



UNIVERSITAS INDONESIA

**REVIEW AND INVESTIGATION OF SIMPLIFIED RULES
FUZZY LOGIC SPEED CONTROLLER OF HIGH
PERFORMANCE INDUCTION MOTOR DRIVES**

**UJIAN AKHIR SEMESTER
SISTEM KENDALI MOTOR LISTRIK**

Hansel Matthew

1806194914

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO**

**MADIUN
JUNI 2021**

ABSTRAK

Nama : Hansel Matthew

Program Studi : Teknik Elektro

Judul : Review and Investigation of Simplified Rules Fuzzy Logic Speed Controller of High Performance Induction Motor Drives

Penggunaan Fuzzy Logic Controller sebagai pengendali kecepatan untuk motor induksi telah banyak dikembangkan oleh banyak peneliti karena telah terbukti dapat memberikan hasil yang lebih baik dibandingkan pengendali konvensional. Berdasarkan literatur yang ada, metode yang paling umum untuk mendesain rule base dari Membership Function (MF) untuk Fuzzy Logic Controller adalah menggunakan pengetahuan dari ketrampilan keteknikan dan juga pengalaman dalam pengoperasian plant tersebut. Permasalahan kedua pada FLC adalah beban komputasi yang besar. Untuk mengatasi masalah ini maka beberapa peneliti membuat peraturan Fuzzy sederhana dengan mengurangi jumlah peraturan fuzzy yang digunakan agar mempermudah dalam implementasi. Sebagian besar dari metode yang diusulkan memiliki kelemahannya masing – masing dan juga masih mengandalkan ketrampilan pendesain dalam melakukan desain tersebut. Studi ini dilakukan untuk meneliti apakah ada acara sistematis untuk mendesain suatu aturan fuzzy dan apakah aturan tersebut bisa disimplifikasi agar mempermudah implementasi pada motor induksi. Sehingga dibutuhkan suatu metode untuk mendesain dan menyederhanakan peraturan dari Fuzzy Logic Controller untuk motor induksi. Metode yang diusulkan adalah dengan menyederhanakan peraturan fuzzy berukuran 9,25 dan 49 menjadi 5,7 dan 9. Tingkat efektivitas dan akurasi dari peraturan tersebut juga diverifikasi dengan melakukan simulasi menggunakan MATLAB/Simulink.

Kata kunci : FLC, IM, Fuzzy Logic

DAFTAR ISI

ABSTRAK	i
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Tujuan Penulisan	2
1.3. Rumusan Masalah.....	2
1.4. Sistematika Penulisan	3
BAB 2 LANDASAN TEORI	4
2.1. Motor Induksi.....	4
2.2. Transformasi Clarke dan Park.....	7
2.3. Vector Control.....	8
2.4. Fuzzy Logic Controller	9
2.4.1. Fuzzification (Pre Processing).....	10
2.4.2. Inference Engine – Rule Base (Processing).....	11
2.4.3. Defuzzification (Post Processing).....	12
2.5. Sinusoidal Pulse Width Modulation (SPWM).....	13
BAB 3 SIMULASI DAN ANALISIS	15
3.1. Parameter IM pada Simulasi	15
3.2. Pemodelan Fuzzy Logic Controller pada Simulasi	15
3.2.1. Fungsi Keanggotaan FLC	16
3.2.2. Rule Base FLC.....	17
3.3. Simulasi Pengendali	20
3.3.1. Simulasi Rule Base (3x3) dan Simplified 5	21
3.3.2. Simulasi Rule Base (5x5) dan Simplified 7	29
3.3.3. Simulasi Rule Base (7x7) dan Simplified 9	37
BAB 4 KESIMPULAN	46
DAFTAR PUSTAKA	47
LAMPIRAN KODE SUMBER	49
FUZZY9.c	49
FUZZY25.c	57
FUZZY49.c	66
FUZZY5SIM.c.....	78
FUZZY7SIM.c.....	86
FUZZY9SIM.c.....	94
RFOC_IM.c	103
PWM.c.....	106
IM.c	107

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Motor induksi membutuhkan respon yang cepat, variasi parameter yang robust, kemampuan untuk tahan akan gangguan dan implementasi yang mudah, Field Oriented Control (FOC), Direct Torque Control (DTC), Model Predictive Control (MPC) adalah metode kendali yang paling umum digunakan untuk mengendalikan motor induksi.

Efektivitas kecepatan sangat penting untuk keandalan motor induksi. Umumnya, pengendali Proportional Integral (PI) diimplementasikan sebagai pengontrol kecepatan pada motor AC, yang dimana memperoleh respons transien yang cepat dan respons kondisi tunak yang baik. Namun, pengendali PI biasanya sensitif terhadap variasi parameter motor, sistem non linear, gangguan beban, dan variasi kecepatan, yang akibatnya menurunkan performa penggerak. Oleh karena itu, Fuzzy Logic Controller (FLC) diusulkan sebagai pengontrol adaptif untuk menggantikan pengontrol PI karena kurang sensitif terhadap variasi parameter, penanganan non-linier, gangguan beban kemampuan penolakan, dan ketahanan terhadap variasi kecepatan. Fitur-fitur FLC ini menjadikannya pilihan utama untuk pengendalian motor induksi.

Selama beberapa waktu belakangan, FLC telah menjadi pengendali yang dominan untuk pengendalian motor induksi karena memiliki respon dinamis yang cepat serta respon steady state yang lebih baik. Pada umumnya terdapat 3 jenis peraturan MF FLC yaitu 7x7, 5x5 dan 3x3. Jumlah MF dan ukuran peraturan FLC berpengaruh terhadap peforma pengendali dimana performa pengendalian akan membaik seiring bertambahnya jumlah peraturan yang digunakan. Jumlah peraturan yang besar mungkin dapat membantu memperbagus peforma dari motor AC terutama dalam lingkungan simulasi, namun memiliki kelemahan yaitu beban komputasi yang sangat besar pada saat implementasi ke perangkat keras. Hal ini menyebabkan ketika saat implementasi, FLC dengan peraturan yang lebih banyak

memiliki peforma yang lebih buruk dibandingkan FLC dengan peraturan yang lebih sedikit karena beban komputasional dari FLC tersebut.

Untuk mengatasi permasalahan tersebut, maka beberapa peneliti mengembangkan beberapa metode lain yang dapat mengurangi beban komputasi namun tetap mempertahankan peforma yang ada. Simplifikasi FLC merupakan salah satu metoded tersebut. Salah satu cara adalah model FLC baru yang didesain sebagai gabungan dari membership function trapezoidal dan triangular. Cara kedua adalah dengan memilih rule yang dominan pada rule base kemudian mengeluarkan rule yang tidak terlalu dominan sehingga menghasilkan rule base yang lebih sedikit. Penelitian ini memiliki tujuan untuk melihat dan membandingkan peforma dari FLC sebelum dan setelah dilakukan simplifikasi.

Semua simulasi tersebut akan dilakukan dengan menggunakan Simulink pada software MATLAB dan diharapkan dapat memberikan hasil yang sesuai dengan yang diharapkan.

1.2. Tujuan Penulisan

Tujuan dari laporan ini adalah untuk membandingkan peforma yang dihasilkan oleh beberapa ragam rule base Fuzzy Logic Controller baik yang sesudah maupun setelah dilakukan simplifikasi dengan melakukan pemodelan dan simulasi pengendalian motor induksi dengan metode Field Oriented Control.

1.3. Rumusan Masalah

Rumusan masalah pada laporan ini meliputi:

- Bagaimana cara mengendalikan Motor Induksi dengan menggunakan Fuzzy Logic Controller?
- Bagaimana cara membandingkan Motor Induksi dengan menggunakan rule base yang sebelum dan setelah disimplifikasi?
- Bagaimana performa yang dihasilkan dari Fuzzy Logic Controller yang didesain?

1.4. Sistematika Penulisan

1. Bab I: Pendahuluan

Bab ini berisi mengenai latar belakang, tujuan penulisan, rumusan masalah, dan sistematika penulisan.

2. Bab II: Landasan Teori

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

3. Bab III: Simulasi dan Analisis

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

4. Bab IV: Kesimpulan

Bab ini berisi mengenai kesimpulan dari penelitian yang telah dibuat.

BAB 2

LANDASAN TEORI

2.1. Motor Induksi

Motor Induksi (IM) adalah motor AC yang memiliki berbagai aplikasi industri dan konsumen karena konstruksinya yang kokoh, perawatan yang lebih sedikit, dan keandalan. Karena penggunaannya yang intensif dalam aplikasi daya tinggi, IM memerlukan sistem penggerak kinerja tinggi untuk mengontrol operasinya secara efisien dan tepat. Dua metode penggerak IM berkinerja tinggi yang populer adalah Field Oriented Control (FOC) dan Direct Torque Control (DTC) yang keduanya bekerja berdasarkan pemodelan matematika IM untuk menggerakkan kecepatan dan/atau torsinya. FOC bekerja dengan menguraikan torsi dan fluks ke dalam bingkai DQ dan dengan bantuan transformasi fase dan kontrol histeresis atau kontrol vektor ruang, dapat menghasilkan pulsa switching untuk inverter, DTC bekerja dengan menggunakan dua pengontrol fluks histeresis dan torsi untuk memilih yang paling tepat vektor tegangan berdasarkan tabel switching yang telah ditentukan sesuai dengan posisi fluks dan torsi dan sinyal kesalahan fluks. Gambar berikut menunjukkan diagram blok sistem penggerak IM yang terdiri dari model IM, pengontrol kecepatan, metode penggerak FOC atau DTC, dan Inverter Sumber Tegangan (VSI).

Sistem penggerak Motor Induksi (IM) dapat dimodelkan secara matematis dalam kerangka acuan yang berbeda seperti kerangka acuan stasioner di mana sumbu DQ tidak berputar, kerangka acuan sinkron di mana sumbu DQ berputar pada kecepatan sinkron atau kerangka acuan putar di mana sumbu DQ berputar pada kecepatan rotor. Persamaan tegangan IM yang dinyatakan dalam kerangka acuan stasioner dapat ditulis sebagai berikut:

$$V_{sd} = R_s I_{sd} + \frac{d\varphi_{sd}}{dt} \quad (1)$$

$$V_{sq} = R_s I_{sq} + \frac{d\varphi_{sq}}{dt} \quad (2)$$

$$V_{rd} = R_r I_{rd} + \frac{d\varphi_{rd}}{dt} - \omega_r \quad (3)$$

$$V_{rq} = R_r I_{rq} + \frac{d\varphi_{rq}}{dt} + \omega_r \varphi_{rd} \quad (4)$$

Dan persamaan fluks pada motor induksi dapat diekspresikan sebagai berikut:

$$\varphi_{sd} = L_s I_{sd} + L_m I_{rd} \quad (5)$$

$$\varphi_{sq} = L_s I_{sq} + L_m I_{rq} \quad (6)$$

$$\varphi_{rd} = L_m I_{sd} + L_r I_{rd} \quad (7)$$

$$\varphi_{rq} = L_m I_{sq} + L_r I_{rq} \quad (8)$$

di mana V_{sd} , V_{sq} adalah tegangan yang diberikan ke stator; dan I_s ; I_{sq} ; I_r ; I_{rq} adalah arus stator dan arus rotor sumbu d dan q yang sesuai. ' $'sd$ '; ' $'sq$ ', ' $'rd$ ', ' $'rq$ adalah komponen ux stator dan rotor. R_s , R_r adalah resistansi stator dan rotor: L_s , L_r masing-masing menunjukkan induktansi stator dan rotor, sedangkan L_m adalah induktansi timbal balik.

Persamaan vektor ruang dari motor induksi dalam kerangka acuan stasioner juga dapat ditulis dalam bentuk matriks dalam hal komponen d-q sebagai berikut:

$$\begin{bmatrix} V_{qs} \\ V_{ds} \\ V_{qr} \\ V_{dr} \end{bmatrix} = \begin{bmatrix} R_s + sL_s & 0 & sL_m & 0 \\ 0 & R_s + sL_s & 0 & sL_m \\ sL_m & \omega_r L_m & R_r + sL_r & \omega_r L_r \\ -\omega_r L_m & sL_m & -\omega_r L_r & R_r + sL_r \end{bmatrix} \times \begin{bmatrix} i_{qs} \\ i_{ds} \\ i_{qr} \\ i_{dr} \end{bmatrix} \quad (9)$$

dimana term S pada persamaan tersebut melambangkan operator laplace yang merepresentasikan operator turunan d/dt . Persamaan tersebut juga dapat diturunkan

menjadi bentuk ruang keadaan dengan arus sebagai variabel keadaan. Penurunan persamaan tersebut dapat ditulis sebagai berikut:

$$\begin{aligned}
 & \begin{bmatrix} \dot{i}_{sd} \\ \dot{i}_{sq} \\ \dot{i}_{rd} \\ \dot{i}_{rq} \end{bmatrix} \\
 &= \frac{1}{L_m^2 - L_r L_s} \\
 &\times \begin{bmatrix} R_s L_r & -\omega_r L_m^2 i_{sq} & -R_r L_m & -\omega_r L_m L_r \\ \omega_r L_m^2 & R_s L_r & \omega_r L_m L_r & -R_r L_m \\ -R_s L_m & \omega_r L_m L_s & R_r L_s & \omega_r L_r L_s \\ -\omega_r L_m L_s & -R_s L_m & -\omega_r L_r L_s & R_r L_s \end{bmatrix} \\
 &\times \frac{1}{L_m^2 - L_r L_s} \times \begin{bmatrix} -L_r & 0 \\ 0 & L_r \\ L_m & 0 \\ 0 & L_m \end{bmatrix} \times \begin{bmatrix} v_{sd} \\ v_{sq} \\ v_{rd} \\ v_{rq} \end{bmatrix} \quad (10)
 \end{aligned}$$

Persamaan torsi pada motor induksi juga dapat ditulis dalam bentuk mekanis menjadi:

$$T_e = J \frac{d\omega_m}{dt} + B\omega_m + T_L = \frac{J}{P} \frac{d\omega_r}{dt} + \frac{B}{P}\omega_r + T_L \quad (11)$$

dimana, J adalah momen inersia total, B adalah gesekan viskos, TL adalah torsi beban. wr adalah kecepatan sudut listrik rotor dalam rad./s, wm adalah kecepatan motor dalam rad/s.

Persamaan torsi juga dapat ditulis dalam bentuk elektris menjadi:

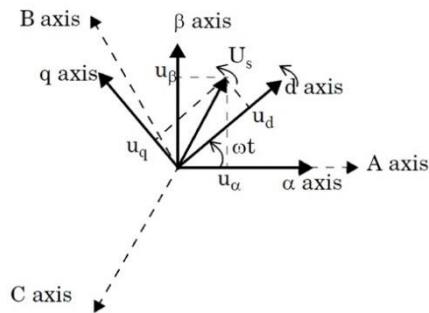
$$T_e = \frac{3}{2} P (\bar{\varphi}_s x \bar{i}_s) = \frac{3}{2} P (\varphi_{sd} i_{sq} - \varphi_{sq} i_{sd}) \quad (12)$$

$$T_e = \frac{3}{2} P L_m (i_{sq} i_{rd} - i_{sd} i_{rq}) \quad (13)$$

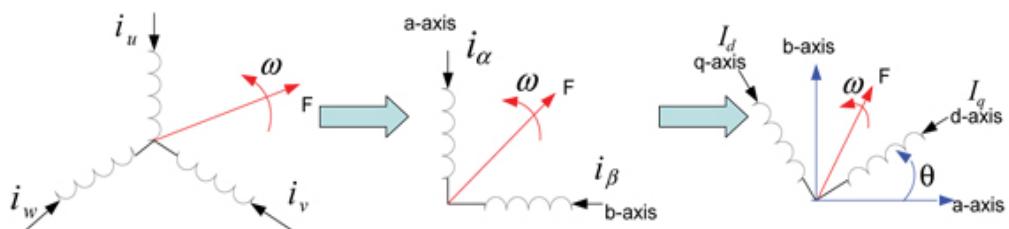
dimana P merupakan pasangan pole pada motor induksi.

2.2. Transformasi Clarke dan Park

Pada motor induksi dibutuhkan pengendalian dari torsi dan fluks. Pengendalian motor induksi membutuhkan transformasi untuk mengubah bentuk 3 fasa yaitu abc ke dalam 2 fasa yang terdiri dari frame stasioner $\alpha\beta$ dan frame dinamik dq. Ketiga bentuk fasa tersebut dapat dilihat pada gambar berikut.



Motor induksi akan mengeluarkan arus dalam bentuk 3 fasa abc yang kemudian akan ditransformasikan kedalam bentuk 2 fasa $\alpha\beta$. Namun transformasi ini masih sulit untuk dikendalikan karena komponen tersebut masih dalam kondisi berputar sehingga masih terkait dengan rumusan trigonometri. Sehingga bentuk $\alpha\beta$ ditransformasikan lagi ke dalam bentuk dq yaitu bentuk yang sudah sinkron yang sudah dapat merepresentasikan perputaran medan magnet di dalam motor. Dengan kedua transformasi ini maka komponen yang ingin dikendalikan menjadi seolah – olah diam sehingga dapat dikendalikan dengan mudah. Berikut merupakan ilustrasi tahapan transformasi tersebut.



Transformasi dari bentuk 3 fasa menjadi $\alpha\beta$ disebut sebagai transformasi Clarke dan mempunyai rumusan sebagai berikut:

$$[\mathbf{c}] = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} i_{\alpha} \\ i_{\beta} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}$$

Transformasi dari bentuk $\alpha\beta$ menjadi dq disebut sebagai transformasi park dan mempunyai rumusan sebagai berikut:

$$[\mathbf{c}] = \begin{bmatrix} \cos \theta_e & \sin \theta_e \\ -\sin \theta_e & \cos \theta_e \end{bmatrix} \begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} i_{\alpha} \\ i_{\beta} \end{bmatrix}$$

2.3. Vector Control

Vector Control atau disebut juga Field Oriented (FOC), adalah metode kendali penggerak frekuensi variabel di mana arus stator dari motor induksi diidentifikasi sebagai dua komponen ortogonal yang dapat divisualisasikan dengan sebuah vektor. Satu komponen mendefinisikan fluks magnet motor, yang lain torsi. Sistem kendali akan menghitung referensi komponen arus yang sesuai dari referensi fluks dan torsi yang diberikan oleh kontrol kecepatan drive. Field Oriented Control menggunakan pengendali proporsional-integral (PI) untuk menjaga komponen arus terukur agar mencapai nilai referensinya. Modulasi lebar pulsa dari penggerak frekuensi variabel akan mengatur peralihan transistor sesuai dengan referensi tegangan stator yang merupakan keluaran dari pengontrol arus PI. Vector Control ini biasa digunakan pada motor yang memiliki performa yang tinggi di mana untuk mampu bekerja pada kecepatan penuh dengan baik, menghasilkan torsi yang maksimal pada kecepatan nol, dan mampu menghasilkan percepatan maupun perlambatan yang cepat dan tepat.

Terdapat beberapa metode FOC namun pada laporan ini metode yang digunakan adalah Rotor Field Oriented Control dimana metode tersebut pada implementasi motor induksi mempunyai persamaan pengendali arus sebagai berikut.

$$u_{ds} = K_{idp} (i_{ds}^{e*} - i_{ds}^e) + K_{idi} \int (i_{ds}^{e*} - i_{ds}^e) dt$$

$$u_{qs} = K_{iqp} (i_{qs}^{e*} - i_{qs}^e) + K_{iqi} \int (i_{qs}^{e*} - i_{qs}^e) dt$$

Kemudian dibuat juga model fluks yang akan menghasilkan magnitude dan sudut dari posisi rotor dan juga torsi. Model tersebut diekspresikan dengan rumus sebagai berikut.

$$\frac{d}{dt} i_{mr} = \frac{R_r}{L_r} (i_{ds}^e - i_{mr})$$

$$\omega_e = N_p \omega_r + \frac{R_r}{L_r} \frac{i_{qs}^{e*}}{i_{mr}}$$

$$\frac{d}{dt} \theta_e = \omega_e$$

$$T_e = N_p (1 - \sigma) L_s i_{mr} i_{qs}^e$$

Terdapat juga voltase decoupling dan voltase reference yang akan dikeluarkan oleh RFOC yang diekspresikan dengan rumus sebagai berikut.

$$v_{cd} = -\omega_e L_s \sigma i_{qs}^e + L_s (1 - \sigma) \frac{d}{dt} i_{mr}$$

$$v_{cq} = \omega_e L_s \sigma i_{ds}^e + L_s (1 - \sigma) \omega_e i_{mr}$$

$$v_{ds}^{e*} = u_{ds} + v_{cd}$$

$$v_{qs}^{e*} = u_{qs} + v_{cq}$$

2.4. Fuzzy Logic Controller

Logika fuzzy adalah model matematika yang didasarkan pada kombinasi logika, kecerdasan buatan dan algoritma probabilistik yang memiliki fungsi untuk memecahkan masalah yang ada seperti pemikiran manusia. Logika fuzzy juga dapat

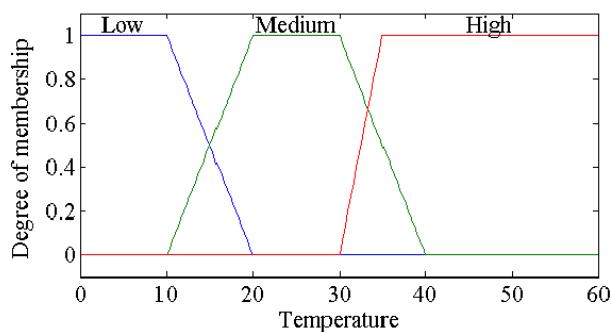
memecahkan masalah yang kompleks atau rumit, seperti persamaan matematika nonlinier. Pada dasarnya logika fuzzy akan mampu menghasilkan tidak hanya nilai keluaran yang berupa nilai pembulatan (0 dan 1), tetapi juga nilai keluaran pada rentang 0-1. Dengan cara ini, logika fuzzy dapat menentukan keputusan yang termasuk dalam rentang nilai, seperti kemampuan berpikir manusia. Terdapat beberapa tahapan dari Fuzzy Logic Controller. Tahapan tersebut dapat dilihat pada gambar berikut.

2.4.1. Fuzzification (Pre Processing)

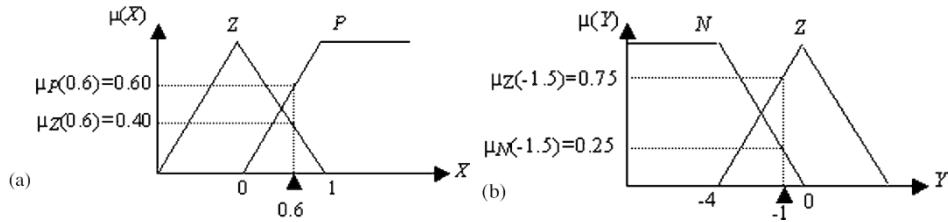
Merupakan proses pengubahan nilai input linguistic menjadi nilai fuzzy. Proses ini dilakukan dengan cara mengubah nilai input crisp menjadi dalam bentuk Degree of Membership (DOM) dan Membership Function (MF).

Derajat keanggotaan atau Degree of Membership adalah suatu derajat atau tingkatan yang merepresentasikan banyaknya nilai yang dihasilkan ketika suatu nilai yang jelas dimasukkan ke dalam suatu nilai fuzzy (nilai linguistik). DOM ini akan menunjukkan seberapa jelas nilai fungsi keanggotaan fuzzy. Biasanya nilai DOM ini berkisar antara nilai 0 sampai 1.

Fungsi keanggotaan atau membership function adalah fungsi yang digunakan sebagai acuan untuk mengklasifikasikan nilai-nilai input crisp ke dalam kelompok anggotanya. Berikut merupakan contoh gambar dari hubungan antara Derajat Keanggotaan dan Fungsi Keanggotaan



Nilai input akan dikelompokkan pada input fungsi keanggotaan yang ada, dimana nilai input akan berada dalam dua variabel linguistik dengan derajat keanggotaannya masing-masing. Berikut merupakan gambar hubungan antara input crisp dengan kedua kemungkinan derajat keanggotan dan fungsi keanggotanya masing – masing.



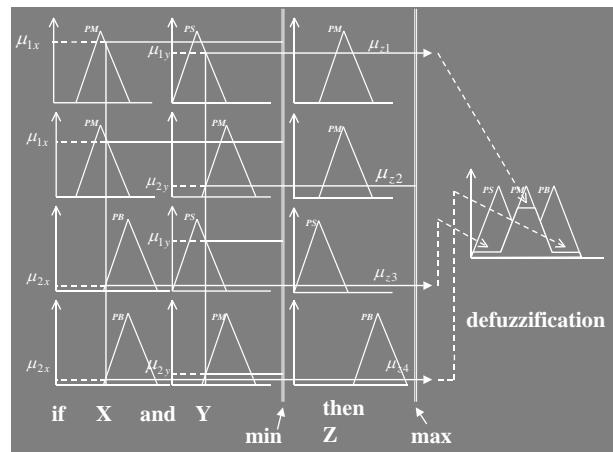
2.4.2. Inference Engine – Rule Base (Processing)

Rule Base merupakan suatu aturan yang disajikan dalam bentuk (IF-THEN) untuk menggambarkan hubungan antara variabel input dan output dalam bentuk linguistik. Berikut merupakan salah satu contoh dari rule base fuzzy logic.

$E@$	NL	NM	NS	Z	PS	PM	PL
NL	NL	NL	NL	NL	NM	NS	Z
NM	NL	NL	NL	NM	NS	Z	PS
NS	NL	NL	NM	NS	Z	PS	PM
Z	NL	NM	NS	Z	PS	PM	PL
PS	NM	NS	Z	PS	PM	PL	PL
PM	NS	Z	PS	PM	PL	PL	PL
PL	Z	PS	PM	PL	PL	PL	PL

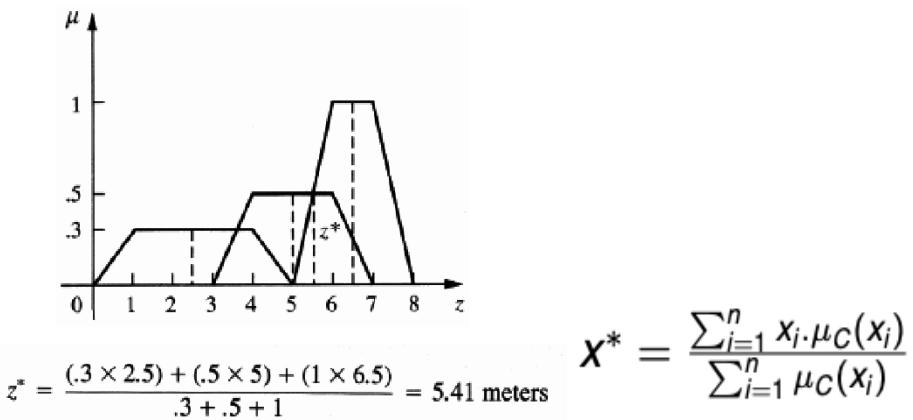
Inference engine adalah sistem yang menentukan nilai keluaran berupa proses penalaran nilai masukan, biasanya dihubungkan dengan metode AND/OR. Inference Engine dibutuhkan untuk menentukan nilai fuzzy mana yang akan digunakan. Terdapat beberapa metode Inference engine namun pada laporan ini menggunakan metode Min-Max Algorithm.

Algoritma Min-Max merupakan algoritma untuk menentukan nilai fuzzy mana yang akan dipilih dari beberapa kemungkinan yang ada. Algoritma ini mula – mula akan membandingkan antara dua kemungkinan fuzzy, maka akan dipilih nilai fuzzy yang memiliki nilai DOM yang paling kecil atau minimum. Setelah itu nilai tersebut akan dibandingkan terhadap hasil perbandingan minimum lainnya, jika terdapat dua atau lebih nilai fuzzy yang memiliki MF yang sama, maka akan dipilih nilai fuzzy yang memiliki DOM yang paling besar atau maximum. Ilustrasi proses algoritma tersebut dapat dilihat pada gambar berikut.



2.4.3. Defuzzification (Post Processing)

Defuzzification adalah tahapan dimana nilai fuzzy yang sudah didapatkan dari hasil inference engine – rule base menjadi luaran crisp (crisp output). Terdapat beberapa metode defuzzification namun pada laporan ini menggunakan metode Weighted Average Method yaitu dengan menjumlahkan hasil perkalian dari nilai DOM dengan nilai MF nya masing – masing kemudian membagi hasil tersebut dengan penjumlahan dari keseluruhan DOM. Metode tersebut diekspresikan dengan rumus serta diilustrasikan sebagai berikut :



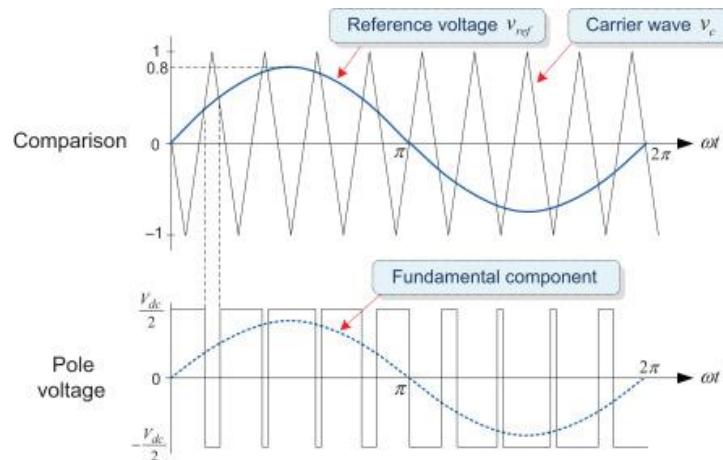
2.5. Sinusoidal Pulse Width Modulation (SPWM)

PWM sinusoidal adalah jenis modulasi lebar pulsa yang berbasis pada sebuah sinyal carrier berupa sinyal sinusoidal. PWM berbasis operator menggunakan sinyal modulasi yang telah ditentukan sebelumnya untuk menentukan tegangan output. Dalam PWM sinusoidal, sinyal modulasi adalah sinusoidal, dengan puncak sinyal modulasi selalu lebih kecil dari puncak sinyal pembawa. Kaki inverter PWM sinusoidal dan tegangan saluran diilustrasikan di bawah ini.

Dalam teknik PWM ini, referensi tegangan AC sinusoidal dibandingkan dengan gelombang pembawa segitiga frekuensi tinggi secara real time untuk menentukan status switching untuk setiap kutub dalam inverter. Setelah membandingkan, status switching untuk setiap kutub dapat ditentukan berdasarkan aturan berikut:

- Referensi tegangan (V_{ref}) > Pembawa segitiga : Tegangan Output = $V_{dc}/2$
(Sakelar atas dihidupkan)
- Referensi tegangan (V_{ref}) < Pembawa segitiga : Tegangan Output = $-V_{dc}/2$
(Sakelar bawah dihidupkan)

Pada metode ini, nilai puncak-ke-puncak dari gelombang pembawa segitiga diberikan sebagai tegangan DC input. Dalam teknik PWM ini, kondisi yang diperlukan untuk modulasi linier adalah bahwa amplitudo tegangan referensi(V_{ref}) harus tetap di bawah puncak pembawa segitiga (V_c). Karena teknik PWM ini menggunakan gelombang pembawa frekuensi tinggi untuk modulasi tegangan, teknik PWM semacam ini disebut teknik PWM berbasis pembawa (Carried Based PWM). Teknik berbasis pembawa ini disebut juga sebagai SPWM karena referensi diberikan sebagai bentuk gelombang sinus. Penamaan lain juga adalah sebagai teknik PWM perbandingan segitiga karena ini menggunakan gelombang pembawa segitiga. Ilustrasi dari SPWM ditunjukkan oleh gambar berikut.



BAB 3

SIMULASI DAN ANALISIS

Laporan ini menggunakan Simulink pada software MATLAB dengan pemrograman CMEC untuk pembentukan blok s-function

3.1. Parameter IM pada Simulasi

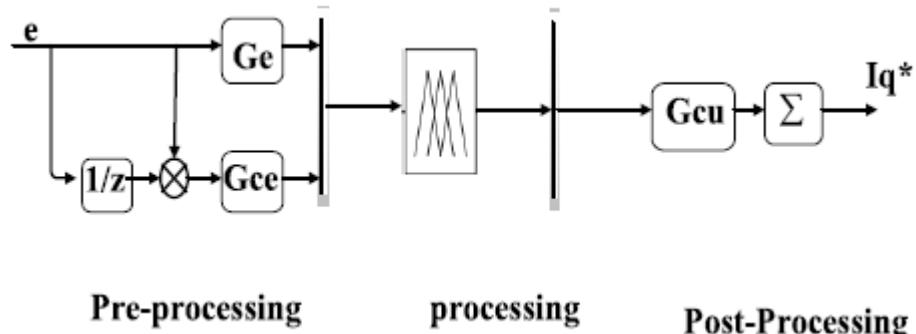
Parameter IM yang digunakan adalah sebagai berikut:

Item	Symbol	Value	Unit
Stator resistance	R_s	2.76	[Ω]
Rotor resistance	R_r	2.90	[Ω]
Stator inductance	L_s	234.9	[mH]
Rotor inductance	L_r	234.9	[mH]
Mutual inductance	L_m	227.9	[mH]

Untuk persamaan model motor menggunakan persamaan pada landasan teori.

3.2. Pemodelan Fuzzy Logic Controller pada Simulasi

Fuzzy Logic Controller digunakan untuk memproses nilai kecepatan arus q reference untuk diberikan pada motor induksi dengan masukan error dari kecepatan motor. Blok diagram dari FLC tersebut dapat dilihat pada gambar berikut.



Nilai error diperoleh dari selisih dari nilai kecepatan sekarang dan kecepatan yang diinginkan. Kemudian dari error tersebut dicari nilai change of error dengan cara mencari selisih antara error sekarang dengan error pada waktu cuplik sebelumnya.

Pada setiap masukan dan luaran juga diberikan sebuah scaling factor, yaitu sebuah pengali yang dipilih secara efisien agar menghasilkan peforma yang terbaik. Terdapat 3 buah scaling factor yaitu 2 pada masukan GE dan GCE dan 1 pada luaran yaitu GCU.

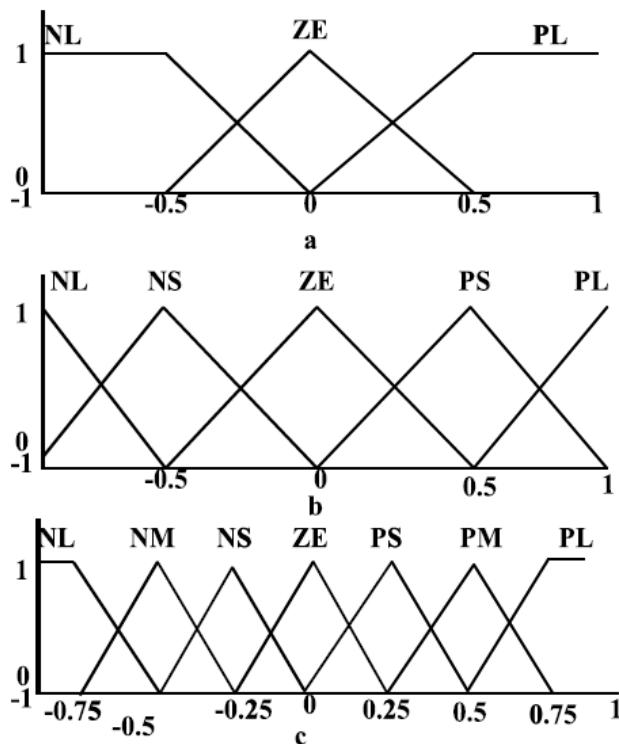
Pada laporan ini juga menggunakan beberapa kombinasi dari parameter fungsi keanggotaan dan rulebase untuk membandingkan peforma dari masing – masing rulebase.

3.2.1. Fungsi Keanggotaan FLC

Terdapat 3 buah fungsi keanggotaan yang digunakan. Masing – masing fungsi keanggotaan memiliki jumlah variabel linguistik yang berbeda. Terdapat 3 variasi yaitu:

- Menggunakan 3 buah variabel linguistik (NL, ZE, PL)
- Menggunakan 5 buah variabel linguistik (NL, NS, ZE, PS, PL)
- Menggunakan 7 buah variabel linguistic (NL,NM,NS,ZE,PS,PM,PL)

Range dari nilai – nilai variabel linguistik tersebut dapat dilihat pada Gambar berikut.



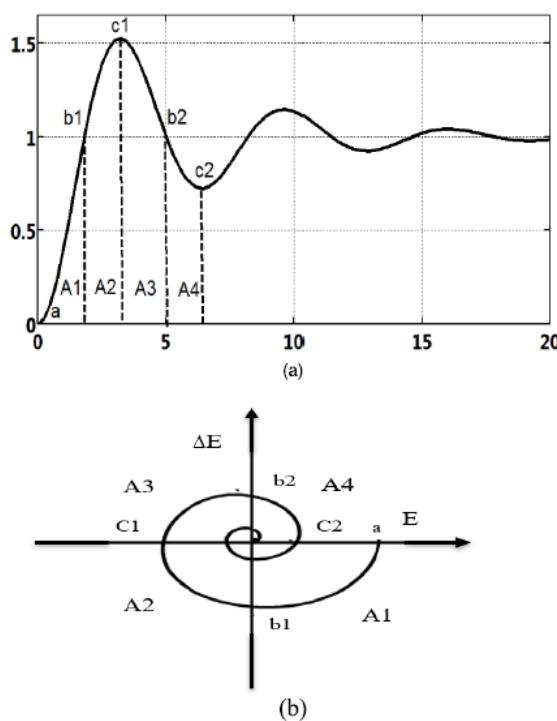
Sehingga secara total terdapat 7 buah variabel linguistik yang digunakan yaitu Negative Small (NS), Negative Medium (NM), Negative Large (NL), Zero (ZE), Positive Small (PS), Positive Medium (PM), dan Positive Large (PL).

Membership function dengan 3 variabel linguistic digunakan pada rule base 3x3 dan Simplified 5. Membership function dengan 5 variabel lingustik digunakan pada rule base 5x5 dan Simplified 7. Membership function dengan 7 variabel linguistic digunakan pada rule base 7x7 dan Simplified 9.

3.2.2. Rule Base FLC

Metode Pengendali Fuzzy Logic yang ditawarkan mensitesa rule base dari menganalisa respon step dinamis dari motor induksi dan kemudian menghasilkan juga rule base yang merupakan simplifikasi dari rule base hasil sintesa dengan cara menggunakan konsep mencari rute terdekat untuk mencapai titik equilibrium.

Rule base tersebut dapat disintesa dengan melihat step respons orde 2 berdasarkan karakteristik dari sistem penggerak motor induksi. Metode sintesa ini dapat diterapkan pada motor induksi dan pada sistem orde 2 lainnya. Berikut merupakan contoh step respon orde 2.



Respon tersebut dapat dipisah menjadi 4 area menjadi A1-A4 yang dimana terdapat dua buah titik sebrang yaitu b1 dan b2 dan juga dua buah titik puncak dan lembah yaitu c1 dan c2. Respon tersebut kemudian dapat diubah menjadi bentuk phase plane. Rule base kemudian dapat disintesa dengan memikirkan respon apa yang seharusnya diberikan pada kedelapan situasi tersebut. Respon tersebut kemudian dapat disusun menjadi tabel mengikuti framework berikut ini.

E ΔE	NL	NM	NS	ZE	PS	PM	PL
PL	A3			b2	A4		
PM							
PS							
ZE	C1			ZE	C2		
NS	A2			b1	A1		
NM							
NL							

Rule base hasil sintesa serta pasangan simplifikasinya tersebut adalah sebagai berikut.

- **Rule base (3x3) dan Simplified 5**

Rule base (3x3) ini memiliki 3 fungsi keanggotaan dan disimplifikasi menjadi rule base Simplified 5 dengan mengambil 5 pasangan aturan dari rule base (3x3). Berikut merupakan tabel rule base tersebut serta rule base simplifikasinya.

E ΔE	NL	ZE	PL	* Simplified rules:
PL	ZE	PL	PL	1. IF{E is NL and ΔE is ZE}THEN ΔU is NL
ZE	NL	ZE	PL	2. IF{E is ZE and ΔE is PL}THEN ΔU is PL
NL	NL	NL	ZE	3. IF{E is ZE and ΔE is ZE}THEN ΔU is ZE

- **Rule base (5x5) dan Simplified 7**

Rule base (5x5) ini memiliki 5 fungsi keanggotaan dan disimplifikasi menjadi rule base Simplified 7 dengan mengambil 7 pasangan aturan

dari rule base (5x5). Berikut merupakan tabel rule base tersebut serta rule base simplifikasiannya.

ΔE	NL	NS	ZE	PS	PL
E	ZE	PS	PL	PL	PL
ΔE	NS	ZE	PS	PL	PL
E	NL	NS	ZE	PL	PL
ΔE	NL	NS	NS	PS	PS
E	NL	NL	NL	ZE	ZE

* Simplified rules :

1. $IF\{E \text{ is } NL \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } NL$
2. $IF\{E \text{ is } NS \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } NS$
3. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } PS\} THEN \Delta U \text{ is } PS$
4. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } ZE$
5. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } NS\} THEN \Delta U \text{ is } NS$
6. $IF\{E \text{ is } PS \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } PL$
7. $IF\{E \text{ is } PL \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } PL$

- **Rule base (7x7) dan Simplified 9**

Rule base (7x7) ini memiliki 7 fungsi keanggotaan dan disimplifikasi menjadi rule base Simplified 9 dengan mengambil 9 pasangan aturan dari rule base (7x7). Berikut merupakan tabel rule base tersebut serta rule base simplifikasiannya.

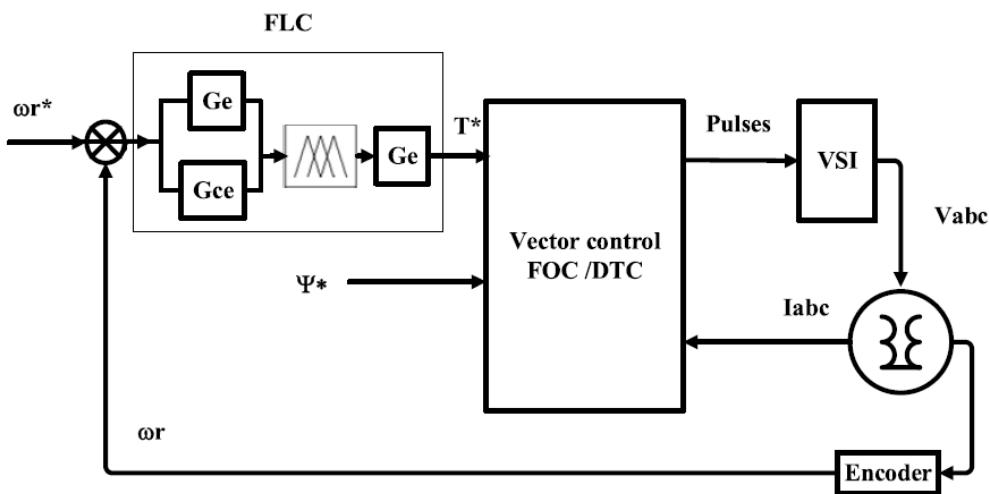
ΔE	NL	NM	NS	ZE	PS	PM	PL
E	ZE	PS	PS	PL	PL	PL	PL
ΔE	NS	ZE	PS	PM	PL	PL	PL
E	NS	NS	ZE	PS	PS	PL	PL
ΔE	NL	NM	NS	ZE	PS	PM	PL
E	NL	NL	NS	NS	ZE	PS	PS
ΔE	NL	NL	NL	NM	NS	ZE	PS
E	NL	NL	NL	NL	NS	NS	ZE

* Simplified rules :

1. $IF\{E \text{ is } NL \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } NL$
2. $IF\{E \text{ is } NM \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } NM$
3. $IF\{E \text{ is } NS \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } NS$
4. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } PS\} THEN \Delta U \text{ is } PS$
5. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } ZE$
6. $IF\{E \text{ is } ZE \text{ and } \Delta E \text{ is } NS\} THEN \Delta U \text{ is } NS$
7. $IF\{E \text{ is } PS \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } PS$
8. $IF\{E \text{ is } PM \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } PM$
9. $IF\{E \text{ is } PL \text{ and } \Delta E \text{ is } ZE\} THEN \Delta U \text{ is } PL$

3.3. Simulasi Pengendali

Blok diagram dari simulasi pada SIMULINK mengikuti desain pada paper referensi. Berikut merupakan blok diagram pada paper referensi.



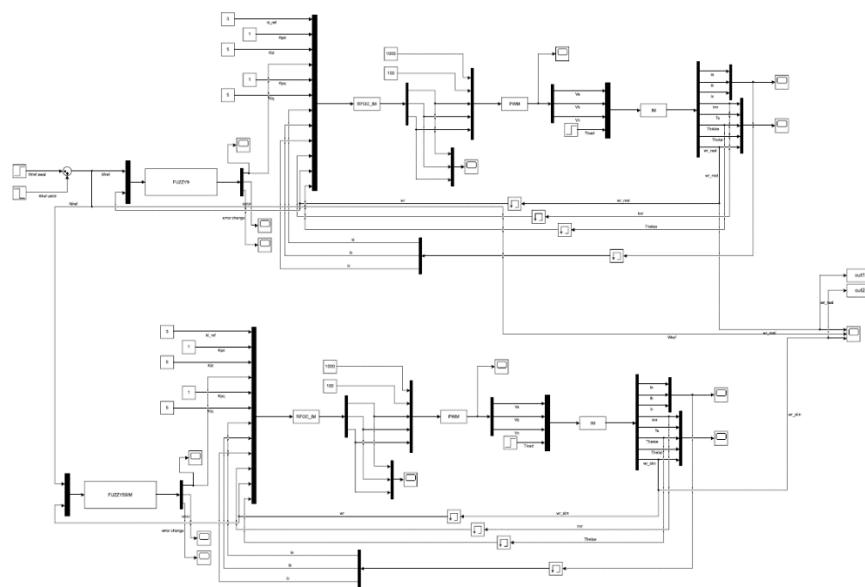
Blok diagram tersebut menggunakan masukan yaitu kecepatan referensi, kemudian membandingkan kecepatan referensi tersebut dengan kecepatan aktual untuk mendapatkan error kecepatan. Kecepatan aktual didapatkan dari model motor dengan menggunakan encoder. Hasil error tersebut kemudian digunakan sebagai masukan dari FLC untuk menghasilkan Torsi referensi sesuai dengan kaidah FLC yang sudah dijelaskan pada Bab 3.2.

Torsi referensi dan Fluks referensi kemudian dimasukkan kedalam Field Oriented Control. Model FOC yang digunakan adalah model yang dijelaskan pada Bab 2.3. Keluaran FOC kemudian akan menjadi sinyal pengendali 3 fasa yang kemudian akan diubah menjadi tegangan masukan 3 fasa untuk motor induksi oleh VSI. Metode VSI yang digunakan pada laporan ini adalah Sinusoidal Pulse Width Modulation yang sudah dijelaskan pada Bab 2.5. Kemudian tegangan 3 fasa pwm

tersebut akan menjadi masukan dari motor dimana model motor akan mengeluarkan arus 3 fasa dan kecepatan aktual untuk diumpam balik ke FLC dan FOC. Model motor yang digunakan adalah model yang dijelaskan pada Bab 2.1.

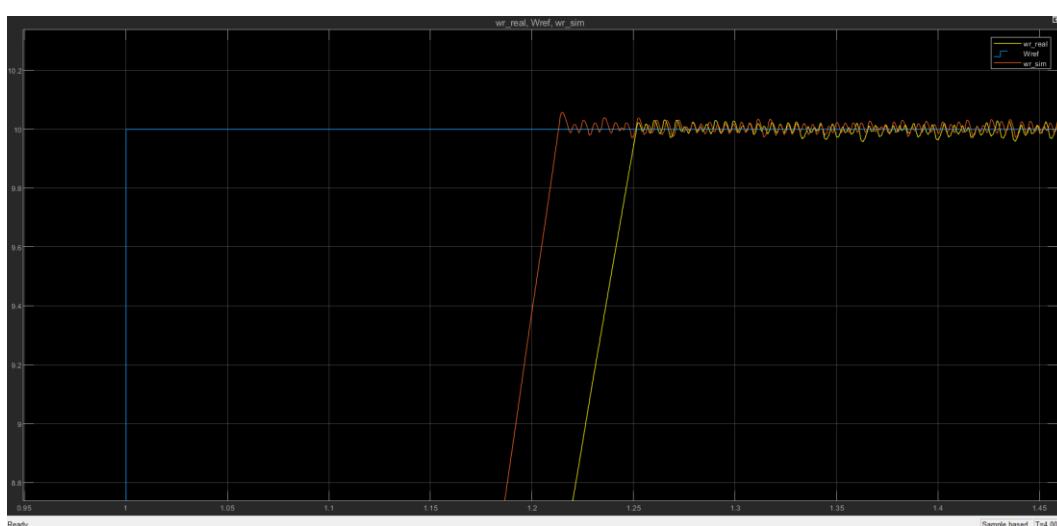
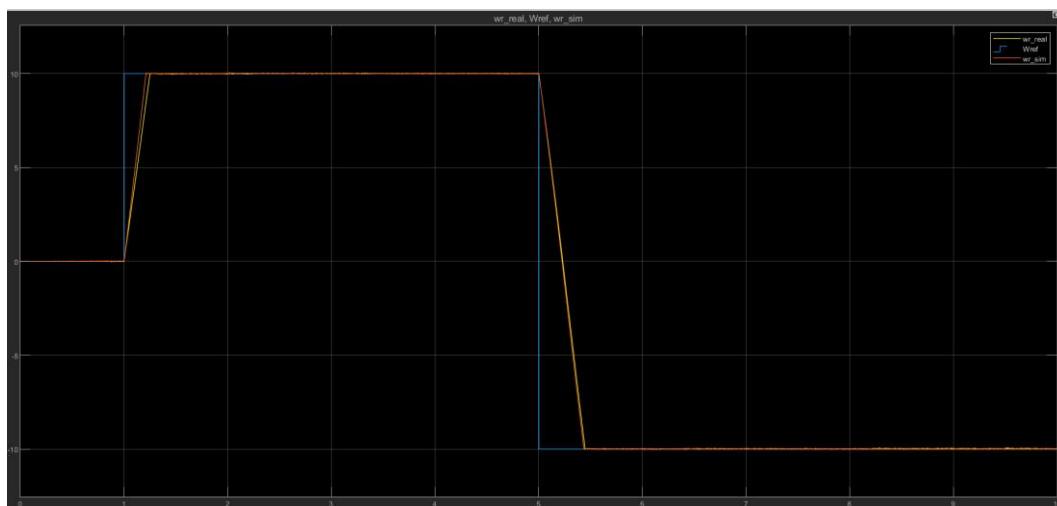
3.3.1. Simulasi Rule Base (3x3) dan Simplified 5

Blok diagram yang digunakan dalam simulasi Rule Base (3x3) dan Simplified 5 ditunjukkan oleh gambar berikut.



Pada simulasi tersebut menggunakan Gain KI sebesar 5, KD sebesar 1, Id ref sebesar 3 untuk RFOC serta menggunakan VDC sebesar 100 Volt dengan frekuensi sebesar 1000 Hz. Hasil yang diperoleh dari simulasi tersebut adalah sebagai berikut.

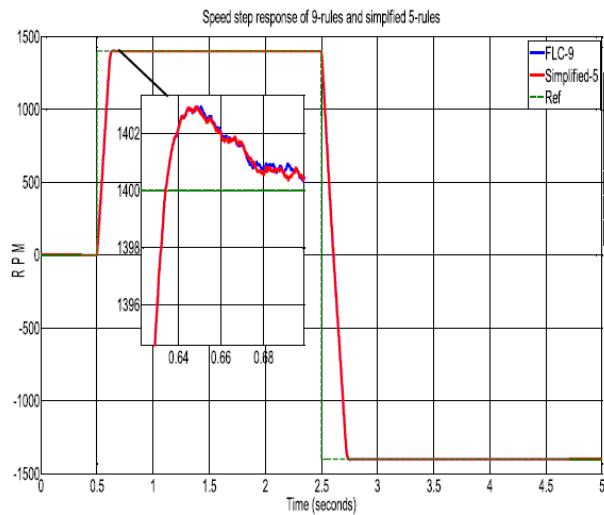
- Kecepatan



Hasil dari FLC dengan rule base (3x3) ditunjukkan oleh garis berwarna oranye, hasil FLC dengan rule base Simplified 5 ditunjukkan oleh garis berwarna kuning, serta kecepatan reference ditunjukkan oleh garis berwarna biru.

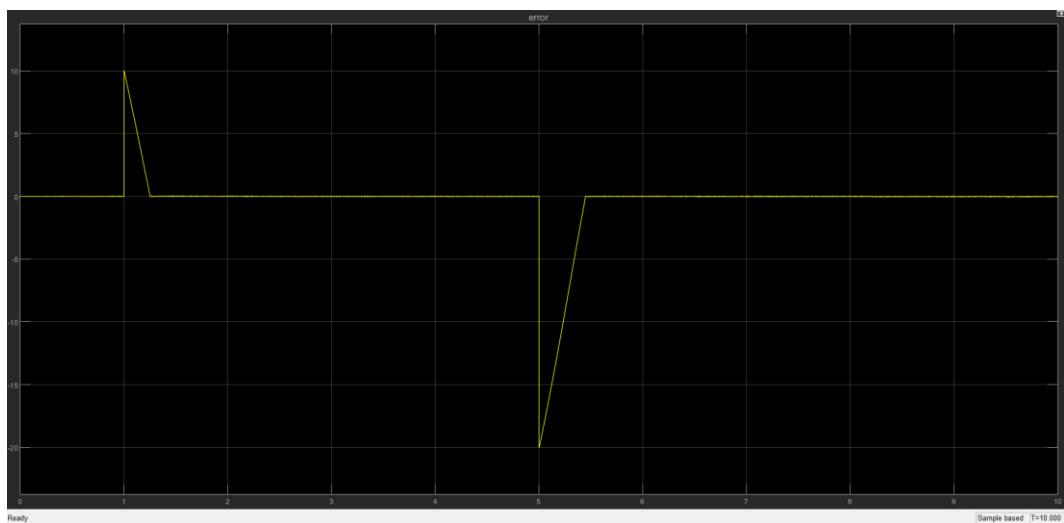
Setpoint awalnya diberikan kecepatan positif dapat terlihat bahwa kedua FLC sudah dapat mengikuti setpoint yang diberikan dan kemudian pada $t=5$ setpoint diubah menjadi negatif dengan amplitude yang sama dan juga kedua FLC sudah dapat mengikuti kembali setpoint yang diubah. Terdapat sedikit perbedaan ketika plot diperbesar dimana kecepatan hasil pengendali simplifikasi menghasilkan respons yang lebih cepat sekitar 0.5s dibandingkan hasil pengendali non simplifikasi.

Berikut merupakan hasil kecepatan dari paper referensi.

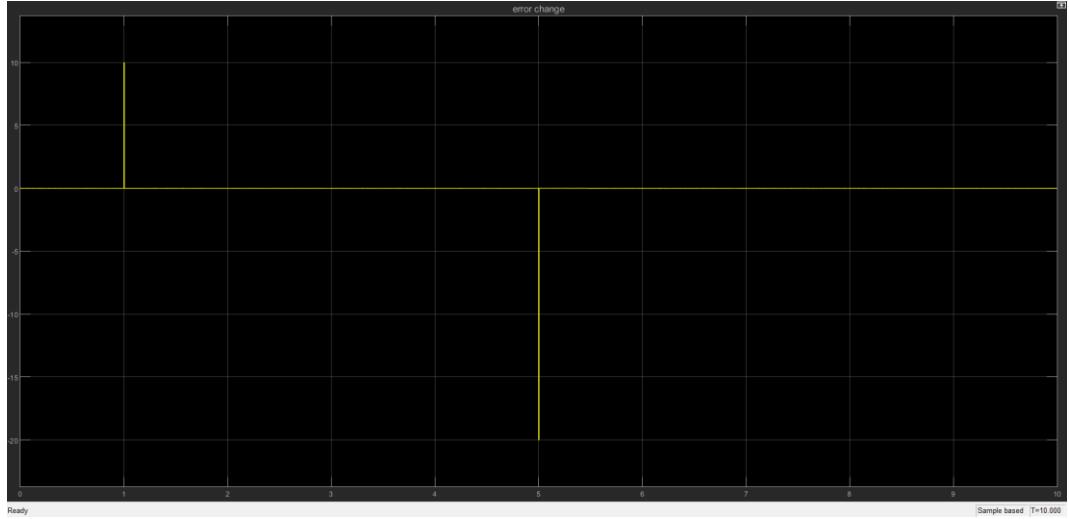


Diperoleh hasil lain juga selain kecepatan yaitu:

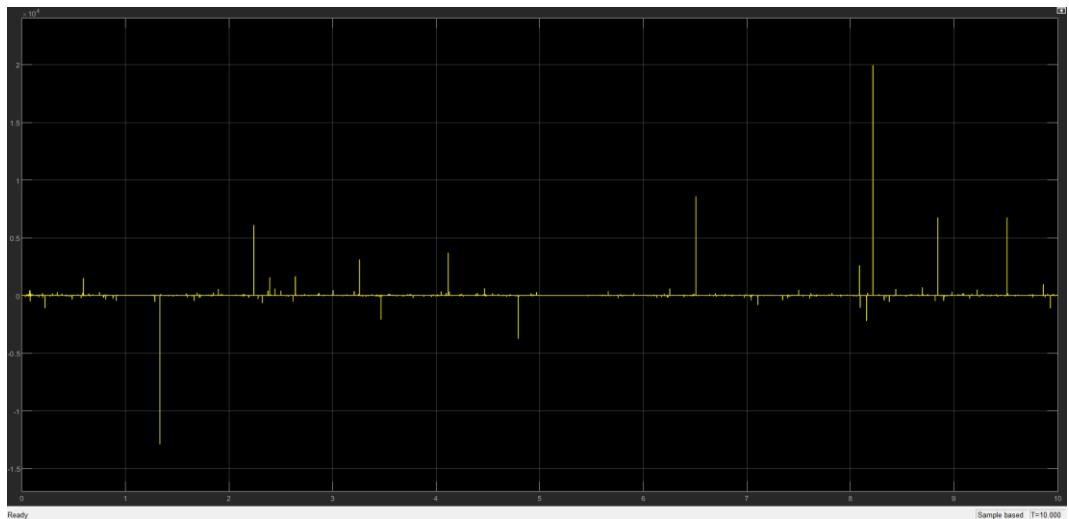
- **Rule Base (3x3)**
- **error FLC**



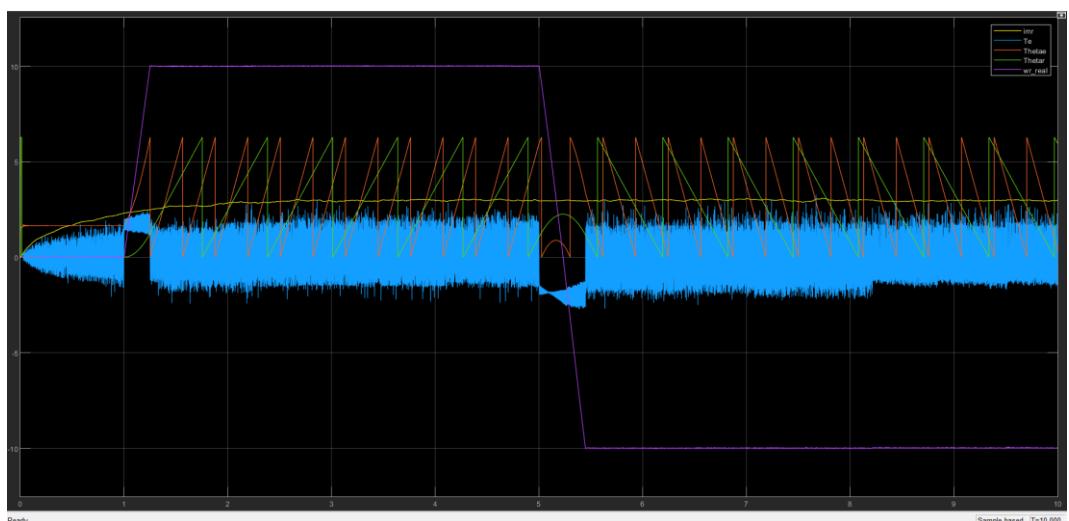
- error change FLC



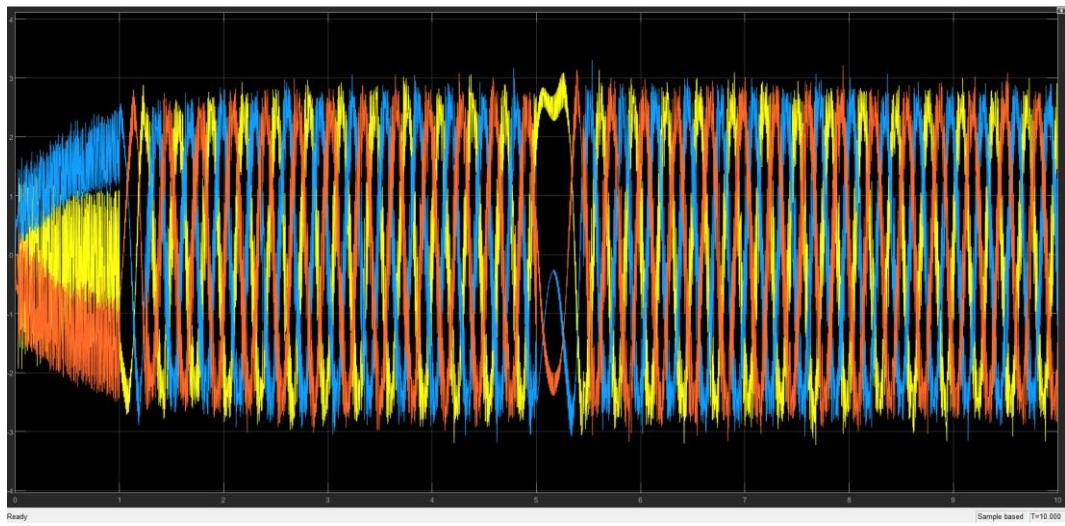
- Iq ref



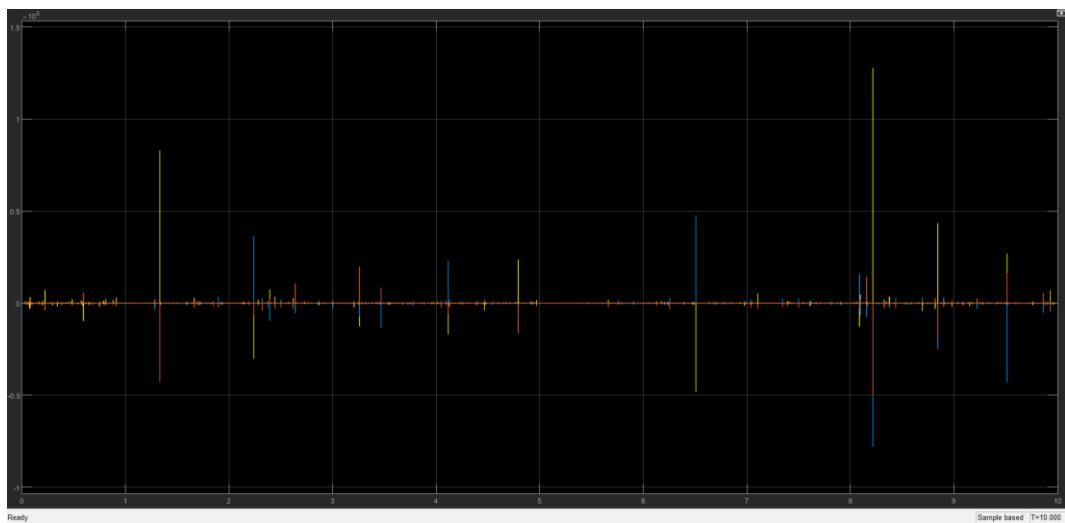
- imr, Te, θe, θr.



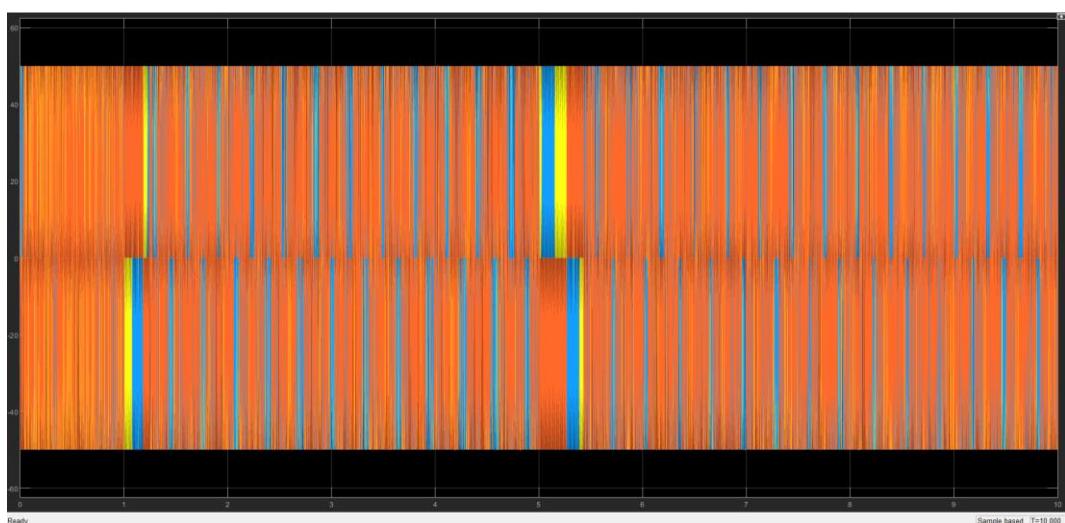
- **Ia,Ib,Ic**



- **Va,Vb,Vc**

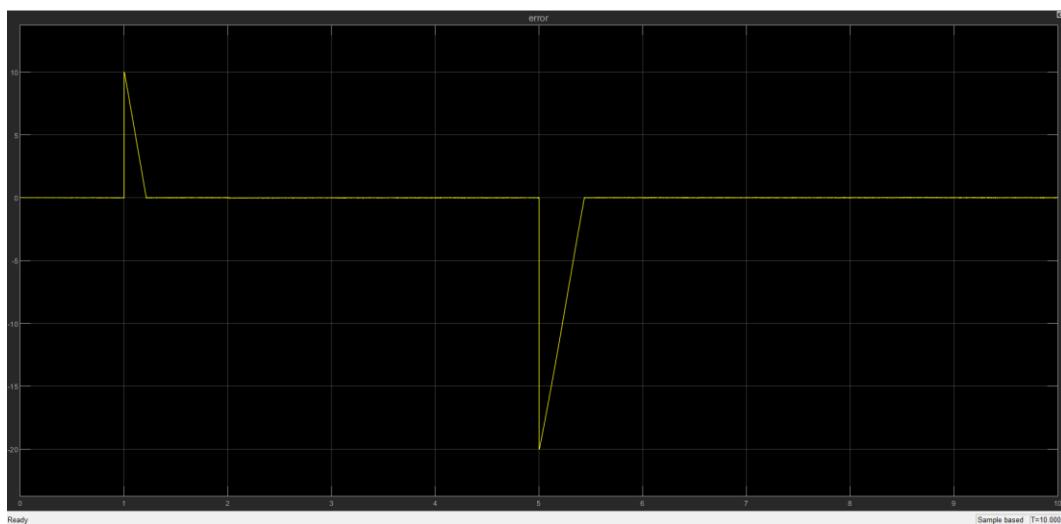


- **PWMa,PWMb,PWMc**

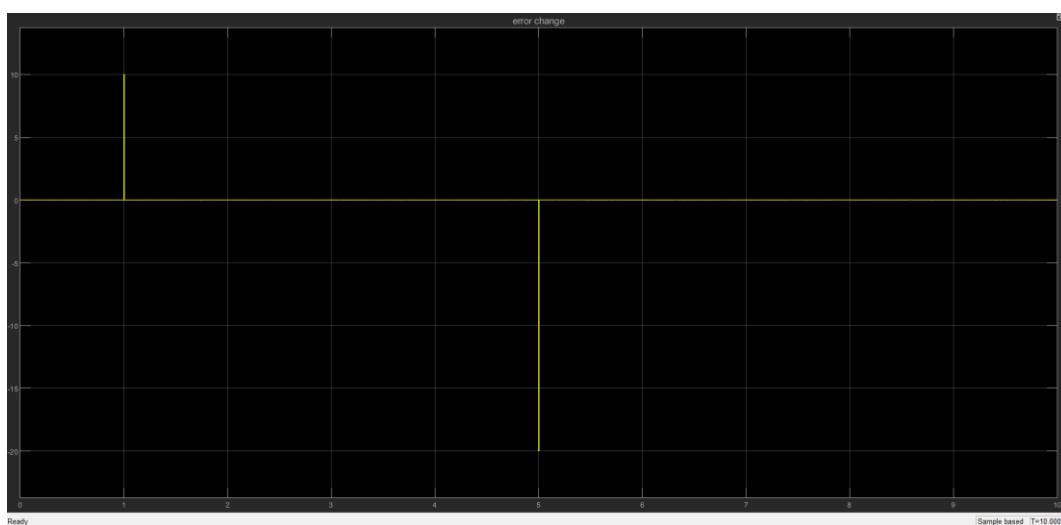


➤ **Rule Base Simplified 5**

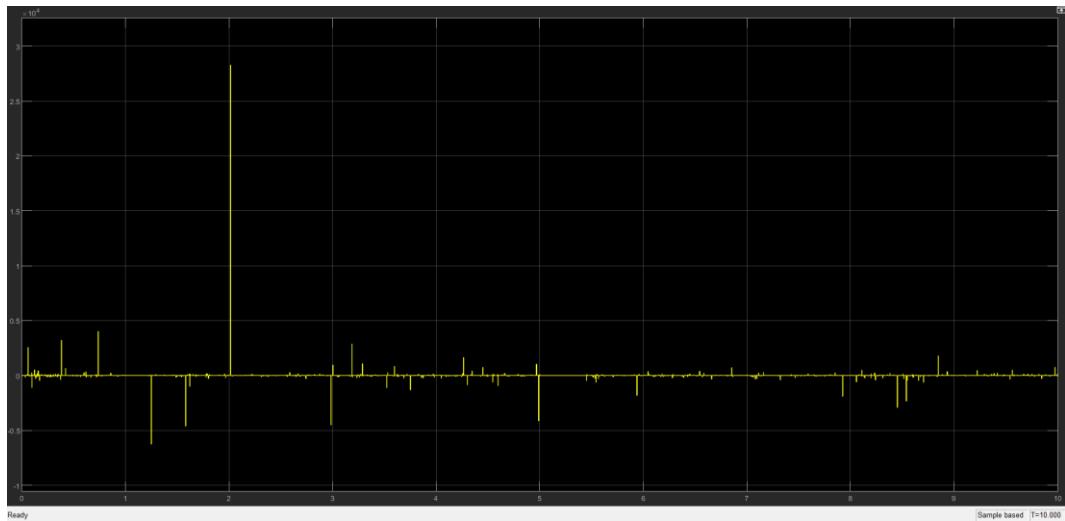
- **error FLC**



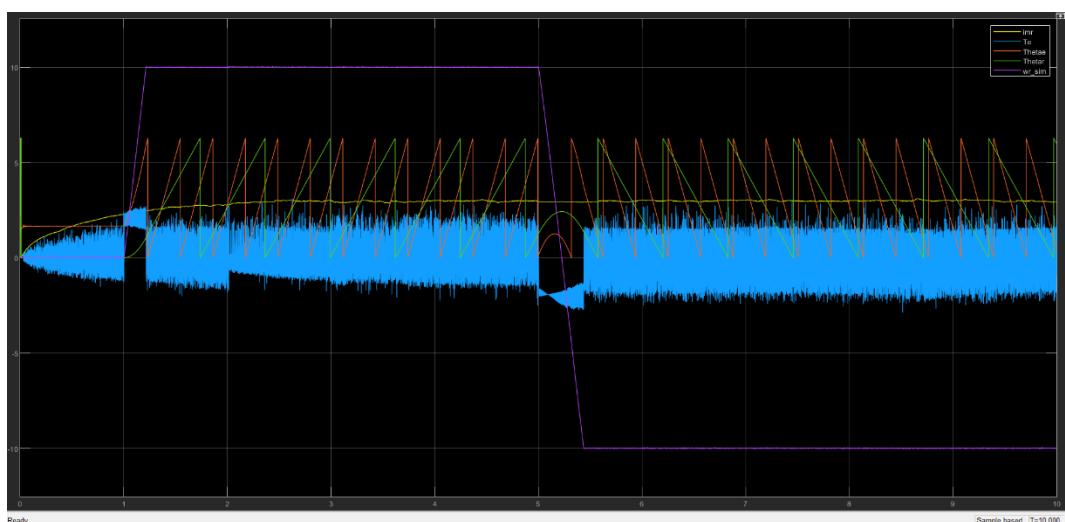
- **error change FLC**



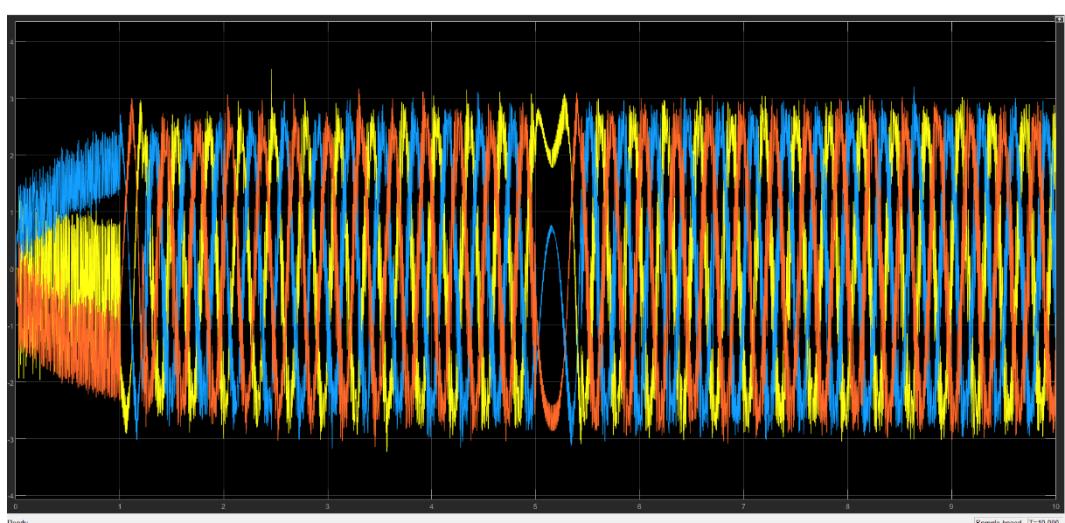
- I_q ref



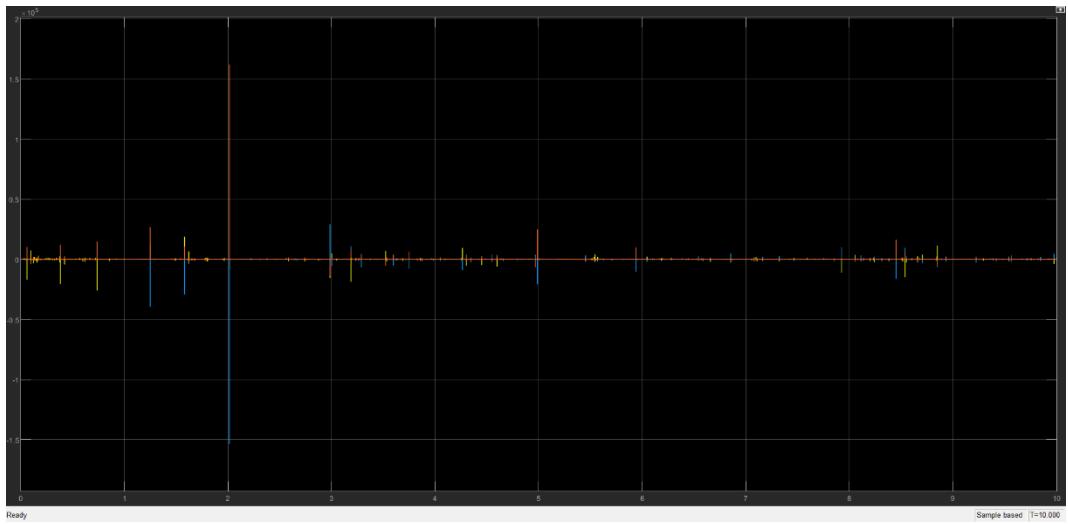
- $imr, Te, \theta e, \theta r$



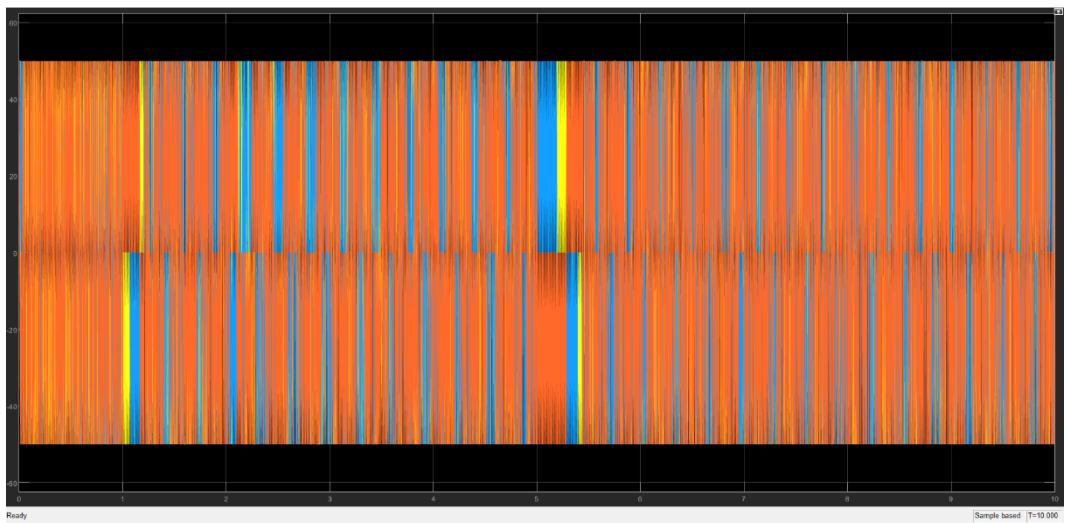
- I_a, I_b, I_c



- **V_a,V_b,V_c**

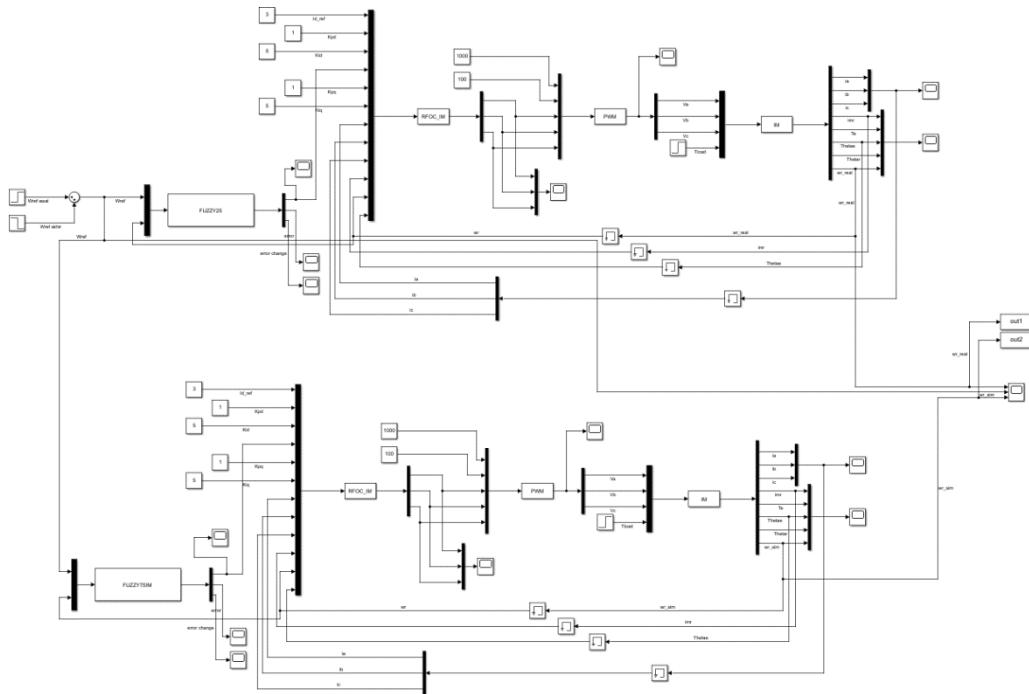


- **PWM_a,PWM_b,PWM_c**



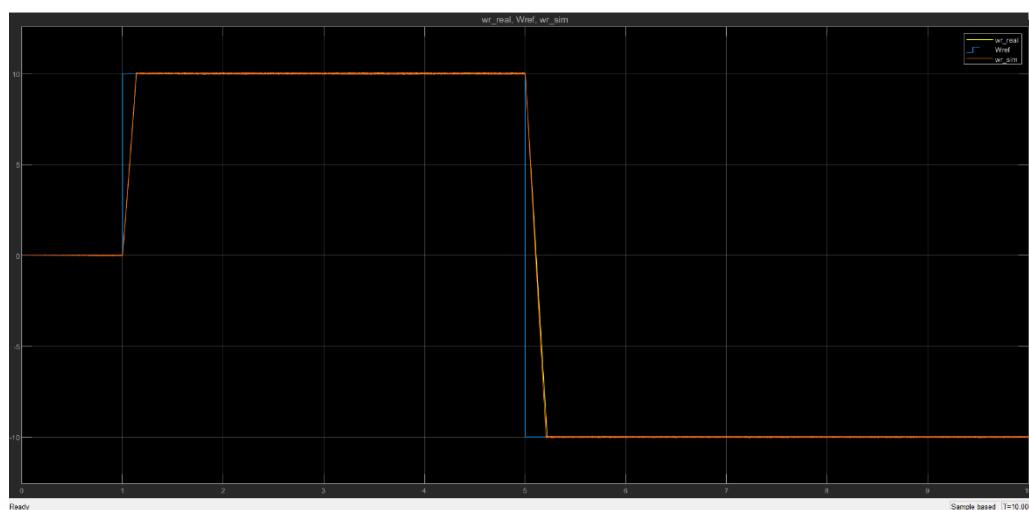
3.3.2. Simulasi Rule Base (5x5) dan Simplified 7

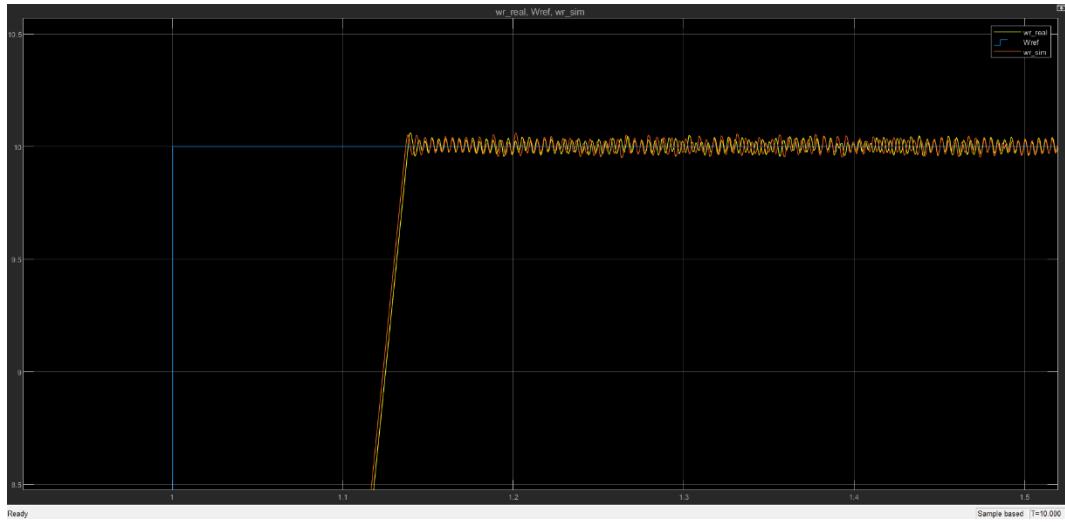
Blok diagram yang digunakan dalam simulasi Rule Base (5x5) dan Simplified 7 ditunjukkan oleh gambar berikut.



Pada simulasi tersebut menggunakan Gain KI sebesar 5, KD sebesar 1, Id ref sebesar 3 untuk RFOC serta menggunakan VDC sebesar 100 Volt dengan frekuensi sebesar 1000 Hz. Hasil yang diperoleh dari simulasi tersebut adalah sebagai berikut.

- **Kecepatan**

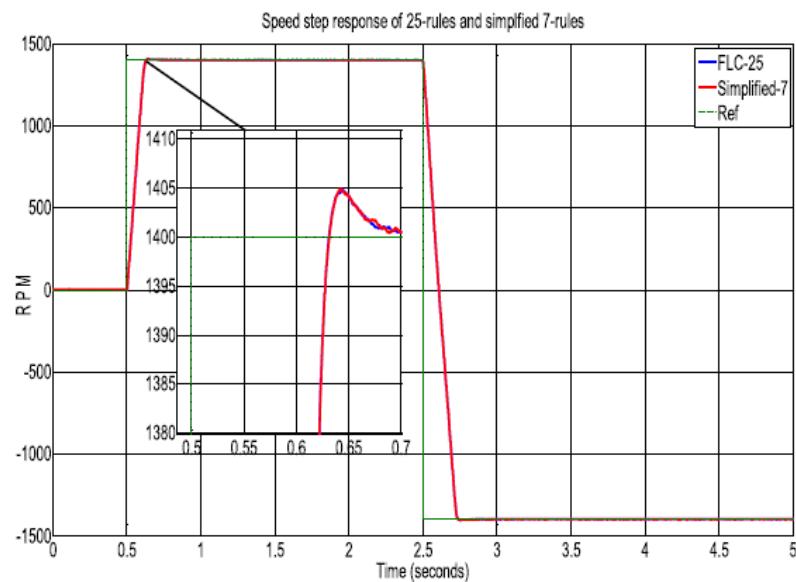




Hasil dari FLC dengan rule base (5x5) ditunjukkan oleh garis berwarna oranye, hasil FLC dengan rule base Simplified 7 ditunjukkan oleh garis berwarna kuning, serta kecepatan reference ditunjukkan oleh garis berwarna biru.

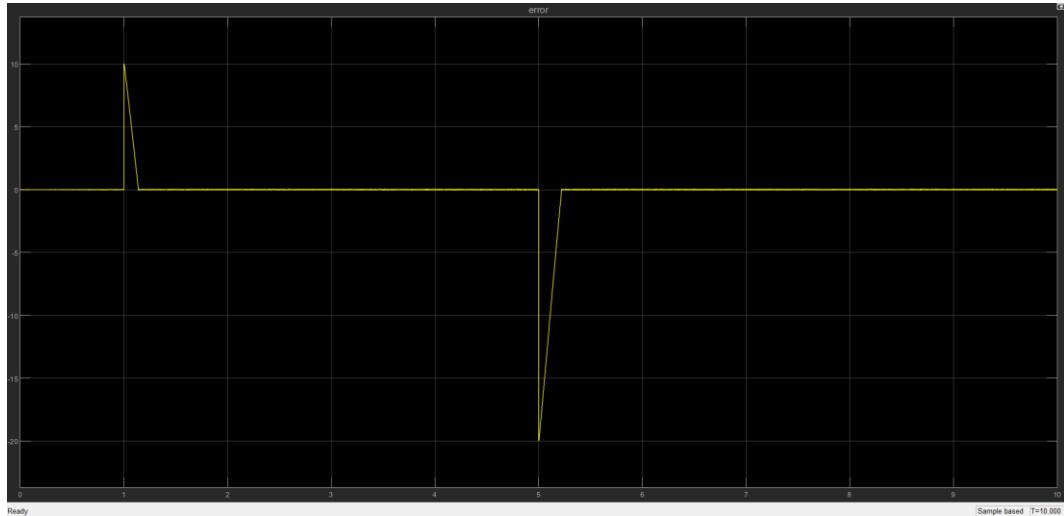
Setpoint awalnya diberikan kecepatan positif dapat terlihat bahwa kedua FLC sudah dapat mengikuti setpoint yang diberikan dan kemudian pada $t=5$ setpoint diubah menjadi negatif dengan amplitude yang sama dan juga kedua FLC sudah dapat mengikuti kembali setpoint yang diubah. Hasil antara kedua pengendali sudah cukup sama dan sesuai dengan paper referensi.

Berikut merupakan hasil kecepatan dari paper referensi.

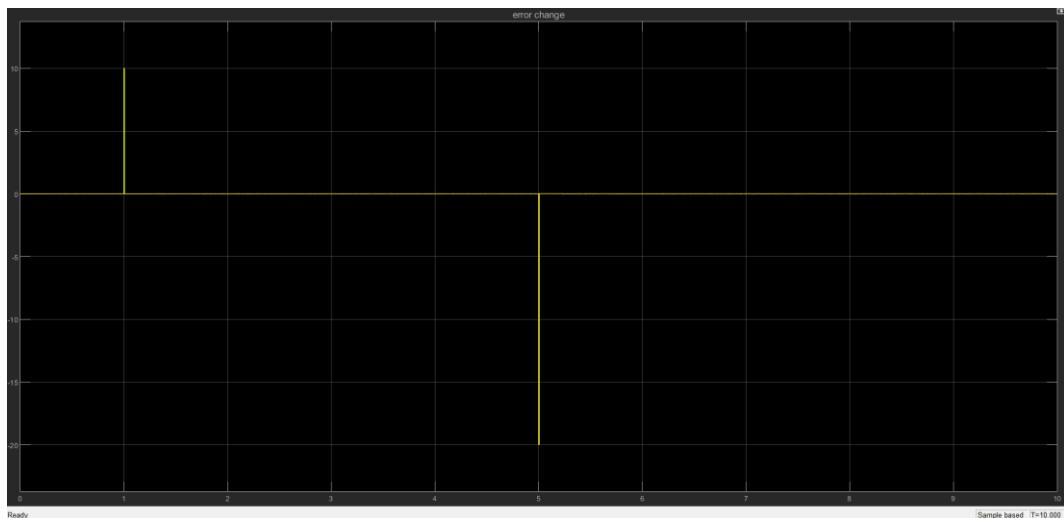


Diperoleh hasil lain juga selain kecepatan yaitu:

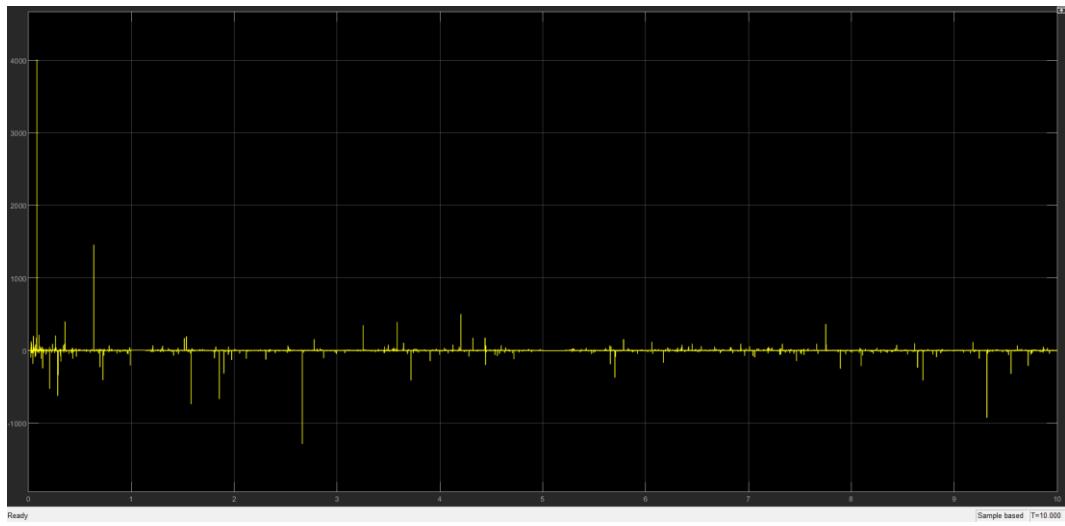
- **Rule Base (5x5)**
 - error FLC



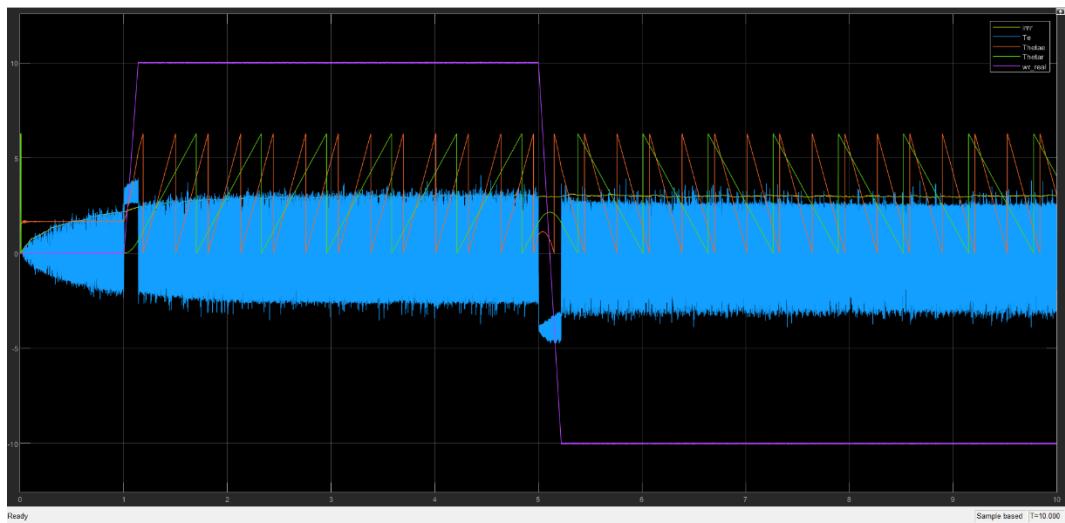
- error change FLC



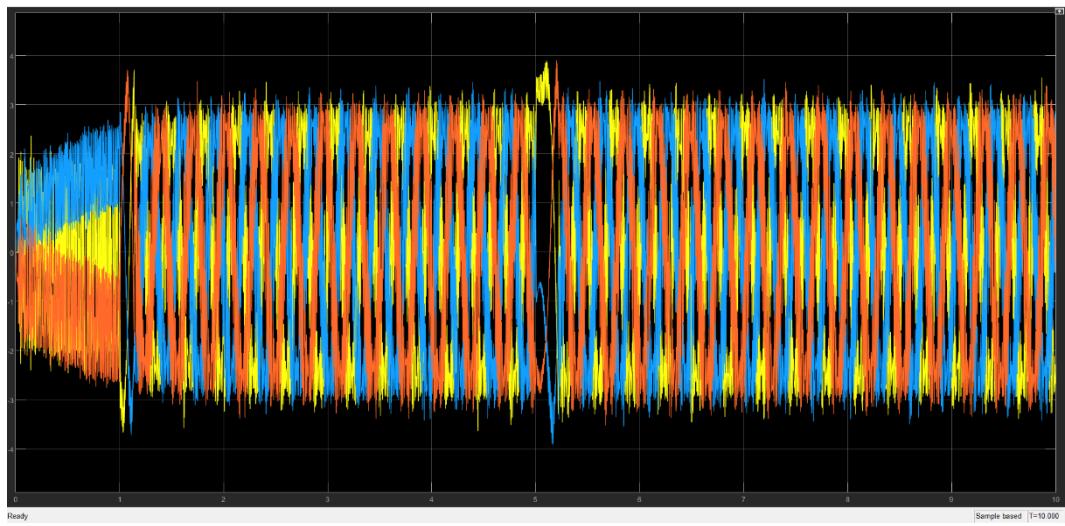
- $I_{q \text{ ref}}$



- $\text{imr}, T_e, \theta e, \theta r$



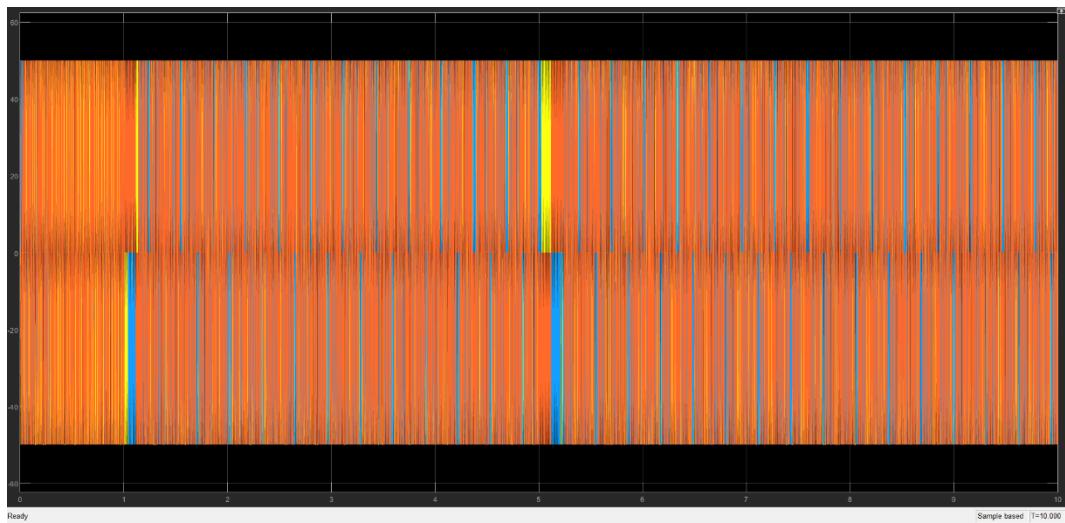
- **Ia,Ib,Ic**



- **Va,Vb,Vc**

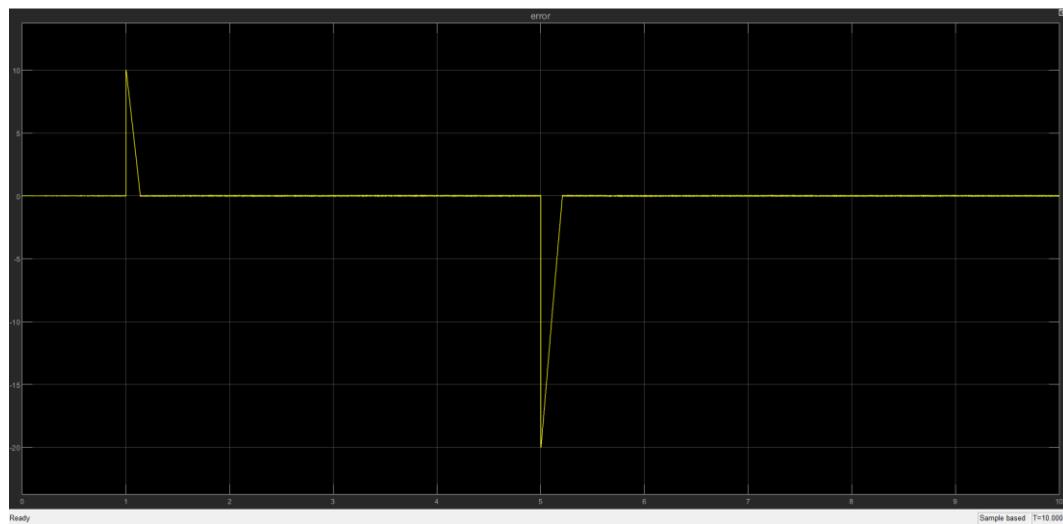


- **PWMa,PWMb,PWMcs**

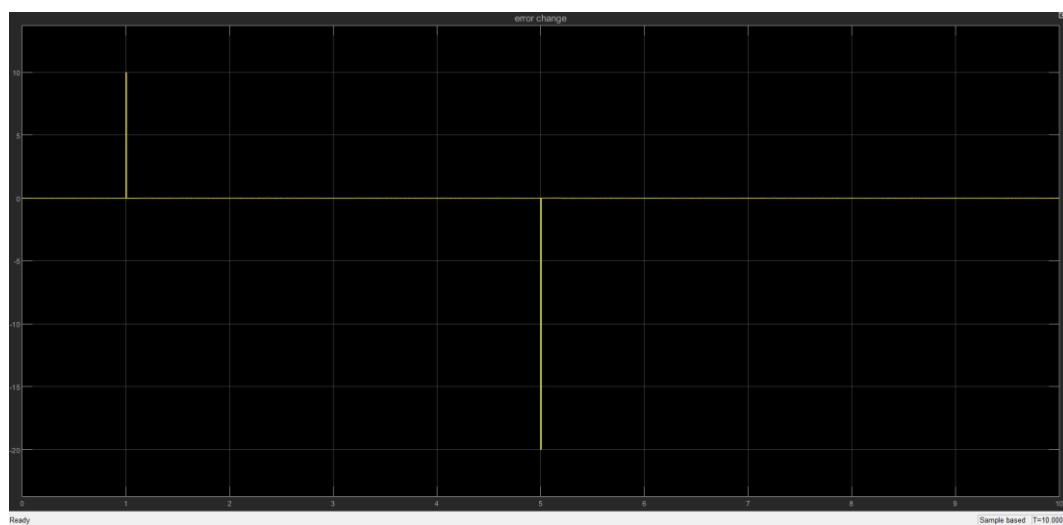


➤ **Rule Base Simplified 7**

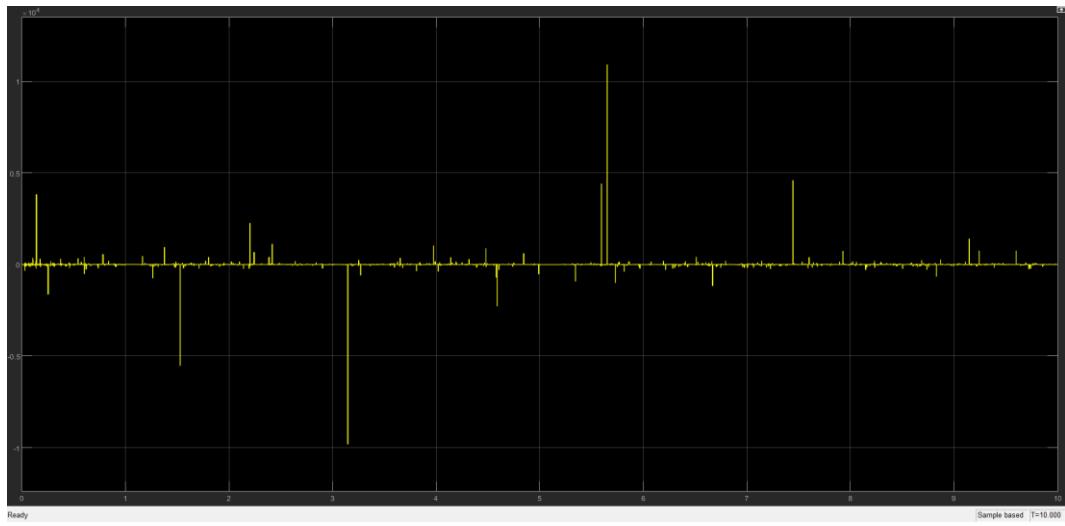
- **error FLC**



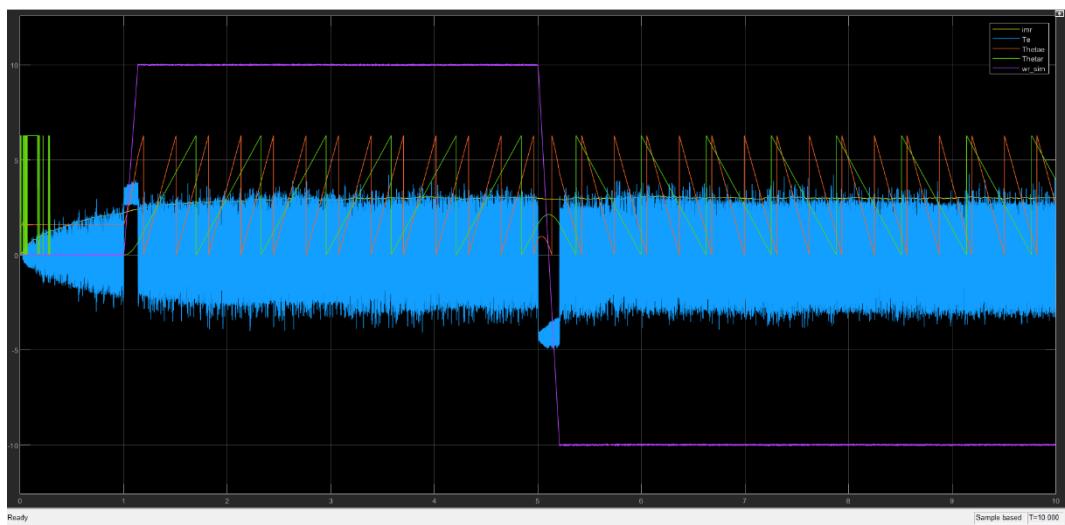
- **error change FLC**



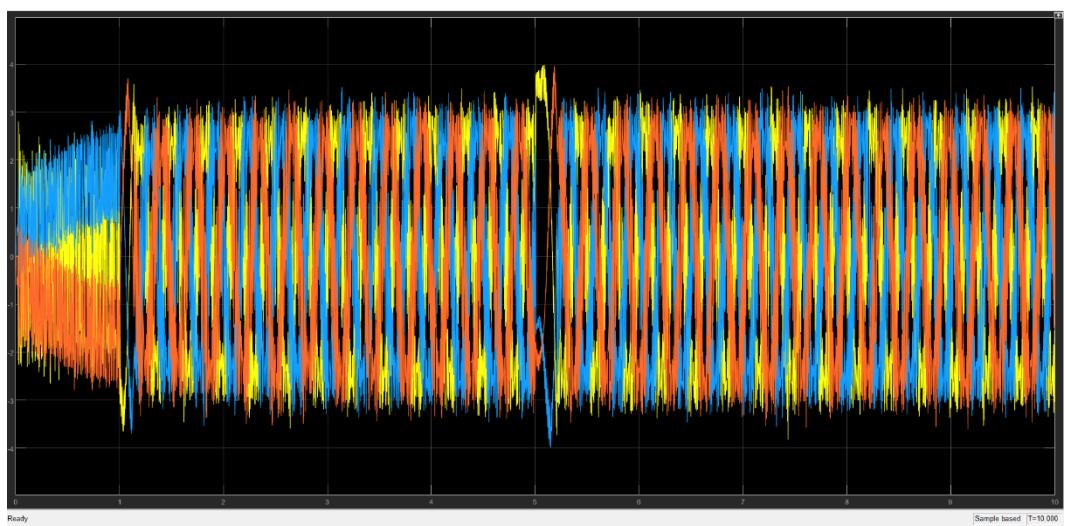
- **Iq ref**



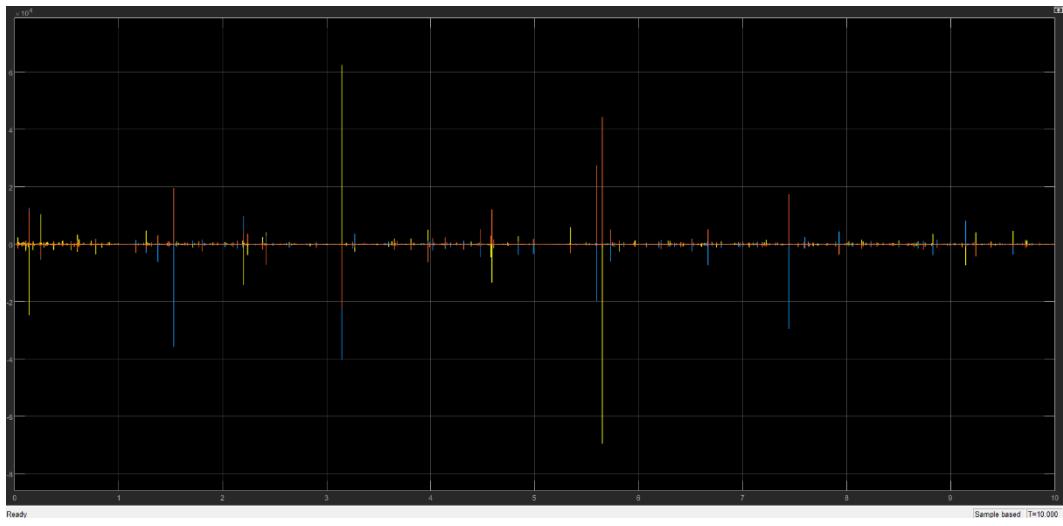
- **imr, Te, θe, θr**



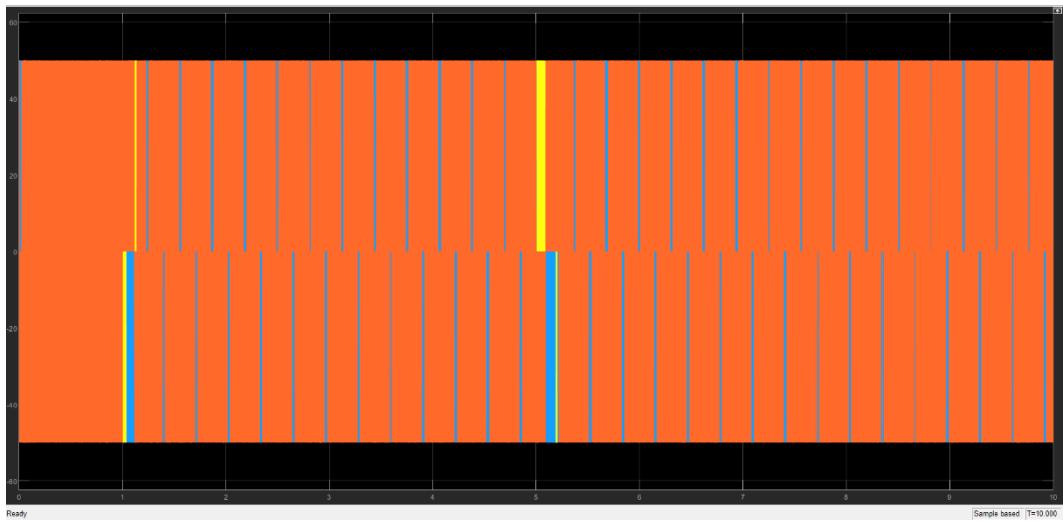
- **Ia,Ib,Ic**



- **V_a,V_b,V_c**

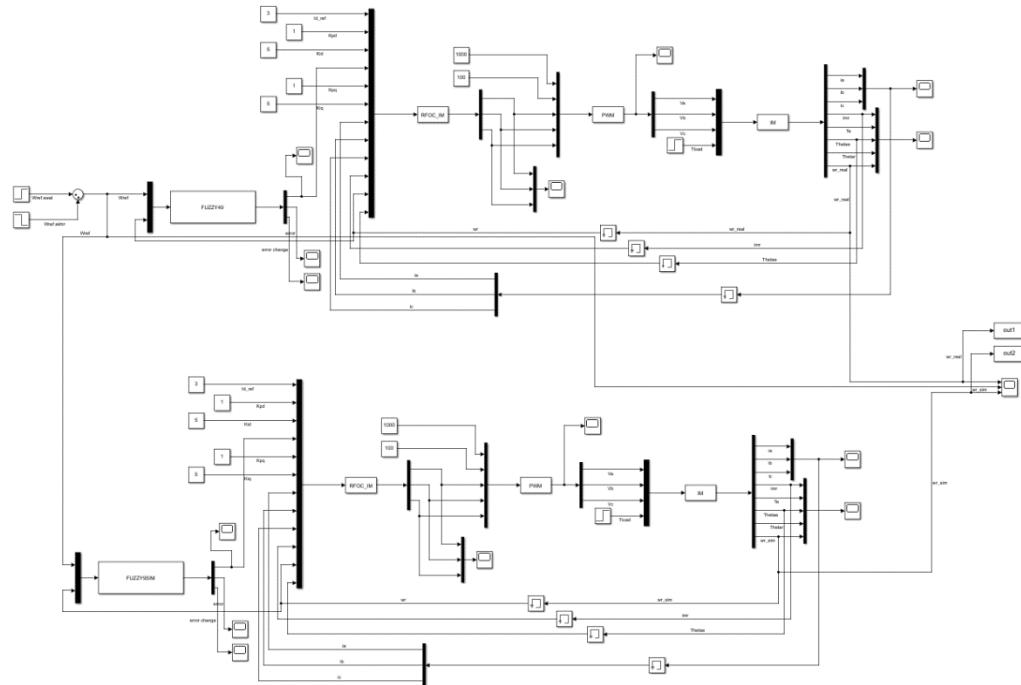


- **PWM_a,PWM_b,PWM_c**



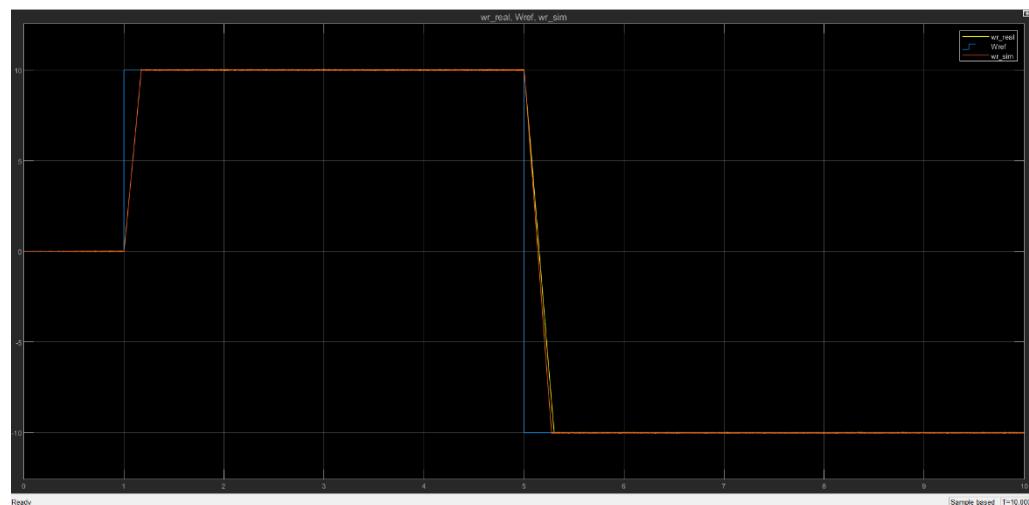
3.3.3. Simulasi Rule Base (7x7) dan Simplified 9

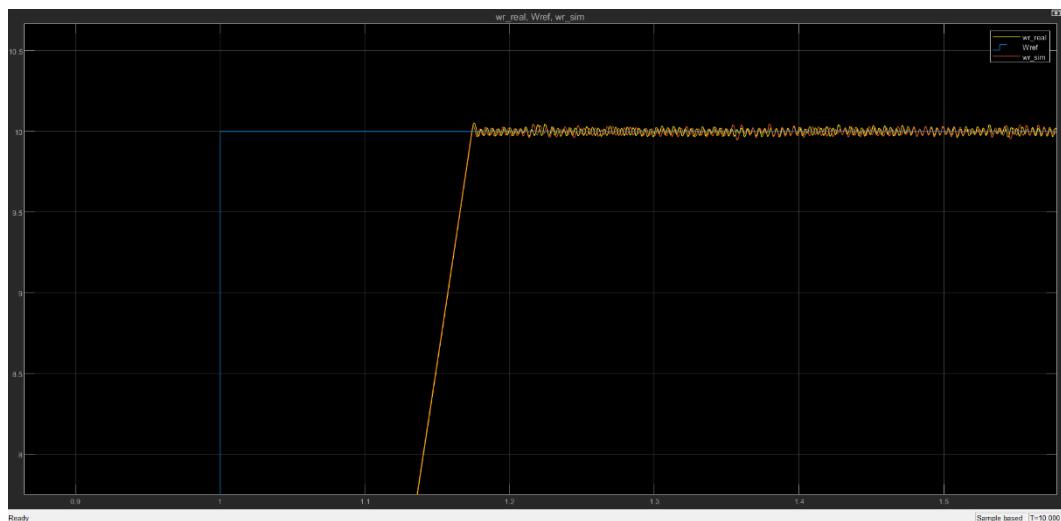
Blok diagram yang digunakan dalam simulasi Rule Base (7x7) dan Simplified 9 ditunjukkan oleh gambar berikut.



Pada simulasi tersebut menggunakan Gain KI sebesar 5, KD sebesar 1, Id ref sebesar 3 untuk RFOC serta menggunakan VDC sebesar 100 Volt dengan frekuensi sebesar 1000 Hz. Hasil yang diperoleh dari simulasi tersebut adalah sebagai berikut.

- **Kecepatan**

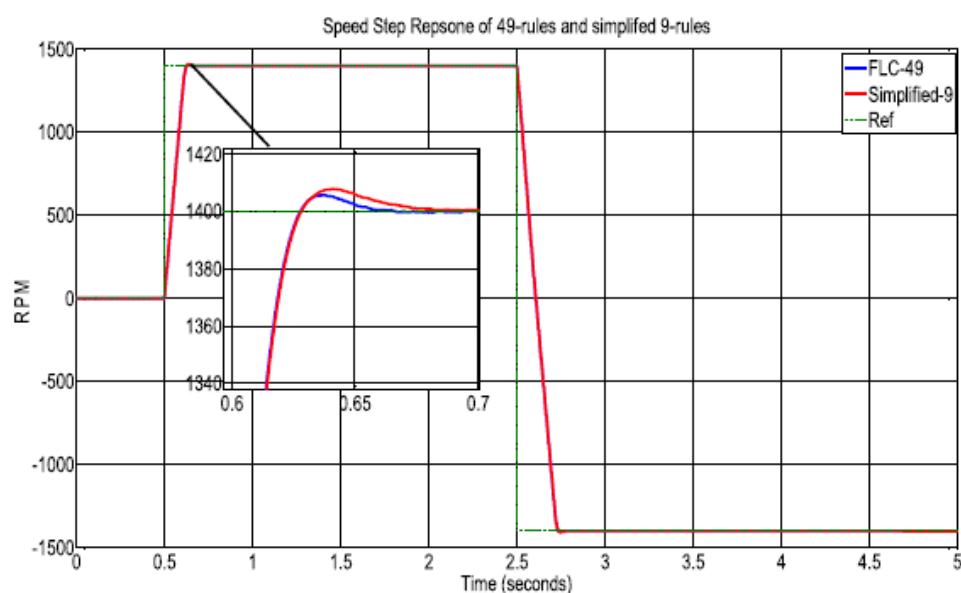




Hasil dari FLC dengan rule base (7x7) ditunjukkan oleh garis berwarna oranye, hasil FLC dengan rule base Simplified 9 ditunjukkan oleh garis berwarna kuning, serta kecepatan reference ditunjukkan oleh garis berwarna biru.

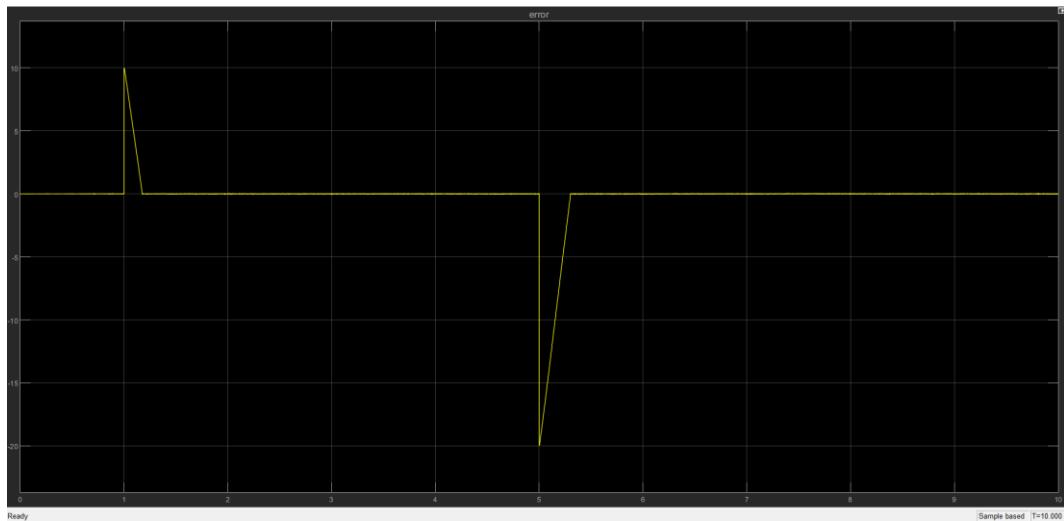
Setpoint awalnya diberikan kecepatan positif dapat terlihat bahwa kedua FLC sudah dapat mengikuti setpoint yang diberikan dan kemudian pada $t=5$ setpoint diubah menjadi negatif dengan amplitude yang sama dan juga kedua FLC sudah dapat mengikuti kembali setpoint yang diubah. Hasil antara kedua pengendali sudah cukup sama dan sesuai dengan paper referensi.

Berikut merupakan hasil kecepatan dari paper referensi.

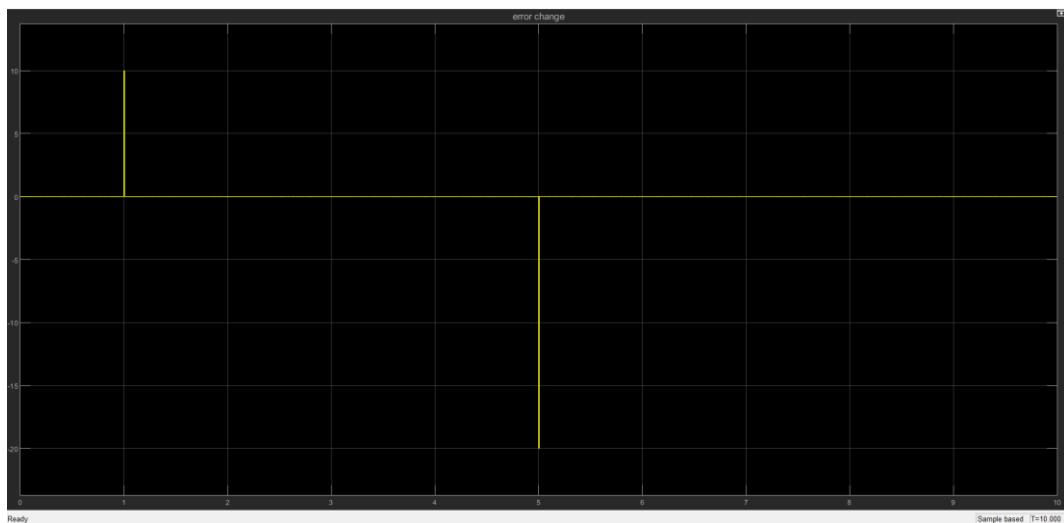


Diperoleh hasil lain juga selain kecepatan yaitu:

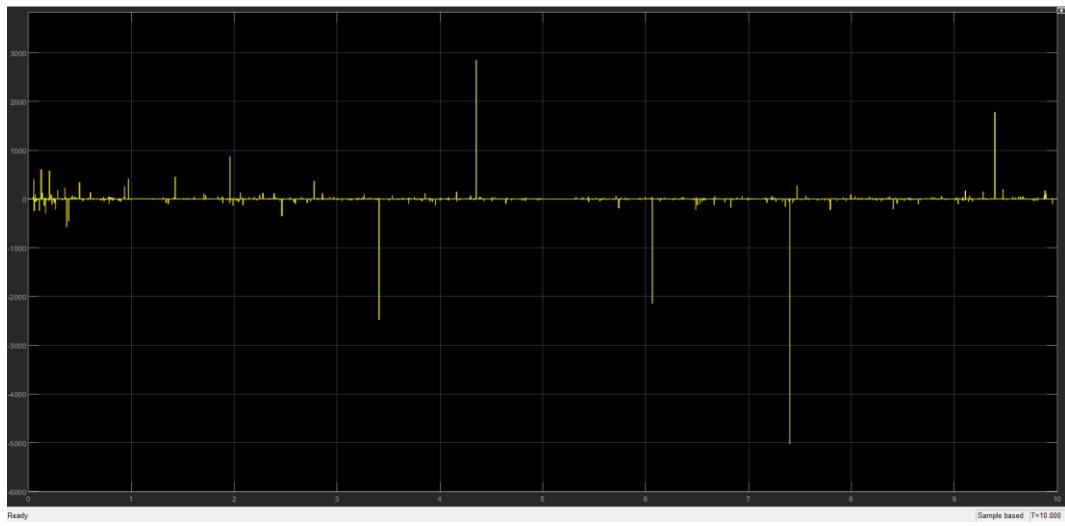
- **Rule Base (7x7)**
 - error FLC



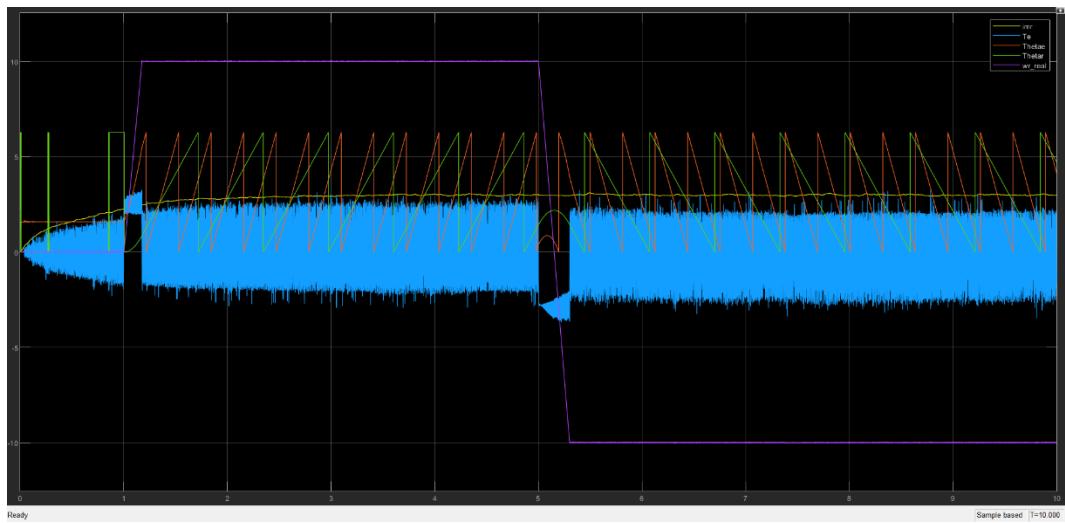
- error change FLC



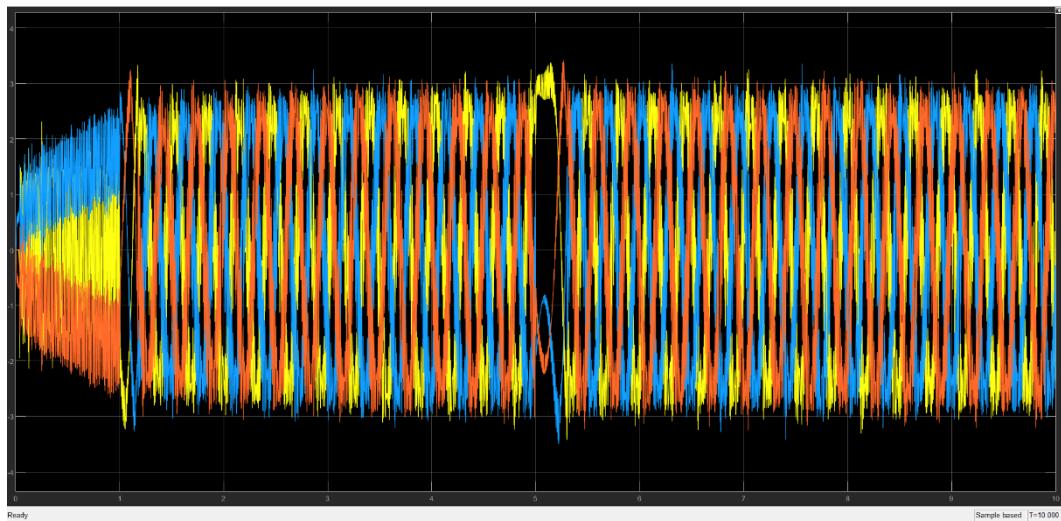
- $I_{q \text{ ref}}$



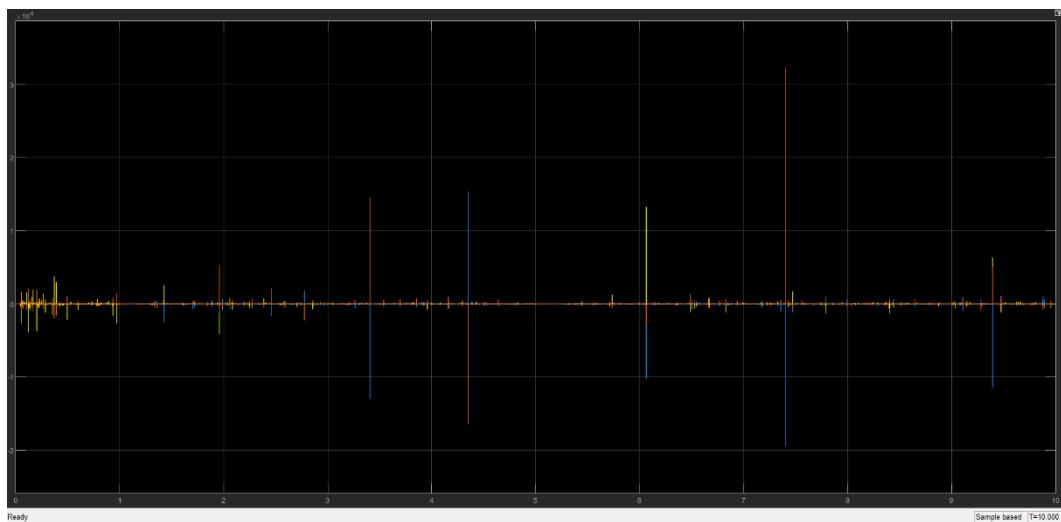
- $i_{mr}, T_e, \theta e, \theta r$



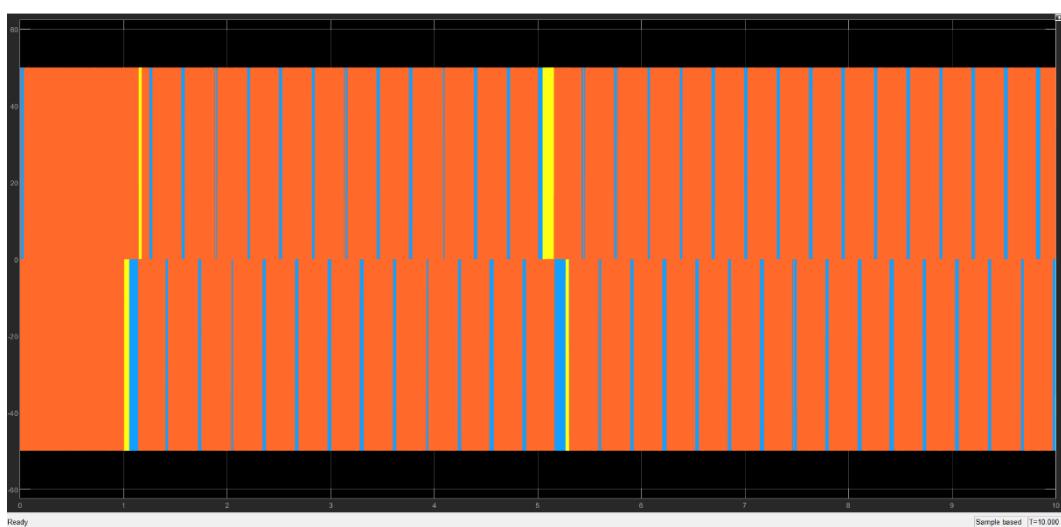
- **Ia,Ib,Ic**



- **Va,Vb,Vc**

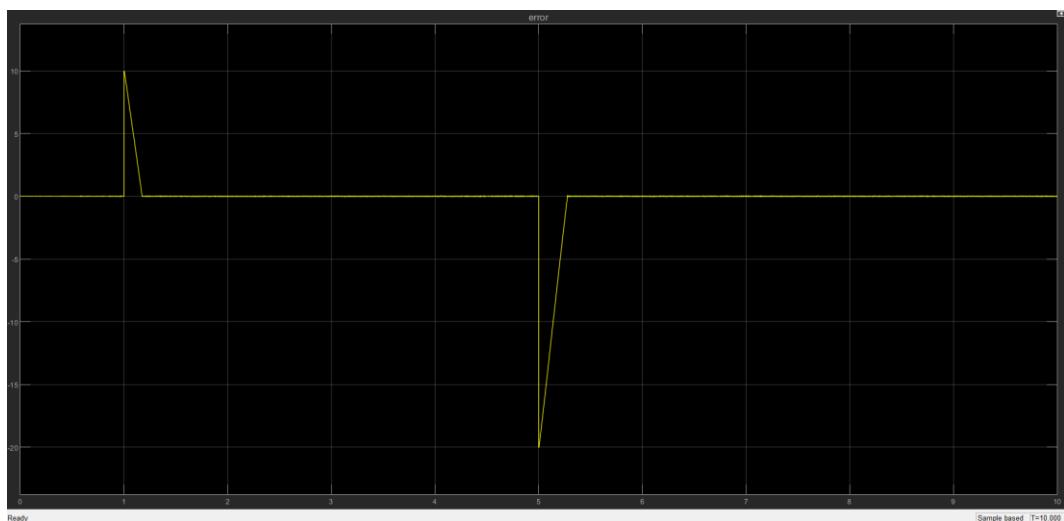


- **PWMa,PWMb,PWMcs**

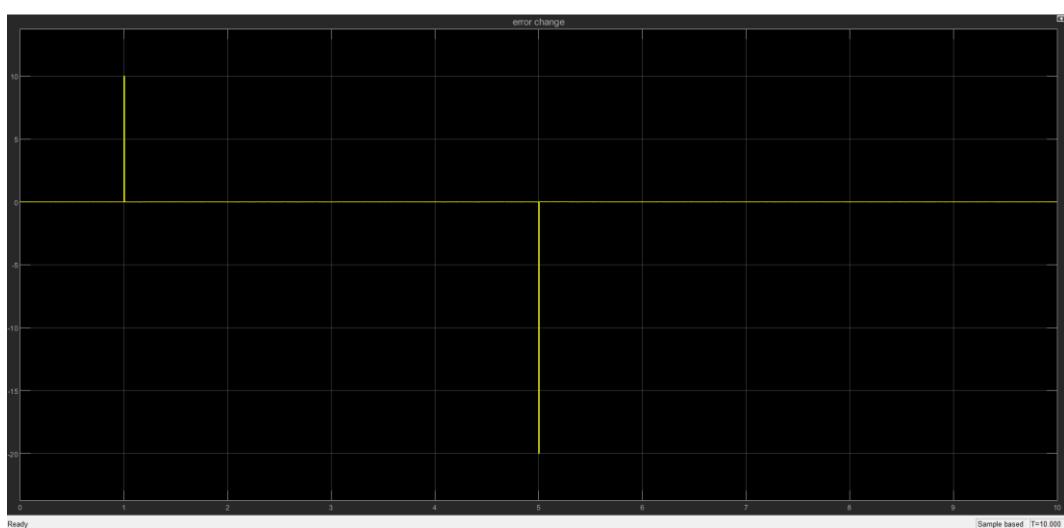


➤ **Rule Base Simplified 9**

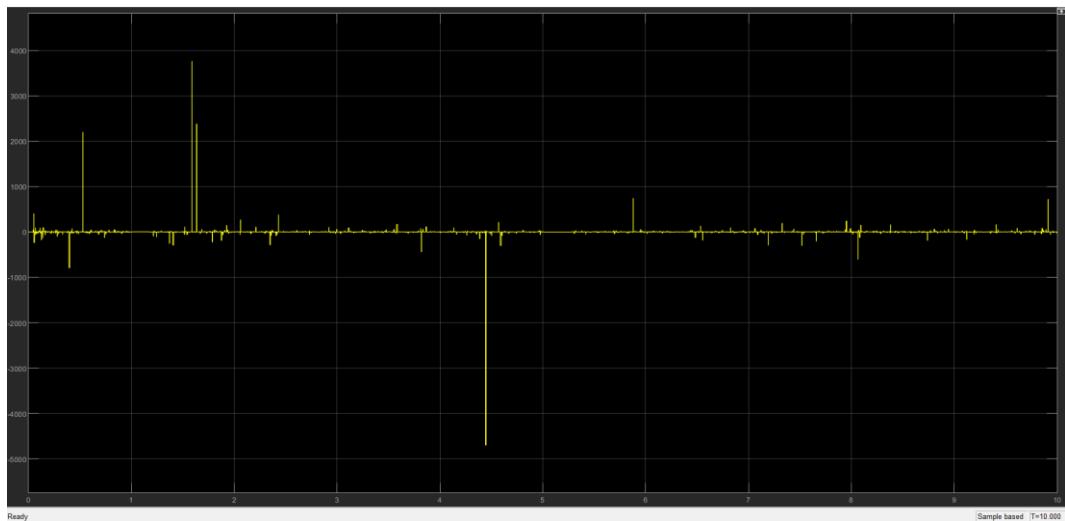
- **error FLC**



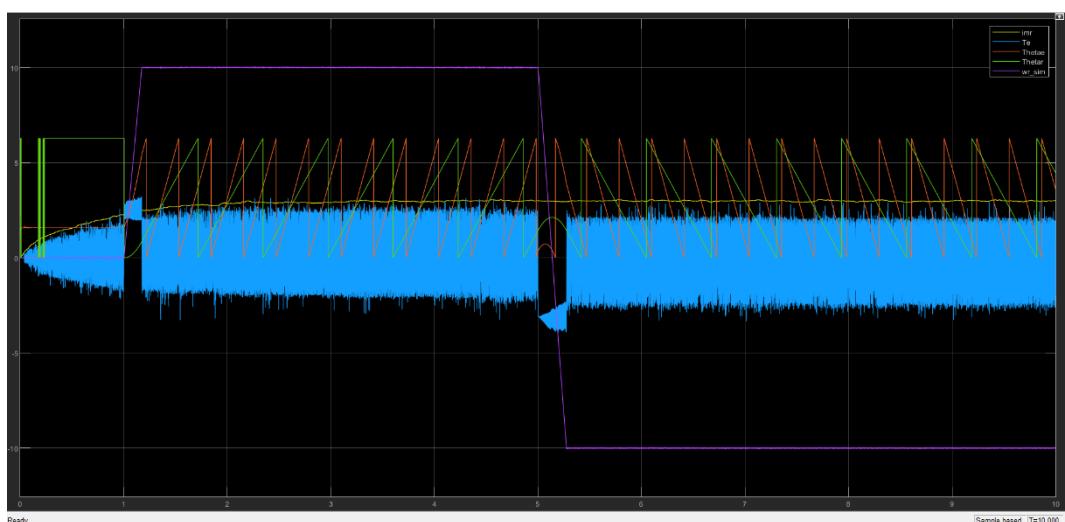
- **error change FLC**



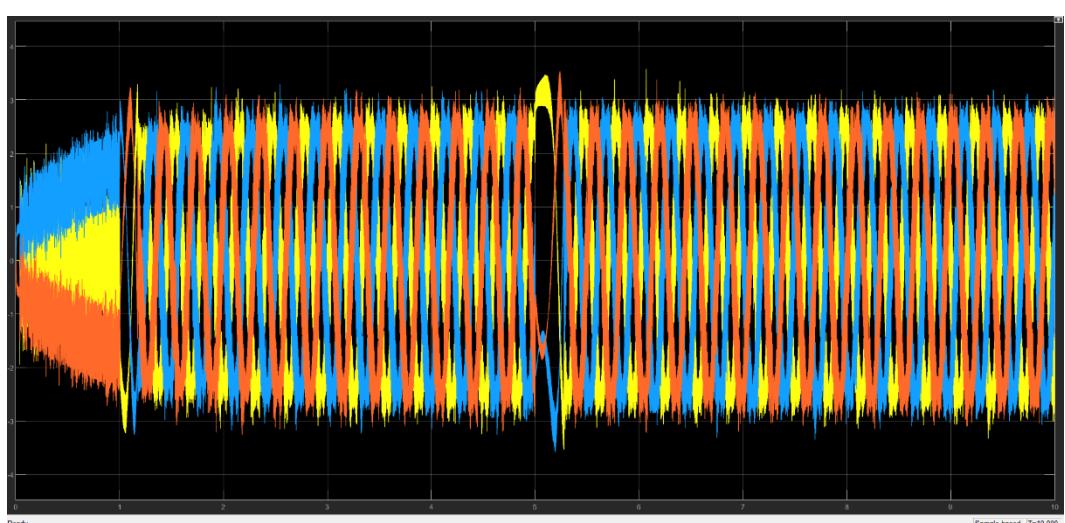
- **Iq ref**



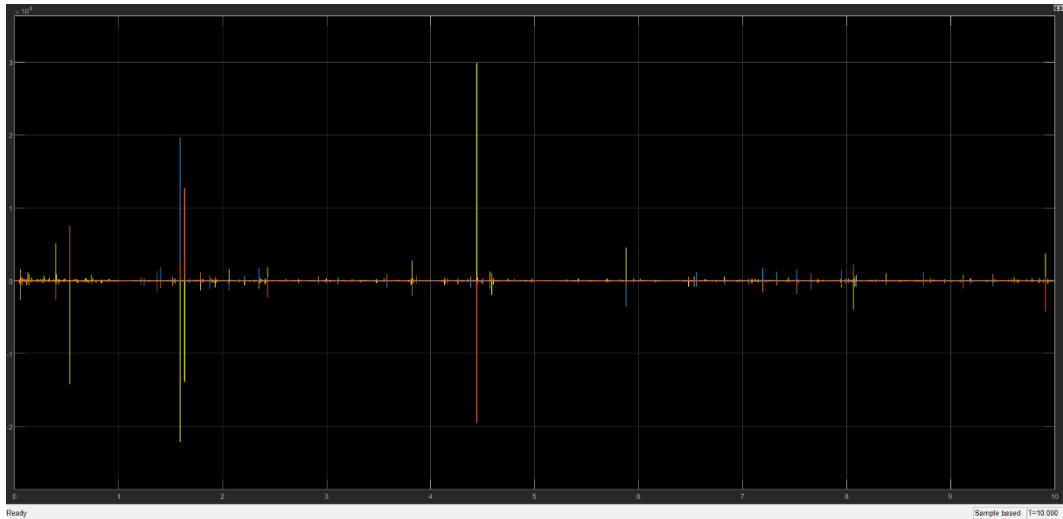
- **imr, Te, θe , θr .**



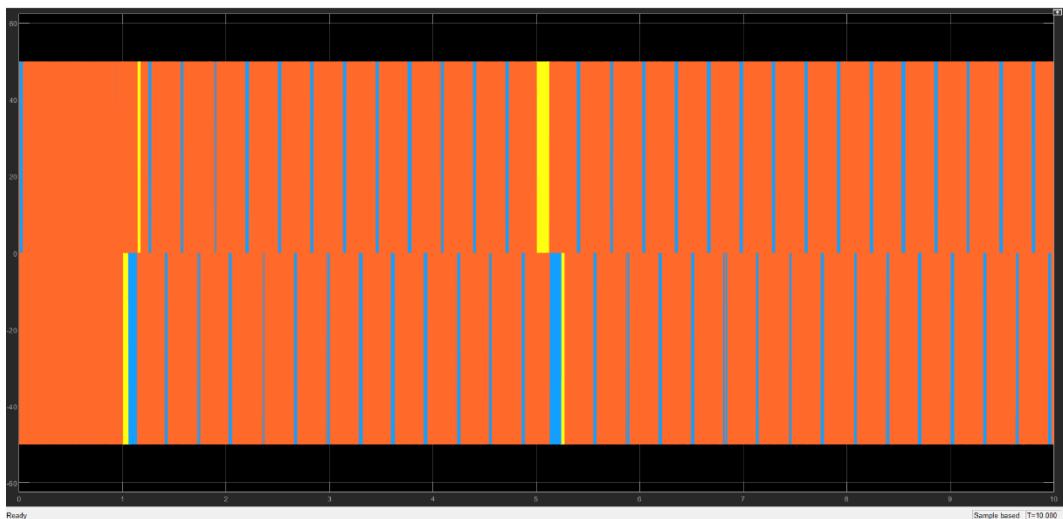
- **Ia, Ib, Ic**



- **V_a,V_b,V_c**



- **PWM_a,PWM_b,PWM_c**



3.4. Karakteristik Pengendali

Diperoleh hasil respon waktu dari masing – masing pengendali sebagai berikut:

Item	Membership Function					
	7x7		5x5		3x3	
OS(%)	49-rules	9-rules	25-rules	7-rules	9-rules	5-rules
OS(%)	0.529	0.325	0.505	0.505	0.3635	0.4523
T _s (s)	1.2432	1.2435	1.2264	1.2243	1.3884	1.3505
T _r (s)	1.1574	1.1572	1.125	1.2350	1.2264	1.1921

Hasil respon waktu tersebut kemudian dibandingkan dengan paper referensi. Berikut merupakan hasil respon waktu dari paper referensi.

Item	MFs	7x7		5x5		3x3	
		49-rules	9-rules	25-rules	7-rules	9-rules	5-rules
OS (%)		0.43	0.57	0.35	0.36	0.21	0.21
Ts (s)		0.16	0.18	0.17	0.18	0.18	0.18
Tr(s)		0.094	0.094	0.095	0.096	0.093	0.093

Secara garis besar hasil yang diperoleh antara FLC non simplifikasi dengan FLC simplifikasi sudah cukup sesuai dengan hasil dari paper referensi. Namun terdapat beberapa perbedaan sedikit yang ditandai oleh warna kuning pada tabel. Dimana pada paper referensi, hasil FLC pengendali non simplifikasi seharusnya memberikan hasil yang lebih baik dibandingkan hasil simplifikasi, namun pada keempat pasangan data tersebut, terdapat perbedaan hasil dimana hasil FLC simplifikasi memiliki hasil yang lebih baik dibandingkan FLC non simplifikasi.

Perbedaan ini dapat dihasilkan karena adanya perbedaan parameter yang digunakan pada simulasi dengan pada paper referensi dimana mereka tidak menyebutkan beberapa parameter yang digunakan. Parameter yang dapat menjadi berbeda dengan paper referensi yaitu : Scaling Factor FLC, Gain RFOC, Vdc dan Freq pada PWM serta parameter motor IM. Perbedaan hasil juga dapat dihasilkan oleh perbedaan metode FOC dan VSI yang digunakan oleh penulis paper dimana penulis tidak menyebutkan metode yang dia gunakan.

BAB 4

KESIMPULAN

Setelah melakukan simulasi dari peforma pengendalian motor induksi dengan rule base hasil simplifikasi dengan non simplifikasi, diperoleh kesimpulan bahwa Fuzzy Logic Controller sebagai pengendali non konvensional memiliki kemampuan untuk mengimitasi cara berfikir manusia sehingga cocok untuk diterapkan pada sistem non linear yang dimana susah untuk dimodelkan secara matematis. Pada sistem penggerak motor induksi, FLC dapat diterapkan sebagai pengendali kecepatan dengan masukan adalah nilai selisih dari nilai kecepatan referensi dengan nilai kecepatan aktual. Rule base dari FLC dapat disintesa dengan menganalisa respon step dinamis dari motor induksi. Rule base tersebut juga dapat disimplifikasi dengan cara menggunakan konsep mencari rute terdekat untuk mencapai titik equilibrium. Hasil dari simulasi menunjukkan bahwa rule base hasil sintesa dapat digunakan untuk pengendalian motor induksi. Simulasi juga menunjukkan bahwa rule base hasil simflikasi dari rule base hasil sintesa dapat digunakan untuk pengendalian motor induksi serta memberikan hasil yang tidak jauh berbeda. Simplifikasi ini menghasilkan FLC yang mempunyai beban komputasi lebih rendah sehingga lebih mudah untuk diimplementasikan. Metode sintesa rule base FLC serta simplifikasinya dengan menganalisa respon step sistem dapat diterapkan pada sistem orde kedua yang serupa.

DAFTAR PUSTAKA

- [1] J. Janfzen, “Tuning-rules for fuzzy controllers,” *Proc. IEEE Int. Work. Intell. Motion Control. IMC 1990*, vol. 1, pp. 83–86, 1990, doi: 10.1109/IMC.1990.687300.
- [2] B. G. Lamme, “The story of the induction motor,” *J. Am. Inst. Electr. Eng.*, vol. 40, no. 3, pp. 203–223, 2013, doi: 10.1109/jaiee.1921.6592844.
- [3] T. Pwm, “The PWM Control of the Three-phase Induction Motor Ping Wei, jinpeng Yu, Fatao Shi, Xiao Wei, Yan Wang, Quanwen Zhao,” no. Icmse, pp. 842–845, 2015.
- [4] S. Reddy and M. S. Aspalli, “Speed Control of Three Phase Induction Motor using Digital Signal Controller,” *IJSRD-International J. Sci. Res. Dev.*, vol. 4, no. 12, pp. 2321–0613, 2017, [Online]. Available: www.ijsr.com.
- [5] S. Wadhwani, “Speed Control of Separately Excited Dc Motor Using Fuzzy Logic Controller,” vol. 4, no. June, pp. 2518–2523, 2013.
- [6] Y. Satyanarayana, “Speed Control of Induction Motor using Fuzzy PI Controller Based on Space Vector Pulse Width Modulation,” *Int. J. Comput. Eng. Res.*, vol. 2, no. 5, pp. 2250–3005, 2012.
- [7] V. A. Arun, N. S. Nair, and I. K. Simon, “Simulation And Implementation Of Ifoc Based 3-Phase Induction Motor Drive,” pp. 60–68, 2016.
- [8] “Research Papers Speed Control of Induction Motor Using,” vol. 8, no. 2, pp. 21–31, 2014.
- [9] K. A. El-Metwally and O. P. Malik, “Parameter Tuning for a Fuzzy Logic Controller,” *IFAC Proc. Vol.*, vol. 26, no. 2, pp. 581–584, 1993, doi: 10.1016/s1474-6670(17)49009-4.
- [10] D. Uma and K. Vijayarekha, “Modeling and simulation of VSI fed induction motor drive in Matlab/Simulink,” *Int. J. Electr. Comput. Eng.*, vol. 7, no. 2, pp. 584–595, 2017, doi: 10.11591/ijece.v7i2.pp584-595.
- [11] B. M. Mohan and A. Sinha, *Mathematical Model of the Simplest Fuzzy PID Controller with Asymmetric Fuzzy Sets*, vol. 41, no. 2. IFAC, 2008.
- [12] G. Nhivekar, S. Nirmale, and R. Mudholker, “Implementation of fuzzy logic control algorithm in embedded microcomputers for dedicated application,” *Int. J. Eng. Sci. Technol.*, vol. 3, no. 4, pp. 276–283, 2011, doi: 10.4314/ijest.v3i4.68559.
- [13] M. A. Mannan, A. Islam, M. N. Uddin, M. K. Hassan, T. Murata, and J. Tamura, “Fuzzy-Logic Based Speed Control of Induction Motor Considering Core Loss into Account,” *Intell. Control Autom.*, vol. 03, no. 03, pp. 229–235, 2012, doi: 10.4236/ica.2012.33026.

- [14] R. M. Ariff *et al.*, “Fuzzy logic control design for induction motor speed control improvement through field oriented control,” *Lect. Notes Electr. Eng.*, vol. 253 LNEE, no. June, pp. 273–280, 2013, doi: 10.1007/978-94-007-6996-0_29.
- [15] B. M. Badr, A. M. Eltamaly, and A. I. Alolah, “Fuzzy controller for three phases induction motor drives,” *2010 IEEE Veh. Power Propuls. Conf. VPPC 2010*, no. 1, 2010, doi: 10.1109/VPPC.2010.5729080.
- [16] F. Wahab, A. Sumardiono, A. R. Al Tahtawi, and A. F. A. Mulayari, “Desain dan Purwarupa Fuzzy Logic Control untuk Pengendalian Suhu Ruangan,” *J. Teknol. Rekayasa*, vol. 2, no. 1, p. 1, 2017, doi: 10.31544/jtera.v2.i1.2017.1-8.
- [17] M. Deb, “Control of Voltage Source Inverter for Adjustable Speed Drive- a Study Report,” vol. 1, no. 1, pp. 1–3, 2013.
- [18] S. F. It, “Chapter4 Model of Three-Phase Inverter,” pp. 58–80, [Online]. Available: <https://www.tntech.edu/files/cesr/StudThesis/asuri/Chapter4.pdf>.
- [19] D. N. Dewangan, M. Jha, and M. F. Qureshi, “A study on parameter tuning of weighted fuzzy rule base using genetic algorithm for steam turbine model,” *Adv. Model. Anal. B*, vol. 55, no. 1–2, pp. 1–19, 2012.
- [20] E.-A. P. J., “A Novel Design and Tuning Procedure for PID Type Fuzzy Logic Controllers.” 2002.
- [21] A. Mishra, G. Dubey, D. Joshi, P. Agarwal, and S. P. Sriavstava, “A complete fuzzy logic based real-time simulation of vector controlled PMSM drive,” *2018 2nd IEEE Int. Conf. Power Electron. Intell. Control Energy Syst. ICPEICES 2018*, pp. 809–814, 2018, doi: 10.1109/ICPEICES.2018.8897373.

LAMPIRAN KODE SUMBER

FUZZY9.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY9
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDIT IONS
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);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    //DECLARE VARIABEL
    real_T iq_ref, e, ec;

    //AMBIL DARI MEMORI
    iq_ref = X[0];
    e = X[1];
    ec = X[2];

    //OUTPUT
    Y[0] = iq_ref;
    Y[1] = e;
    Y[2] = ec;
}

#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;

    //DECLARE VARIABEL
    real_T e, ec, e_prev, w_ref, w_act;
    real_T nl, nm, ns, z, ps, pm, pl;
    real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
    real_T dom_xh, dom_xl, dom_yh, dom_yl;
    real_T rb1,rb2,rb3,rb4;
    real_T rb_compare_low,rb_compare_high;
    real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
    real_T iq_ref;
    real_T gmf = 4.1;

    //INPUT
    e_prev = X[1];
    w_ref = U(0);
    w_act = U(1);

    //HITUNG ERROR DAN CHANGE OF ERROR
    e = (w_ref - w_act);
    ec = (e - e_prev);
}

```

```

//MEMBERSHIP FUNCTION
pl = 0.5*gmf;
z = 0.0*gmf;
nl = -0.5*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E
if(x <= nl){
    xl = z;
    xh = nl;
    a = 0;
    b = 1;
}
else if(nl < x && x < pl){
    if(z <= x < pl){
        xl = z;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x < z){
        xl = z;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
else if(x >= pl){
    xl = z;
    xh = pl;
    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = z;
    yh = nl;
    c = 0;
    d = 1;
}

```

```

else if(nl < y && y < pl){
    if(z <= y < pl){
        yl = z;
        yh = pl;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(nl < y < z){
        yl = z;
        yh = nl;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(y >= pl){
    yl = z;
    yh = pl;
    c = 0;
    d = 1;
}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==nl){rb_compare_low = z;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==nl){rb_compare_high = z;}
}

if(xl==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==nl){rb_compare_low = z;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==nl){rb_compare_high = z;}
}

```

```

if(xl==z){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = z;}
    if (yh==nl){rb_compare_high = nl;}
}

if(xl==ns){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = nl;}
}

if(xl==nl){
    if (yl==pl){rb_compare_low = z;}
    if (yl==z){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = z;}
    if (yh==z){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}
}

rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==nl){rb_compare_low = z; }

    if (yh==pl){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==nl){rb_compare_high = z; }
}

if(xh==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==nl){rb_compare_low = z; }
}

```

```

        if (yh==pl){rb_compare_high = pl;}
        if (yh==z){rb_compare_high = pl;}
        if (yh==nl){rb_compare_high = z;}
    }

    if(xh==z){
        if (yl==pl){rb_compare_low = pl;}
        if (yl==z){rb_compare_low = z;}
        if (yl==nl){rb_compare_low = nl;}

        if (yh==pl){rb_compare_high = pl;}
        if (yh==z){rb_compare_high = z;}
        if (yh==nl){rb_compare_high = nl;}
    }

    if(xh==ns){
        if (yl==pl){rb_compare_low = ps;}
        if (yl==z){rb_compare_low = ns;}
        if (yl==nl){rb_compare_low = nl;}


        if (yh==pl){rb_compare_high = ps;}
        if (yh==z){rb_compare_high = ns;}
        if (yh==nl){rb_compare_high = nl;}
    }

    if(xh==nl){
        if (yl==pl){rb_compare_low = z;}
        if (yl==z){rb_compare_low = nl;}
        if (yl==nl){rb_compare_low = nl;}


        if (yh==pl){rb_compare_high = z;}
        if (yh==z){rb_compare_high = nl;}
        if (yh==nl){rb_compare_high = nl;}
    }
    rb3 = rb_compare_low;
    rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){
    mag_infer_1 = dom_xh;
}
else {
    mag_infer_1 = dom_yh;
}

if(dom_xh < dom_yl){
    mag_infer_2 = dom_xh;
}

```

```

    }
    else {
        mag_infer_2 = dom_yl;
    }

    if(dom_xl < dom_yh){
        mag_infer_3 = dom_xl;
    }
    else {
        mag_infer_3 = dom_yh;
    }

    if(dom_xl < dom_yl){
        mag_infer_4 = dom_xl;
    }
    else {
        mag_infer_4 = dom_yl;
    }

    //Perhitungan MAX
    //Max Layer
    if(rb1 == rb2){
        if(mag_infer_1 < mag_infer_2){
            mag_infer_1 = 0.0;
        }
        else{
            mag_infer_2 = 0.0;
        }
    }

    else if(rb1 == rb3){
        if(mag_infer_1 < mag_infer_3){
            mag_infer_1 = 0.0;
        }
        else{
            mag_infer_3 = 0.0;
        }
    }

    else if(rb1 == rb4){
        if(mag_infer_1 < mag_infer_4){
            mag_infer_1 = 0.0;
        }
        else{
            mag_infer_4 = 0.0;
        }
    }

    else if(rb2 == rb3){

```

```

        if(mag_infer_2 < mag_infer_3){
            mag_infer_2 = 0.0;
        }
        else{
            mag_infer_3 = 0.0;
        }
    }

    else if(rb2 == rb4){
        if(mag_infer_2 < mag_infer_4){
            mag_infer_2 = 0.0;
        }
        else{
            mag_infer_4 = 0.0;
        }
    }

    else if(rb3 == rb4){
        if(mag_infer_3 < mag_infer_4){
            mag_infer_3 = 0.0;
        }
        else{
            mag_infer_4 = 0.0;
        }
    }

}

//DEFUZZIFICATION
iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

X[0] = iq_ref;
X[1] = e;
X[2] = ec;
}

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

#ifndef 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

```

FUZZY25.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY25
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDIT IONS
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);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

//DECLARE VARIABEL
real_T iq_ref, e, ec;

//AMBIL DARI MEMORI
iq_ref = X[0];
e = X[1];
ec = X[2];

//OUTPUT
Y[0] = iq_ref;
Y[1] = e;
Y[2] = ec;
}

#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;

    //DECLARE VARIABEL
    real_T e, ec, e_prev, w_ref, w_act;
    real_T nl, nm, ns, z, ps, pm, pl;
    real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
    real_T dom_xh, dom_xl, dom_yh, dom_yl;
    real_T rb1,rb2,rb3,rb4;
    real_T rb_compare_low,rb_compare_high;
    real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
    real_T iq_ref;
    real_T gmf = 4.1;

    //INPUT
    e_prev = X[1];
    w_ref = U(0);
    w_act = U(1);

    //HITUNG ERROR DAN CHANGE OF ERROR
    e = (w_ref - w_act);
    ec = (e - e_prev);

    //MEMBERSHIP FUNCTION
    pl = 1.0*gmf;
}

```

```

ps = 0.5*gmf;
z = 0.0*gmf;
ns = -0.5*gmf;
nl = -1.0*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E
if(x <= nl){
    xl = ns;
    xh = nl;
    a = 0;
    b = 1;
}
else if(nl < x && x < pl){
    if(z <= x < ps){
        xl = z;
        xh = ps;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ps <= x < pl){
        xl = ps;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ns < x < z){
        xl = z;
        xh = ns;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x <= ns){
        xl = ns;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
else if(x >= pl){
    xl = ps;
    xh = pl;
}

```

```

    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = ns;
    yh = nl;
    c = 0;
    d = 1;
}
else if(nl < y && y < pl){
    if(z <= y < ps){
        yl = z;
        yh = ps;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ps <= y < pl){
        yl = ps;
        yh = pl;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ns < y < z){
        yl = z;
        yh = ns;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(nl < y <= ns){
        yl = ns;
        yh = nl;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(y >= pl){
    yl = ps;
    yh = pl;
    c = 0;
    d = 1;
}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

```

```

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}
}

if(xl==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}
}

if(xl==z){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = nl; }

    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = z;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = nl;}
}

if(xl==ns){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==ps){rb_compare_low = z; }
}

```

```

    if (yl==z){rb_compare_low = ns;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = ps;}
    if (yh==ps){rb_compare_high = z;}
    if (yh==z){rb_compare_high = ns;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = nl;}

}
if(xl==nl){
    if (yl==pl){rb_compare_low = z;}
    if (yl==ps){rb_compare_low = ns;}
    if (yl==z){rb_compare_low = nl;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = z;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nl;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}

}
rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}

}

if(xh==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}


}

```

```

    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}

}

if(xh==z){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = z;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = nl;}


}

if(xh==ns){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==ps){rb_compare_low = z;}
    if (yl==z){rb_compare_low = ns;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = ps;}
    if (yh==ps){rb_compare_high = z;}
    if (yh==z){rb_compare_high = ns;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = nl;}


}

if(xh==nl){
    if (yl==pl){rb_compare_low = z;}
    if (yl==ps){rb_compare_low = ns;}
    if (yl==z){rb_compare_low = nl;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = z;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nl;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}}

```

```

    }

rb3 = rb_compare_low;
rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){
    mag_infer_1 = dom_xh;
}
else {
    mag_infer_1 = dom_yh;
}

if(dom_xh < dom_yl){
    mag_infer_2 = dom_xh;
}
else {
    mag_infer_2 = dom_yl;
}

if(dom_xl < dom_yh){
    mag_infer_3 = dom_xl;
}
else {
    mag_infer_3 = dom_yh;
}

if(dom_xl < dom_yl){
    mag_infer_4 = dom_xl;
}
else {
    mag_infer_4 = dom_yl;
}

//Perhitungan MAX
//Max Layer
if(rb1 == rb2){
    if(mag_infer_1 < mag_infer_2){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_2 = 0.0;
    }
}

else if(rb1 == rb3){
    if(mag_infer_1 < mag_infer_3){
        mag_infer_1 = 0.0;
    }
}

```

```

        else{
            mag_infer_3 = 0.0;
        }
    }

else if(rb1 == rb4){
    if(mag_infer_1 < mag_infer_4){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb2 == rb3){
    if(mag_infer_2 < mag_infer_3){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb2 == rb4){
    if(mag_infer_2 < mag_infer_4){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb3 == rb4){
    if(mag_infer_3 < mag_infer_4){
        mag_infer_3 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

//DEFUZZYFICATION
iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

X[0] = iq_ref;
X[1] = e;
X[2] = ec;

```

```

}

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

#ifndef 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

```

FUZZY49.c

```

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY49
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDIT IONS
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);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    //DECLARE VARIABEL
    real_T iq_ref, e, ec;

    //AMBIL DARI MEMORI
    iq_ref = X[0];
    e = X[1];
    ec = X[2];

    //OUTPUT
    Y[0] = iq_ref;
    Y[1] = e;
    Y[2] = ec;
}

#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;

    //DECLARE VARIABEL
    real_T e, ec, e_prev, w_ref, w_act;
    real_T nl, nm, ns, z, ps, pm, pl;
    real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
    real_T dom_xh, dom_xl, dom_yh, dom_yl;
    real_T rb1,rb2,rb3,rb4;
    real_T rb_compare_low,rb_compare_high;
    real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
    real_T iq_ref;
}

```

```

real_T gmf = 4.1;

//INPUT
e_prev = X[1];
w_ref = U(0);
w_act = U(1);

//HITUNG ERROR DAN CHANGE OF ERROR
e = (w_ref - w_act);
ec = (e - e_prev);

//MEMBERSHIP FUNCTION
pl = 0.75*gmf;
pm = 0.5*gmf;
ps = 0.25*gmf;
z = 0.0*gmf;
ns = -0.25*gmf;
nm = -0.5*gmf;
nl = -0.75*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E

if(x <= nl){
    xl = nm;
    xh = nl;
    a = 0;
    b = 1;
}

else if(nl < x && x < pl){
    if(z <= x < ps){
        xl = z;
        xh = ps;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ps <= x < pm){
        xl = ps;
        xh = pm;
        b = (x-xl)*se;
    }
}

```

```

        a = 1-b;
    }
    else if(pm <= x < pl){
        xl = pm;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ns < x < z){
        xl = z;
        xh = ns;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nm < x <= ns){
        xl = nm;
        xh = nm;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x <= nm){
        xl = nm;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
}

else if(x >= pl){
    xl = pm;
    xh = pl;
    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = nm;
    yh = nl;
    c = 0;
    d = 1;
}

else if(nl < y && y < pl){
    if(z <= y < ps){
        yl = z;
        yh = ps;
        d = (y-yl)*se;
    }
}

```

```

        c = 1-d;
    }
    else if(ps <= y < pm){
        yl = ps;
        yh = pm;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(pm <= y < pl){
        yl = pm;
        yh = pl;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ns < y < z){
        yl = z;
        yh = ns;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(nm < y <= ns){
        yl = ns;
        yh = nm;
        d = (x-xl)*se;
        c = 1-d;
    }
    else if(nl < y <= nm){
        yl = nm;
        yh = nl;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(y >= pl){
    yl = pm;
    yh = pl;
    c = 0;
    d = 1;
}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

```

```

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nm){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nm){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}


}

if(xl==pm){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pm;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nm){rb_compare_low = z;}
    if (yl==nl){rb_compare_low = ns;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pm;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nm){rb_compare_high = z;}
    if (yh==nl){rb_compare_high = ns;}


}

if(xl==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = ps;}
    if (yl==ns){rb_compare_low = z;}
    if (yl==nm){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = ns;}


    if (yh==pl){rb_compare_high = pl;}


}

```

```

    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = ps;}
    if (yh==ns){rb_compare_high = z;}
    if (yh==nm){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = ns;}
}

if(xl==z){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pm;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nm){rb_compare_low = nm;}
    if (yl==nl){rb_compare_low = nl;}
}

if (yh==pl){rb_compare_high = pl;}
if (yh==pm){rb_compare_high = pm;}
if (yh==ps){rb_compare_high = ps;}
if (yh==z){rb_compare_high = z;}
if (yh==ns){rb_compare_high = ns;}
if (yh==nm){rb_compare_high = nm;}
if (yh==nl){rb_compare_high = nl;}


if(xl==ns){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==pm){rb_compare_low = ps;}
    if (yl==ps){rb_compare_low = z;}
    if (yl==z){rb_compare_low = ns;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = ps;}
    if (yh==pm){rb_compare_high = ps;}
    if (yh==ps){rb_compare_high = z;}
    if (yh==z){rb_compare_high = ns;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}


}

if(xl==nm){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==pm){rb_compare_low = z;}
    if (yl==ps){rb_compare_low = ns;}}

```

```

    if (yl==z){rb_compare_low = nm;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = ps;}
    if (yh==pm){rb_compare_high = z;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nm;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}

}

if(xl==nl){
    if (yl==pl){rb_compare_low = z;}
    if (yl==pm){rb_compare_low = ns;}
    if (yl==ps){rb_compare_low = ns;}
    if (yl==z){rb_compare_low = nl;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = z;}
    if (yh==pm){rb_compare_high = ns;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nl;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}

}
rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pl;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nm){rb_compare_low = ps;}
    if (yl==nl){rb_compare_low = z;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pl;}


}

```

```

    if (yh==ns){rb_compare_high = ps;}
    if (yh==nm){rb_compare_high = ps;}
    if (yh==nl){rb_compare_high = z;}
}

if(xh==pm){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = pl;}
    if (yl==z){rb_compare_low = pm;}
    if (yl==ns){rb_compare_low = ps;}
    if (yl==nm){rb_compare_low = z;}
    if (yl==nl){rb_compare_low = ns;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = pl;}
    if (yh==z){rb_compare_high = pm;}
    if (yh==ns){rb_compare_high = ps;}
    if (yh==nm){rb_compare_high = z;}
    if (yh==nl){rb_compare_high = ns;}


}

if(xh==ps){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pl;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = ps;}
    if (yl==ns){rb_compare_low = z;}
    if (yl==nm){rb_compare_low = ns;}
    if (yl==nl){rb_compare_low = ns;}


    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pl;}
    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = ps;}
    if (yh==ns){rb_compare_high = z;}
    if (yh==nm){rb_compare_high = ns;}
    if (yh==nl){rb_compare_high = ns;}


}

if(xh==z){
    if (yl==pl){rb_compare_low = pl;}
    if (yl==pm){rb_compare_low = pm;}
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nm){rb_compare_low = nm;}}

```

```

    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = pl;}
    if (yh==pm){rb_compare_high = pm;}
    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = z;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nm){rb_compare_high = nm;}
    if (yh==nl){rb_compare_high = nl;}

}

if(xh==ns){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==pm){rb_compare_low = ps;}
    if (yl==ps){rb_compare_low = z;}
    if (yl==z){rb_compare_low = ns;}
    if (yl==ns){rb_compare_low = ns;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = ps;}
    if (yh==pm){rb_compare_high = ps;}
    if (yh==ps){rb_compare_high = z;}
    if (yh==z){rb_compare_high = ns;}
    if (yh==ns){rb_compare_high = ns;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}


}

if(xh==nm){
    if (yl==pl){rb_compare_low = ps;}
    if (yl==pm){rb_compare_low = z;}
    if (yl==ps){rb_compare_low = ns;}
    if (yl==z){rb_compare_low = nm;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}


    if (yh==pl){rb_compare_high = ps;}
    if (yh==pm){rb_compare_high = z;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nm;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}


}

if(xh==nl){

```

```

    if (yl==pl){rb_compare_low = z;}
    if (yl==pm){rb_compare_low = ns;}
    if (yl==ps){rb_compare_low = ns;}
    if (yl==z){rb_compare_low = nl;}
    if (yl==ns){rb_compare_low = nl;}
    if (yl==nm){rb_compare_low = nl;}
    if (yl==nl){rb_compare_low = nl;}

    if (yh==pl){rb_compare_high = z;}
    if (yh==pm){rb_compare_high = ns;}
    if (yh==ps){rb_compare_high = ns;}
    if (yh==z){rb_compare_high = nl;}
    if (yh==ns){rb_compare_high = nl;}
    if (yh==nm){rb_compare_high = nl;}
    if (yh==nl){rb_compare_high = nl;}

}
rb3 = rb_compare_low;
rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){
    mag_infer_1 = dom_xh;
}
else {
    mag_infer_1 = dom_yh;
}

if(dom_xh < dom_yl){
    mag_infer_2 = dom_xh;
}
else {
    mag_infer_2 = dom_yl;
}

if(dom_xl < dom_yh){
    mag_infer_3 = dom_xl;
}
else {
    mag_infer_3 = dom_yh;
}

if(dom_xl < dom_yl){
    mag_infer_4 = dom_xl;
}
else {
    mag_infer_4 = dom_yl;
}

```

```
//Perhitungan MAX
//Max Layer
if(rb1 == rb2){
    if(mag_infer_1 < mag_infer_2){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_2 = 0.0;
    }
}

else if(rb1 == rb3){
    if(mag_infer_1 < mag_infer_3){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb1 == rb4){
    if(mag_infer_1 < mag_infer_4){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb2 == rb3){
    if(mag_infer_2 < mag_infer_3){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb2 == rb4){
    if(mag_infer_2 < mag_infer_4){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb3 == rb4){
```

```

        if(mag_infer_3 < mag_infer_4){
            mag_infer_3 = 0.0;
        }
        else{
            mag_infer_4 = 0.0;
        }
    }

    //DEFUZZIFICATION
    iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

    X[0] = iq_ref;
    X[1] = e;
    X[2] = ec;
}

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

#ifndef 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

```

FUZZY5SIM.c

```

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY5SIM
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDIT    IONS
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);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    //DECLARE VARIABEL
    real_T iq_ref, e, ec;

    //AMBIL DARI MEMORI
    iq_ref = X[0];
    e = X[1];
    ec = X[2];

    //OUTPUT
    Y[0] = iq_ref;
    Y[1] = e;
    Y[2] = ec;
}

#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;

//DECLARE VARIABLE
real_T e, ec, e_prev, w_ref, w_act;
real_T nl, nm, ns, z, ps, pm, pl;
real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
real_T dom_xh, dom_xl, dom_yh, dom_yl;
real_T rb1,rb2,rb3,rb4;
real_T rb_compare_low,rb_compare_high;
real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
real_T iq_ref;
real_T gmf = 4.1;

//INPUT
e_prev = X[1];
w_ref = U(0);
w_act = U(1);

//HITUNG ERROR DAN CHANGE OF ERROR
e = (w_ref - w_act);
ec = (e - e_prev);

//MEMBERSHIP FUNCTION
pl = 1.0*gmf;
ps = 0.5*gmf;
z = 0.0*gmf;
ns = -0.5*gmf;
nl = -1.0*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E
if(x <= nl){
    xl = ns;
    xh = nl;
    a = 0;
    b = 1;
}
else if(nl < x && x < pl){

}

```

```

    if(z <= x < ps){
        xl = z;
        xh = ps;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ps <= x < pl){
        xl = ps;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ns < x < z){
        xl = z;
        xh = ns;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x <= ns){
        xl = ns;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
else if(x >= pl){
    xl = ps;
    xh = pl;
    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = ns;
    yh = nl;
    c = 0;
    d = 1;
}
else if(nl < y && y < pl){
    if(z <= y < ps){
        yl = z;
        yh = ps;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ps <= y < pl){
        yl = ps;
    }
}

```

```

        yh = pl;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ns < y <z){
        yl = z;
        yh = ns;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(nl < y <= ns){
        yl = ns;
        yh = nl;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(y >= pl){
    yl = ps;
    yh = pl;
    c = 0;
    d = 1;
}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==z){rb_compare_low = pl;}
    if (yh==z){rb_compare_high = pl;}
}

if(xl==ps){
    if (yl==z){rb_compare_low = pl;}
    if (yh==z){rb_compare_high = pl;}
}

if(xl==z){
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}
}

```

```

        if (yh==ps){rb_compare_high = ps;}
        if (yh==z){rb_compare_high = z;}
        if (yh==ns){rb_compare_high = ns;}
    }

    if(xl==ns){
        if (yl==z){rb_compare_low = ns;}

        if (yh==z){rb_compare_high = ns;}
    }

    if(xl==nl){
        if (yl==z){rb_compare_low = nl;}

        if (yh==z){rb_compare_high = nl;}
    }
rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==z){rb_compare_low = pl;}

    if (yh==z){rb_compare_high = pl;}
}

if(xh==ps){
    if (yl==z){rb_compare_low = pl;}

    if (yh==z){rb_compare_high = pl;}
}

if(xh==z){
    if (yl==ps){rb_compare_low = ps;}
    if (yl==z){rb_compare_low = z;}
    if (yl==ns){rb_compare_low = ns;}


    if (yh==ps){rb_compare_high = ps;}
    if (yh==z){rb_compare_high = z;}
    if (yh==ns){rb_compare_high = ns;}
}

if(xh==ns){
    if (yl==z){rb_compare_low = ns;}


    if (yh==z){rb_compare_high = ns;}
}

```

```

if(xh==n1){
    if (yl==z){rb_compare_low = n1; }

    if (yh==z){rb_compare_high = n1;}
}
rb3 = rb_compare_low;
rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){
    mag_infer_1 = dom_xh;
}
else {
    mag_infer_1 = dom_yh;
}

if(dom_xh < dom_yl){
    mag_infer_2 = dom_xh;
}
else {
    mag_infer_2 = dom_yl;
}

if(dom_xl < dom_yh){
    mag_infer_3 = dom_xl;
}
else {
    mag_infer_3 = dom_yh;
}

if(dom_xl < dom_yl){
    mag_infer_4 = dom_xl;
}
else {
    mag_infer_4 = dom_yl;
}

//Perhitungan MAX
//Max Layer
if(rb1 == rb2){
    if(mag_infer_1 < mag_infer_2){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_2 = 0.0;
    }
}

```

```

else if(rb1 == rb3){
    if(mag_infer_1 < mag_infer_3){
        mag_infer_1 = 0.0;
    }
}

else if(rb1 == rb4){
    if(mag_infer_1 < mag_infer_4){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb2 == rb3){
    if(mag_infer_2 < mag_infer_3){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb2 == rb4){
    if(mag_infer_2 < mag_infer_4){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb3 == rb4){
    if(mag_infer_3 < mag_infer_4){
        mag_infer_3 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

//DEFUZZIFICATION
iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

```

```

        X[0] = iq_ref;
        X[1] = e;
        X[2] = ec;
    }

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

```

FUZZY7SIM.c

```

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY7SIM
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

```

```

#define MDL_INITIALIZE_CONDIT    IONS
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);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    //DECLARE VARIABEL
    real_T iq_ref, e, ec;

    //AMBIL DARI MEMORI
    iq_ref = X[0];
    e = X[1];
    ec = X[2];

    //OUTPUT
    Y[0] = iq_ref;
    Y[1] = e;
    Y[2] = ec;
}

#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;

    //DECLARE VARIABEL
    real_T e, ec, e_prev, w_ref, w_act;
    real_T nl, nm, ns, z, ps, pm, pl;
    real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
    real_T dom_xh, dom_xl, dom_yh, dom_yl;
}

```

```

real_T rb1,rb2,rb3,rb4;
real_T rb_compare_low,rb_compare_high;
real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
real_T iq_ref;
real_T gmf = 4.1;

//INPUT
e_prev = X[1];
w_ref = U(0);
w_act = U(1);

//HITUNG ERROR DAN CHANGE OF ERROR
e = (w_ref - w_act);
ec = (e - e_prev);

//MEMBERSHIP FUNCTION
pl = 1.0*gmf;
ps = 0.5*gmf;
z = 0.0*gmf;
ns = -0.5*gmf;
nl = -1.0*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E
if(x <= nl){
    xl = ns;
    xh = nl;
    a = 0;
    b = 1;
}
else if(nl < x && x < pl){
    if(z <= x < ps){
        xl = z;
        xh = ps;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ps <= x < pl){
        xl = ps;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
}

```

```

    }
    else if(ns < x < z){
        xl = z;
        xh = ns;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x <= ns){
        xl = ns;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
else if(x >= pl){
    xl = ps;
    xh = pl;
    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = ns;
    yh = nl;
    c = 0;
    d = 1;
}
else if(nl < y && y < pl){
    if(z <= y < ps){
        yl = z;
        yh = ps;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ps <= y < pl){
        yl = ps;
        yh = pl;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ns < y < z){
        yl = z;
        yh = ns;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(nl < y <= ns){

```

```

        yl = ns;
        yh = nl;
        d = (y-yl)*se;
        c = 1-d;
    }
}

else if(y >= pl){
    yl = ps;
    yh = pl;
    c = 0;
    d = 1;
}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==z){rb_compare_low = pl; }

    if (yh==z){rb_compare_high = pl; }

}

if(xl==ps){
    if (yl==z){rb_compare_low = pl; }

    if (yh==z){rb_compare_high = pl; }

}

if(xl==z){
    if (yl==ps){rb_compare_low = ps; }
    if (yl==z){rb_compare_low = z; }
    if (yl==ns){rb_compare_low = ns; }

    if (yh==ps){rb_compare_high = ps; }
    if (yh==z){rb_compare_high = z; }
    if (yh==ns){rb_compare_high = ns; }

}

if(xl==ns){
    if (yl==z){rb_compare_low = ns; }

    if (yh==z){rb_compare_high = ns; }

}

```

```

if(xl==nl){
    if (yl==z){rb_compare_low = nl; }

    if (yh==z){rb_compare_high = nl;}
}

rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==z){rb_compare_low = pl; }

    if (yh==z){rb_compare_high = pl;}
}

if(xh==ps){
    if (yl==z){rb_compare_low = pl; }

    if (yh==z){rb_compare_high = pl;}
}

if(xh==z){
    if (yl==ps){rb_compare_low = ps; }
    if (yl==z){rb_compare_low = z; }
    if (yl==ns){rb_compare_low = ns; }

    if (yh==ps){rb_compare_high = ps; }
    if (yh==z){rb_compare_high = z; }
    if (yh==ns){rb_compare_high = ns; }
}

if(xh==ns){
    if (yl==z){rb_compare_low = ns; }

    if (yh==z){rb_compare_high = ns;}
}

if(xh==nl){
    if (yl==z){rb_compare_low = nl; }

    if (yh==z){rb_compare_high = nl;}
}

rb3 = rb_compare_low;
rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){

}

```

```

        mag_infer_1 = dom_xh;
    }
else {
    mag_infer_1 = dom_yh;
}

if(dom_xh < dom_yl){
    mag_infer_2 = dom_xh;
}
else {
    mag_infer_2 = dom_yl;
}

if(dom_xl < dom_yh){
    mag_infer_3 = dom_xl;
}
else {
    mag_infer_3 = dom_yh;
}

if(dom_xl < dom_yl){
    mag_infer_4 = dom_xl;
}
else {
    mag_infer_4 = dom_yl;
}

//Perhitungan MAX
//Max Layer
if(rb1 == rb2){
    if(mag_infer_1 < mag_infer_2){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_2 = 0.0;
    }
}

else if(rb1 == rb3){
    if(mag_infer_1 < mag_infer_3){
        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb1 == rb4){
    if(mag_infer_1 < mag_infer_4){

```

```

        mag_infer_1 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb2 == rb3){
    if(mag_infer_2 < mag_infer_3){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb2 == rb4){
    if(mag_infer_2 < mag_infer_4){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb3 == rb4){
    if(mag_infer_3 < mag_infer_4){
        mag_infer_3 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

//DEFUZZIFICATION
iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

X[0] = iq_ref;
X[1] = e;
X[2] = ec;
}

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

#ifndef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */

```

```
#include "simulink.c" /*MEX-file interface mechanism*/
#ifndef CG_SFUN_H
#define CG_SFUN_H /*Code generation registration function*/
#endif
```

FUZZY9SIM.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME FUZZY9SIM
#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, 2);
    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-4);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDIT IONS
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);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    //DECLARE VARIABEL
    real_T iq_ref, e, ec;

    //AMBIL DARI MEMORI
    iq_ref = X[0];
    e = X[1];
    ec = X[2];

    //OUTPUT
    Y[0] = iq_ref;
    Y[1] = e;
    Y[2] = ec;
}

#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;

    //DECLARE VARIABEL
    real_T e, ec, e_prev, w_ref, w_act;
    real_T nl, nm, ns, z, ps, pm, pl;
    real_T x, xl, xh, y, yl, yh, se, a, b, c ,d;
    real_T dom_xh, dom_xl, dom_yh, dom_yl;
    real_T rb1,rb2,rb3,rb4;
    real_T rb_compare_low,rb_compare_high;
    real_T mag_infer_1,mag_infer_2,mag_infer_3,mag_infer_4;
    real_T iq_ref;
    real_T gmf = 4.1;

    //INPUT
    e_prev = X[1];
    w_ref = U(0);
    w_act = U(1);
}

```

```

//HITUNG ERROR DAN CHANGE OF ERROR
e = (w_ref - w_act);
ec = (e - e_prev);

//MEMBERSHIP FUNCTION
pl = 0.75*gmf;
pm = 0.5*gmf;
ps = 0.25*gmf;
z = 0.0*gmf;
ns = -0.25*gmf;
nm = -0.5*gmf;
nl = -0.75*gmf;

//SCALING CRISP INPUT GE & GCE
x = e*1000;
y = ec;

if(x < 0.0){se = -1;}
else {se = 1;}

//DEGREE OF MEMBERSHIP E
if(x <= nl){
    xl = nm;
    xh = nl;
    a = 0;
    b = 1;
}

else if(nl < x && x < pl){
    if(z <= x < ps){
        xl = z;
        xh = ps;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(ps <= x < pm){
        xl = ps;
        xh = pm;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(pm <= x < pl){
        xl = pm;
        xh = pl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
else if(ns < x < z){

```

```

        xl = z;
        xh = ns;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nm < x <= ns){
        xl = ns;
        xh = nm;
        b = (x-xl)*se;
        a = 1-b;
    }
    else if(nl < x <= nm){
        xl = nm;
        xh = nl;
        b = (x-xl)*se;
        a = 1-b;
    }
}
}

else if(x >= pl){
    xl = pm;
    xh = pl;
    a = 0;
    b = 1;
}

//DEGREE OF MEMBERSHIP EC
if(y <= nl){
    yl = nm;
    yh = nl;
    c = 0;
    d = 1;
}

else if(nl < y && y < pl){
    if(z <= y < ps){
        yl = z;
        yh = ps;
        d = (y-yl)*se;
        c = 1-d;
    }
    else if(ps <= y < pm){
        yl = ps;
        yh = pm;
        d = (y-yl)*se;
        c = 1-d;
    }
}
else if(pm <= y < pl){

```

```

        yl = pm;
        yh = pl;
        d = (y-yl)*se;
        c = 1-d;

    }
else if(ns < y <z){
    yl = z;
    yh = ns;
    d = (y-yl)*se;
    c = 1-d;
}
else if(nm < y <= ns){
    yl = ns;
    yh = nm;
    d = (x-xl)*se;
    c = 1-d;
}
else if(nl < y <= nm){
    yl = nm;
    yh = nl;
    d = (y-yl)*se;
    c = 1-d;
}
}

else if(y >= pl){
    yl = pm;
    yh = pl;
    c = 0;
    d = 1;

}

dom_xh = a;
dom_xl = b;
dom_yh = c;
dom_yl = d;

//RULE BASE
//XL dengan YL dan Yh
if(xl==pl) {
    if (yl==z){rb_compare_low = pl;}
    if (yh==z){rb_compare_high = pl;}
}

if(xl==pm){

```

```

    if (yl==z){rb_compare_low = pm; }

    if (yh==z){rb_compare_high = pm; }
}

if(xl==ps){
    if (yl==z){rb_compare_low = ps; }

    if (yh==z){rb_compare_high = ps; }
}

if(xl==z){
    if (yl==ps){rb_compare_low = ps; }
    if (yl==z){rb_compare_low = z; }
    if (yl==ns){rb_compare_low = ns; }

    if (yh==ps){rb_compare_high = ps; }
    if (yh==z){rb_compare_high = z; }
    if (yh==ns){rb_compare_high = ns; }
}

if(xl==ns){
    if (yl==z){rb_compare_low = ns; }

    if (yh==z){rb_compare_high = ns; }
}

if(xl==nm){
    if (yl==z){rb_compare_low = nm; }

    if (yh==z){rb_compare_high = nm; }
}

if(xl==nl){
    if (yl==z){rb_compare_low = nl; }

    if (yh==z){rb_compare_high = nl; }
}

rb1 = rb_compare_low;
rb2 = rb_compare_high;

//Xh dengan YL dan Yh
if(xh==pl) {
    if (yl==z){rb_compare_low = pl; }

    if (yh==z){rb_compare_high = pl; }
}

```

```

if(xh==pm){
    if (yl==z){rb_compare_low = pm; }

    if (yh==z){rb_compare_high = pm;}
}

if(xh==ps){
    if (yl==z){rb_compare_low = ps; }

    if (yh==z){rb_compare_high = ps;}
}

if(xh==z){
    if (yl==ps){rb_compare_low = ps; }
    if (yl==z){rb_compare_low = z; }
    if (yl==ns){rb_compare_low = ns; }

    if (yh==ps){rb_compare_high = ps; }
    if (yh==z){rb_compare_high = z; }
    if (yh==ns){rb_compare_high = ns; }
}

if(xh==ns){
    if (yl==z){rb_compare_low = ns; }

    if (yh==z){rb_compare_high = ns;}
}

if(xh==nm){
    if (yl==z){rb_compare_low = nm; }

    if (yh==z){rb_compare_high = nm; }
}

if(xh==nl){
    if (yl==z){rb_compare_low = nl; }

    if (yh==z){rb_compare_high = nl; }
}

rb3 = rb_compare_low;
rb4 = rb_compare_high;

//INFERENCE ENGINE - MAX MIN ALGORITHM
//Min Layer
if(dom_xh < dom_yh){
    mag_infer_1 = dom_xh;
}
else {

```

```

        mag_infer_1 = dom_yh;
    }

    if(dom_xh < dom_yl){
        mag_infer_2 = dom_xh;
    }
    else {
        mag_infer_2 = dom_yl;
    }

    if(dom_xl < dom_yh){
        mag_infer_3 = dom_xl;
    }
    else {
        mag_infer_3 = dom_yh;
    }

    if(dom_xl < dom_yl){
        mag_infer_4 = dom_xl;
    }
    else {
        mag_infer_4 = dom_yl;
    }

    //Perhitungan MAX
    //Max Layer
    if(rb1 == rb2){
        if(mag_infer_1 < mag_infer_2){
            mag_infer_1 = 0.0;
        }
        else{
            mag_infer_2 = 0.0;
        }
    }

    else if(rb1 == rb3){
        if(mag_infer_1 < mag_infer_3){
            mag_infer_1 = 0.0;
        }
        else{
            mag_infer_3 = 0.0;
        }
    }

    else if(rb1 == rb4){
        if(mag_infer_1 < mag_infer_4){
            mag_infer_1 = 0.0;
        }
        else{

```

```

        mag_infer_4 = 0.0;
    }
}

else if(rb2 == rb3){
    if(mag_infer_2 < mag_infer_3){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_3 = 0.0;
    }
}

else if(rb2 == rb4){
    if(mag_infer_2 < mag_infer_4){
        mag_infer_2 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

else if(rb3 == rb4){
    if(mag_infer_3 < mag_infer_4){
        mag_infer_3 = 0.0;
    }
    else{
        mag_infer_4 = 0.0;
    }
}

//DEFUZZIFICATION
iq_ref = ((mag_infer_1*rb1)+(mag_infer_2*rb2)+(mag_infer_3*rb3)+(mag_infer_4*rb4))/(mag_infer_1+mag_infer_2+mag_infer_3+mag_infer_4);

X[0] = iq_ref;
X[1] = e;
X[2] = ec;
}

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

#endif 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_IM.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME RFOC_IM
#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, 12);
    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-4);
    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 = X[2];
real_T Vb = X[3];
real_T Vc = X[4];

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-4;
    //Variabel Input
    real_T Id_ref,Iq_ref,Ia,Ib,Ic,Kpd,Kid,Kpq,Kiq,imr,wr,Thetae;

    //Variabel RFOC
    real_T vds_ref,vqs_ref,Valpha,Vbeta,Va,Vb,Vc;

    //Variabel Transformasi
    real_T Ialpha,Ibeta,Id,Iq;

    //Parameter Motor
    real_T Lm=0.2279, Ls=0.2349, Lr=0.2349;
    real_T Rs=2.76, Rr=2.9;
    real_T a0=1/( Lm*Lm-Ls*Lr);
    real_T K=0.816497, L=0.866025;
    real_T B =0.0005, J=0.0436, Np=2.0;

    //Konstanta Transformasi
    real_T A      = 0.81649658092;      // sqrt(2/3)
    real_T Q      = 0.86602540378;      // sqrt(3)/2

    //Ambil Input
    Id_ref   = U(0);
    Kpd     = U(1);
    Kid     = U(2);
    Iq_ref   = U(3);
    Kpq     = U(4);
    Kiq     = U(5);
    Ia       = U(6);
    Ib       = U(7);
    Ic       = U(8);
    imr     = U(9);
}

```

```

wr      = U(10);
Thetae = U(11);

real_T integralId_old,integralIq_old;
integralId_old = X[0];
integralIq_old = X[1];

//Transformasi Alfabet
Ialfa = A*(Ia - 0.5*Ib - 0.5*Ic);
Ibeta = A*(Q*Ib - Q*Ic);

//Transformasi DQ
Id     = Ialfa*cos(Thetae) + Ibetta*sin(Thetae);
Iq     = -Ialfa*sin(Thetae) + Ibetta*cos(Thetae);

//Current Control Signal
real_T errorId = Id_ref - Id;
real_T errorIq = Iq_ref - Iq;

real_T integralId = integralId_old + errorId*dt;
real_T integralIq = integralIq_old + errorIq*dt;

real_T uds = Kpd*errorId + Kid*integralId;
real_T uqs = Kiq*errorIq + Kid*integralIq;

//Flux Model
real_T omegae = (Np*wr)+((Rr/Lr)*(Iq_ref/(imr+0.001))); //ditambah 0.001 agar tidak pembagian 0 pada awal simulasi

// real_T omegae;
// if(imr==0){
//     omegae = (Np*wr)+((Rr/Lr)*(Iq_ref));
// }
// else{
//     omegae = (Np*wr)+((Rr/Lr)*(Iq_ref/imr));
// }

real_T imr_dt = (Rr/Lr)*(Id-imr);
real_T epsilon = (1-((Lm*Lm)/(Ls*Lr)));

//Decoupling Circuit
real_T vcd = ((-omegae)*Ls*epsilon*Iq) + (Ls*(1-epsilon)*imr_dt);
real_T vcq = (omegae*Ls*epsilon*Id) + (Ls*(1-epsilon)*omegae*imr);

//Voltage references
vds_ref = uds + vcd;
vqs_ref = uqs + vcq;

```

```

//Transformasi Alfabeto
Valpha = vds_ref*cos(Thetae) - vqs_ref*sin(Thetae);
Vbeta = vds_ref*sin(Thetae) + vqs_ref*cos(Thetae);

//Transformasi abc
Va      = A*Valpha;
Vb      = A*(-0.5*Valpha + Q*Vbeta);
Vc      = A*(-0.5*Valpha - Q*Vbeta);

X[0]   = integralId;
X[1]   = integralIq;
X[2]   = Va;
X[3]   = Vb;
X[4]   = Vc;
X[5]   = imr;

}

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

#ifndef 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

```

PWM.c

```

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME PWM
#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); //input
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 3); //output
    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);

int_T i;
real_T t = ssGetT(S);
real_T halfVDC = (0.5*U(1));

for( i=0; i < 3; i++)
{
    if( fabs( U(i+2) )/halfVDC >= U(0)*fmod(t,1/U(0)) )
    {
        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*/

#ifndef 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

```

IM.c

```

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

```

```

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

static void mdlInitializeSizes(SimStruct *S){
    ssSetNumContStates(S, 6);
    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, 8);
    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 K = 0.816497, L = 0.866025;
    real_T Lm = 0.2279, Lr = 0.2349;
    real_T Np = 2.0, pi2 = 6.28318530718;
    real_T fx, fy;

    Y[0] = K*X[0];
    Y[1] = K*(-0.5*X[0] + L*X[1]);
    Y[2] = K*(-0.5*X[0] - L*X[1]);
    fx = (Lr*X[2] + Lm*X[0]);
    fy = (Lr*X[3] + Lm*X[1]);
    Y[3] = (sqrt(fx*fx + fy*fy))/Lm;
    Y[4] = Np*Lm*( X[1]*X[2] - X[0]*X[3] );
    if(fy == 0.0){fy = 1e-6;}
    if(fx < 0.0 && fy < 0.0){Y[5]= pi2 + atan2(fy,fx);}
    if(fx < 0.0 && fy > 0.0){Y[5]= atan2(fy,fx);}
}

```

```

if(fx >= 0.0 && fy < 0.0){Y[5]= pi2 + atan2(fy,fx);}
if(fx >= 0.0 && fy > 0.0){Y[5]= atan2(fy,fx);}
while(X[5] > pi2){ X[5] -= pi2;}
while(X[5] < 0.0) { X[5] += pi2;}
Y[6] = X[5];
Y[7] = X[4];
}

#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 Lm=0.2279, Ls=0.2349, Lr=0.2349;
    real_T Rs=2.76, Rr=2.9;
    real_T a0=1/( Lm*Lm-Ls*Lr);
    real_T K=0.816497, L=0.866025;
    real_T B =0.0005, J=0.0436, Np=2.0;

    real_T idsn_dot, iqsn_dot, idrn_dot, iqrn_dot;
    real_T wr_dot, theta_r_dot, vds, vqs, Te;

    vds = K*( U(0) - 0.5*U(1) - 0.5*U(2) );
    vqs = K*L*( U(1) - U(2) );

    idsn_dot = (Rs*Ls*X[0]-Np*X[4]*Lm*Lm*X[1]-Rr*Lm*X[2]-Np*X[4]*Lr*Lm*X[3]-
Lr*vds)*a0;
    iqsn_dot = (Np*X[4]*Lm*Lm*X[0]+Rs*Lr*X[1] +Np*X[4]*Lr*Lm*X[2]-Rr*Lm*X[3]-
Lr*vqs)*a0;
    idrn_dot = -(Rs*Lm*X[0]-Np*X[4]*Lm*Ls*X[1] -Rr*Ls*X[2]-Np*X[4]*Lr*Ls*X[3]-
Lm*vds)*a0;
    iqrn_dot = -(Np*X[4]*Lm*Ls*X[0]+Rs*Lm*X[1] +Np*X[4]*Lr*Ls*X[2]-Rr*Ls*X[3]-
Lm*vqs)*a0;

    Te = Np * Lm*(X[1] * X[2] - X[0] * X[3]);
    wr_dot = ( Te - U(3) - B * X[4] ) / J;
    theta_r_dot = X[4];

    dX[0] = idsn_dot;
    dX[1] = iqsn_dot;
    dX[2] = idrn_dot;
    dX[3] = iqrn_dot;
    dX[4] = wr_dot;
    dX[5] = theta_r_dot;
}

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 105  
#include "cg_sfun.h" /*Code generation registration function*/  
#endif
```