2_DOM){err_OMF = err_MF1; err_OMF_DOM =
1;}

    if (ce_MF1_DOM > ce_MF2_DOM){ce_OMF = ce_MF1; ce_OMF_DOM = ce_MF1_DOM;}
    else if (ce_MF1_DOM < ce_MF2_DOM){ce_OMF = ce_MF2; ce_OMF_DOM =
ce_MF2_DOM;}
    else if (ce_MF1_DOM == ce_MF2_DOM){ce_OMF = ce_MF1; ce_OMF_DOM = 1;}

    // RULE BASE KP KI
    if (ce_OMF == reg_NL){
        if (err_OMF == reg_NL){OMF_KP = reg_PL; OMF_KI = reg_NL;}
        else if (err_OMF == reg_NM){OMF_KP = reg_PL; OMF_KI = reg_NL;}
        else if (err_OMF == reg_NS){OMF_KP = reg_PM; OMF_KI = reg_NL;}
        else if (err_OMF == reg_Z){OMF_KP = reg_PM; OMF_KI = reg_NM;}
        else if (err_OMF == reg_PS){OMF_KP = reg_PS; OMF_KI = reg_NM;}
        else if (err_OMF == reg_PM){OMF_KP = reg_PS; OMF_KI = reg_Z;}
        else if (err_OMF == reg_PL){OMF_KP = reg_Z; OMF_KI = reg_Z;}
    }
    else if (ce_OMF == reg_NM){
        if (err_OMF == reg_NL){OMF_KP = reg_PL; OMF_KI = reg_NL;}
        else if (err_OMF == reg_NM){OMF_KP = reg_PL; OMF_KI = reg_NL;}
        else if (err_OMF == reg_NS){OMF_KP = reg_PM; OMF_KI = reg_NM;}
        else if (err_OMF == reg_Z){OMF_KP = reg_PM; OMF_KI = reg_NM;}
        else if (err_OMF == reg_PS){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_PM){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_PL){OMF_KP = reg_Z; OMF_KI = reg_Z;}
    }
    else if (ce_OMF == reg_NS){
        if (err_OMF == reg_NL){OMF_KP = reg_PM; OMF_KI = reg_NM;}
        else if (err_OMF == reg_NM){OMF_KP = reg_PM; OMF_KI = reg_NM;}
        else if (err_OMF == reg_NS){OMF_KP = reg_PM; OMF_KI = reg_NS;}
        else if (err_OMF == reg_Z){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_PS){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_PM){OMF_KP = reg_NS; OMF_KI = reg_PS;}
        else if (err_OMF == reg_PL){OMF_KP = reg_NM; OMF_KI = reg_PS;}
    }
    else if (ce_OMF == reg_Z){
        if (err_OMF == reg_NL){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_NM){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_NS){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_Z){OMF_KP = reg_Z; OMF_KI = reg_Z;}
```

```
        else if (err_OMF == reg_PS){OMF_KP = reg_NS; OMF_KI = reg_PS;}
        else if (err_OMF == reg_PM){OMF_KP = reg_NM; OMF_KI = reg_PS;}
        else if (err_OMF == reg_PL){OMF_KP = reg_NM; OMF_KI = reg_PM;}
    }
    else if (ce_OMF == reg_PS){
        if (err_OMF == reg_NL){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_NM){OMF_KP = reg_PS; OMF_KI = reg_NS;}
        else if (err_OMF == reg_NS){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_Z){OMF_KP = reg_NS; OMF_KI = reg_PS;}
        else if (err_OMF == reg_PS){OMF_KP = reg_NM; OMF_KI = reg_PS;}
        else if (err_OMF == reg_PM){OMF_KP = reg_NM; OMF_KI = reg_PM;}
        else if (err_OMF == reg_PL){OMF_KP = reg_NM; OMF_KI = reg_PM;}
    }
    else if (ce_OMF == reg_PM){
        if (err_OMF == reg_NL){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_NM){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_NS){OMF_KP = reg_NS; OMF_KI = reg_PS;}
        else if (err_OMF == reg_Z){OMF_KP = reg_NM; OMF_KI = reg_PM;}
        else if (err_OMF == reg_PS){OMF_KP = reg_NM; OMF_KI = reg_PM;}
        else if (err_OMF == reg_PM){OMF_KP = reg_NM; OMF_KI = reg_PL;}
        else if (err_OMF == reg_PL){OMF_KP = reg_NL; OMF_KI = reg_PL;}
    }
    else if (ce_OMF == reg_PL){
        if (err_OMF == reg_NL){OMF_KP = reg_Z; OMF_KI = reg_Z;}
        else if (err_OMF == reg_NM){OMF_KP = reg_NS; OMF_KI = reg_Z;}
        else if (err_OMF == reg_NS){OMF_KP = reg_NS; OMF_KI = reg_PS;}
        else if (err_OMF == reg_Z){OMF_KP = reg_NM; OMF_KI = reg_PM;}
        else if (err_OMF == reg_PS){OMF_KP = reg_NM; OMF_KI = reg_PL;}
        else if (err_OMF == reg_PM){OMF_KP = reg_NL; OMF_KI = reg_PL;}
        else if (err_OMF == reg_PL){OMF_KP = reg_NL; OMF_KI = reg_PL;}
    }

    //real_T out_scale = err_OMF_DOM * 0.45 + ce_OMF_DOM * 0.55;
    if (OMF_KP == reg_NL){
        delta_Kp = -3;}
    else if (OMF_KP == reg_NM){delta_Kp = -2;}
    else if (OMF_KP == reg_NS){delta_Kp = -1;}
    else if (OMF_KP == reg_Z){delta_Kp = 0;}
    else if (OMF_KP == reg_PS){delta_Kp = 1;}
    else if (OMF_KP == reg_PM){delta_Kp = 2;}
    else if (OMF_KP == reg_NL){delta_Kp = 3;}
    if (OMF_KI == reg_NL){
        delta_Ki = -3;}
    else if (OMF_KI == reg_NM){delta_Ki = -2;}
    else if (OMF_KI == reg_NS){delta_Ki = -1;}
    else if (OMF_KI == reg_Z){delta_Ki = 0;}
    else if (OMF_KI == reg_PS){delta_Ki = 1;}
    else if (OMF_KI == reg_PM){delta_Ki = 2;}
    else if (OMF_KI == reg_NL){delta_Ki = 3;}

    X[0] = error_out;
    X[1] = delta_Kp;
    X[2] = delta_Ki;
    X[3] = theta_m;
    X[4] = change_error_out;

}


static void mdlTerminate(SimStruct *S) { } /* Keep this function empty
since no memory is allocated */
```

```
#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h" /*Code generation registration function*/
#endif
```

Spd_Ctrl.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME Spd_Ctrl
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element]) /*Pointer to Input Port0*/

static void mdlInitializeSizes(SimStruct *S){
    ssSetNumDiscStates(S, 3);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 6);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);

    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE
    | SS_OPTION_DISCRETE_VALUED_OUTPUT));
}

static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, 1e-3);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S){
    real_T *X0 = ssGetRealDiscStates(S);
    int_T nXStates = ssGetNumDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    int_T i;

    /* initialize the states to 0.0 */
    for (i=0; i < nXStates; i++) {
        X0[i] = 0.0;
    }
}

static void mdlOutputs(SimStruct *S, int_T tid){
    real_T *Y = ssGetOutputPortRealSignal(S,0);
    real_T *X = ssGetRealDiscStates(S);
    real_T ref;
    ref = X[2];
    Y[0] = ref;
}

#define MDL_UPDATE
```

```
static void mdlUpdate(SimStruct *S, int_T tid) {
    real_T *X = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T dt = 1e-3;
    real_T Kp, Ki,delta_Kp, delta_Ki, W_ref, W_act,
error,theta_m,theta_old;
    real_T integral, integral_old, ip;
    W_ref = U(0);
    Ki = U(1);
    Kp = U(2);
    theta_m = U(3);
    delta_Kp = U(4);
    delta_Ki = U(5);

    Kp = Kp + delta_Kp;
    Ki = Ki + delta_Ki;

//  PI Controller
    integral_old = X[0];
    theta_old = X[1];
    W_act = (theta_m - theta_old)/dt;
    error = W_ref - W_act;
    integral = integral_old + error*dt;
    ip = Ki*integral - Kp*W_act;

    X[0] = integral;
    X[1] = theta_m;
    X[2] = ip;
}

static void mdlTerminate(SimStruct *S)
{ } /*Keep this function empty since no memory is allocated*/

#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h" /*Code generation registration function*/
#endif
```

RFOC.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME RFOC
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element]) /*Pointer to Input Port0*/

static void mdlInitializeSizes(SimStruct *S){
    ssSetNumDiscStates(S, 8);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 5);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 3);
    ssSetNumSampleTimes(S, 1);
```

```
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE
    | SS_OPTION_DISCRETE_VALUED_OUTPUT));
}

static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, 1e-3);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S){
real_T *X0 = ssGetRealDiscStates(S);
int_T nXStates = ssGetNumDiscStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
int_T i;

/* initialize the states to 0.0 */
for (i=0; i < nXStates; i++) {
X0[i] = 0.0; } }

static void mdlOutputs(SimStruct *S, int_T tid){
    real_T *Y = ssGetOutputPortRealSignal(S,0);
    real_T *X = ssGetRealDiscStates(S);
    real_T Va,Vb,Vc;
    Va = X[5];
    Vb = X[6];
    Vc = X[7];
    Y[0] = Va;
    Y[1] = Vb;
    Y[2] = Vc;
}

#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid) {
    real_T *X = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T dt = 1e-3;
    real_T iq,id,theta_e,ia,ib,ic,isalfa,isbeta,isd,isq,we;
    real_T Vsd,Vsq,V_Alfa,V_Beta,Va,Vb,Vc;
    real_T integral_old_q, integral_q,error_q,pi_q;
    real_T integral_old_d, integral_d,error_d,pi_d;
    real_T theta_e_old,wm,iq_old,id_old,iq_new,id_new;
    real_T Kpd,Ki,Kpq;

    iq = U(0);
    theta_e = U(1);
    ia = U(2);
    ib = U(3);
    ic = U(4);
    id = 0.00;

    // Konstanta Transformasi
    real_T Constant = 0.816497;
    real_T Constant2 = 0.866025;
    // Parameter Motor PMSM
    real_T N = 4; //4;
    real_T psi = 0.121;
    real_T Lsd = 16.61e-3;
    real_T Lsq = 16.22e-3;
    real_T Rs = 0.55;
```

```
    real_T Td = 0.01;

    // ABC -> ALFA BETA
    isalfa = Constant*(ia - 0.5*ib - 0.5*ic);
    isbeta = Constant*Constant2*(ib - ic);
    // ALFA BETA -> d q
    isd = isalfa*cos(theta_e) + isbeta*sin(theta_e);
    isq = -isalfa*sin(theta_e) + isbeta*cos(theta_e);

    Kpd = (Lsd/Td);
    Kpq = (Lsq/Td);
    Ki = (Rs/Td);
    // PI Iq
    integral_old_q = X[0];
    error_q = iq - isq;
    integral_q = integral_old_q + error_q*dt;
    pi_q = Kpq*error_q + Ki*integral_q;

    //PI Id
    integral_old_d = X[1];
    error_d = id - isd;
    integral_d = integral_old_d + error_d*dt;
    pi_d = Kpd*error_d + Ki*integral_d;

    //Output Vsd dan Vsq
    theta_e_old = X[2];
    iq_old = X[3];
    id_old = X[4];
    wm = (theta_e - theta_e_old)/(dt*N);
    iq_new = iq_old + (iq-iq_old)*dt/Td;
    id_new = id_old + (id-id_old)*dt/Td;
    Vsd = pi_d - N*wm*Lsq*iq_new;
    Vsq = pi_q + N*wm*(Lsd*id_new + psi);

    //d q -> ALFA BETA
    V_Alfa = Vsd*cos(theta_e) - Vsq*sin(theta_e);
    V_Beta = Vsd*sin(theta_e) + Vsq*cos(theta_e);

    //ALFA BETA -> ABC
    Va = Constant*V_Alfa;
    Vb = Constant*(-0.5*V_Alfa + (Constant2)*V_Beta);
    Vc = Constant*(-0.5*V_Alfa - (Constant2)*V_Beta);

    X[0] = integral_q;
    X[1] = integral_d;
    X[2] = theta_e;
    X[3] = iq_new;
    X[4] = id_new;
    X[5] = Va;
    X[6] = Vb;
    X[7] = Vc;
}

static void mdlTerminate(SimStruct *S)
{ } /*Keep this function empty since no memory is allocated*/

#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h" /*Code generation registration function*/
```

```
#endif
```

SPWM.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME SPWM
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element]) /*Pointer to Input Port0*/


static void mdlInitializeSizes(SimStruct *S){
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 5);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 3);
    ssSetNumSampleTimes(S, 1);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

static void mdlInitializeSampleTimes(SimStruct *S) {
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

static void mdlOutputs(SimStruct *S, int_T tid) {
    real_T *Y = ssGetOutputPortRealSignal(S,0);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T t = ssGetT(S);

    real_T halfVDC = (0.5*U(0));
    int_T i;

    for(i=0; i < 3;i++){
        if(fabs(U(i+2))/halfVDC >= fabs(U(0)*(fabs(fmod(t,2/U(1))-
1/U(1))*U(1)))){
            Y[i] = halfVDC;
        }else{
            Y[i] = 0.0;
        };
        if(U(i+2) < 0.0){
            Y[i] = -Y[i];
        };
    }
}


static void mdlTerminate(SimStruct *S)
{ } /*Keep this function empty since no memory is allocated*/

#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /*Code generation registration function*/
#endif
```

PMSM_Model.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME PMSM_Model
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element])

static void mdlInitializeSizes(SimStruct *S){
    ssSetNumContStates(S, 4);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 4);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 7);
    ssSetNumSampleTimes(S, 1);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

static void mdlInitializeSampleTimes(SimStruct *S) {
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S) {

    real_T *X0 = ssGetContStates(S);
    int_T nStates = ssGetNumContStates(S);
    int_T i;

    /* initialize the states to 0.0 */
    for (i=0; i < nStates; i++) {X0[i] = 0.0;}
}

static void mdlOutputs(SimStruct *S, int_T tid) {
    real_T *Y = ssGetOutputPortRealSignal(S,0);
    real_T *X = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T Wm, Theta_e, Isd, Isq, Te, Ia, Ib, Ic, Vsd, Vsq;
    real_T I_Alfa, I_Beta;
//      Konstanta Transformasi
    real_T Constant = 0.816497;
    real_T Constant2 = 0.866025;
//      Parameter Motor PMSM
    real_T N = 4;

    Wm = X[0];
    Theta_e = X[1];
    Isq = X[2];
    Isd = X[3];
    Te = X[4];
```

```
    //d q -> ALFA BETA
    I_Alfa = Isd*cos(Theta_e) - Isq*sin(Theta_e);
    I_Beta = Isd*sin(Theta_e) + Isq*cos(Theta_e);
    //ALFA BETA -> ABC
    Ia = Constant*I_Alfa;
    Ib = Constant*(-0.5*I_Alfa + (Constant2)*I_Beta);
    Ic = Constant*(-0.5*I_Alfa - (Constant2)*I_Beta);

    Y[0] = Ia;
    Y[1] = Ib;
    Y[2] = Ic;
    Y[3] = Wm;
    Y[4] = Te;
    Y[5] = Theta_e/N;
    Y[6] = Theta_e;
}

#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S) {
    real_T *dX = ssGetdX(S);
    real_T *X = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    real_T Va, Vb, Vc, Vsalfa, Vsbeta, Vsd, Vsq;
    real_T Te, Tl, Wm, Wm_dot, Isd_dot, Isq_dot;
    real_T Theta_e, Theta_e_dot, Isd, Isq;
    real_T Ia, Ib, Ic, I_Alfa, I_Beta;
//      Konstanta Transformasi
    real_T Constant  = 0.816497;
    real_T Constant2 = 0.866025;
//      Parameter Motor PMSM
    real_T N = 4; //4;
    real_T psi = 0.121;
     real_T Lsd = 16.61e-3;
     real_T Lsq = 16.22e-3;
     real_T Rs = 0.55;
     real_T J = 0.01;

     //input
     Tl = U(0);
     Va = U(1);
     Vb = U(2);
     Vc = U(3);

     Wm = X[0];
     Theta_e = X[1];
     Isd = X[2];
     Isq = X[3];

//      ABC -> ALFA BETA
     Vsalfa = Constant*(Va - 0.5*Vb - 0.5*Vc);
     Vsbeta = Constant*Constant2*(Vb - Vc);
```

```
//      ALFA BETA -> d q
    Vsd = Vsalfa*cos(Theta_e) + Vsbeta*sin(Theta_e);
    Vsq = -Vsalfa*sin(Theta_e) + Vsbeta*cos(Theta_e);

//      PMSM DQ MODEL
    Te = N*(psi + (Lsd - Lsq)*Isd)*Isq;
    Wm_dot = (Te - Tl) / J;
    Theta_e_dot = N*Wm;
    Isd_dot = (Vsd - Rs*Isd + N*Wm*Lsq*Isq)/Lsd;
    Isq_dot = (Vsq - Rs*Isq - N*Wm*(Lsd*Isd+psi))/Lsq;

//      state
    dX[0] = Wm_dot;
    dX[1] = Theta_e_dot;
    dX[2] = Isd_dot;
    dX[3] = Isq_dot;

    X[4] = Te;
}

static void mdlTerminate(SimStruct *S)
{} /*Keep this function empty since no memory is
allocated*/

#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /*Code generation registration
function*/
#endif
```