



TESIS - TM 185400

**ANALISA TRANSIENT PADA CONTROLLER MOTOR
BRUSHLESS DIRECT CURRENT DENGAN MENGGUNAKAN
LOOK UP TABLE FIELD ORIENTED CONTROL**

AGUS NURTRIARTONO

02111850080003

Supervisor

Dr. Muhammad Nur Yuniarto

Departement Teknik Mesin

Fakultas Teknologi Industri

Institut Teknologi Sepuluh Nopember

2020



THESIS - TM 185400

***ANALISA TRANSIENT PADA CONTROLLER
MOTOR BRUSHLESS DIRECT CURRENT
DENGAN MENGGUNAKAN LOOK UP TABLE
FIELD ORIENTED CONTROL***

AGUS NURTRIARTONO
02111850080003

Dosen Pembimbing
Dr. Muhammad Nur Yuniarto

Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember
2020

(halaman ini sengaja dikosongkan)



TESIS - TM 185400

TRANSIENT ANALYSIS ON BRUSHLESS DIRECT CURRENT MOTOR CONTROLLER USING LOOK UP TABLE FIELD ORIENTED CONTROL

AGUS NURTRIARTONO
02111850080003

Supervisor
Dr. Muhammad Nur Yuniarto

**Mechanical Engineering Department
Faculty of Industrial and Systems Engineering
Institut Teknologi Sepuluh Nopember
2020**

(halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar

Magister Teknik (M.T.)

di

Institut Teknologi Sepuluh Nopember

oleh:

Agus Nurtriartono

NRP: 02111850080003

Tanggal Ujian: 15 Juli 2020

Periode Wisuda: September 2020

Disetujui Oleh:

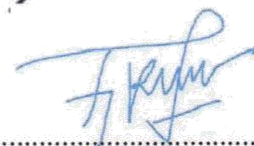

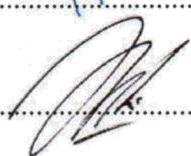
Pembimbing:

1. Dr.Muhammad Nur Yuniarto
NIP. 197506301998021001


.....


Penguji:

1. Dr. Ir. Agus Sigit Pramono, DEA
NIP. 196508101991021001
2. Alief Wikarta, ST, M.Sc.Eng. PhD
NIP. 198202102006041002
3. Dr.Eng.Yohanes, ST, M.Sc
NIP. 1980062720012121003

.....

.....

.....


Kepala Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem

.....

Dr.Ir.Atok Setiyawan, M.Eng.Sc.
NIP. 196604021989031002

(halaman ini sengaja dikosongkan)

ANALISA TRANSIENT PADA CONTROLER MOTOR BRUSHLESS DIRECT CURRENT DENGAN MENGGUNAKAN LOOK UP TABLE FIELD ORIENTED CONTROL

Nama Mahasiswa : Agus Nurtriartono
NRP : 02111850080003
Dosen Pembimbing : Dr.Muhammad Nur Yuniarto

ABSTRAK

Brushless Direct Current (BLDC) *motor controller* umum digunakan pada kendaraan listrik. Institut Teknologi Sepuluh Nopember (ITS) Surabaya merupakan salah satu pusat pengembangan kendaraan listrik di Indonesia dengan mengembangkan komponen-komponen utama kendaraan listrik, salah satunya adalah BLDC *motor controller*. Namun BLDC *motor controller* ini masih memiliki kelemahan yaitu kondisi *starting* yang kasar akibat limit torsi atau limit arus yang membuat controller berhenti beroperasi ketika torsi atau arus mencapai limit.

Penelitian ini bertujuan untuk memperbaiki kelemahan tersebut dengan menganalisa respon *transient controller* pada pemodelan *Look up Table Field Oriented Control* (FOC) menggunakan *Matlab/Simulink*. Adapun langkah-langkah yang dilakukan ialah mencari persamaan *Longitudinal Vehicle Dynamic* untuk memperoleh nilai torsi, *power*, dan rpm maksimum sehingga dapat menentukan spesifikasi motor BLDC. Kemudian mendeteksi parameter dan menguji stabilitas model motor BLDC. Selanjutnya menentukan spesifikasi respon *transient* berupa nilai *rise time*, *settling time*, dan *percent overshoot*. Terakhir melakukan pemodelan FOC dengan *look up table* menggunakan *Matlab/Simulink* dan *tuning* parameter PI untuk memperoleh respon yang memiliki nilai spesifikasi respon *transient* sesuai target.

Pada pemodelan *lookup table field oriented control* dengan nilai parameter PI *default* ($k_p = 1$ dan $k_i = 1$) diperoleh hasil simulasi yang ditinjau dari respon rpm berupa nilai *rise time* sebesar 0,6445 detik, *settling time* sebesar 1,979 detik,

dan nilai *percent overshoot* sebesar 0%. Sedangkan untuk hasil simulasi setelah melakukan proses *tuning* parameter ($k_p = 10$ dan $k_i = 0,5$) diperoleh nilai *rise time* sebesar 0,5522 detik, *settling time* sebesar 0,878 detik, dan *percent overshoot* sebesar 0%. Sementara pada pemodelan *field oriented control* diperoleh hasil simulasi yang ditinjau dari respon rpm berupa nilai *rise time* sebesar 0,6 detik, *settling time* sebesar 1 detik, dan *percent overshoot* sebesar 0%. Pada pemodelan *field oriented control vedder* diperoleh hasil simulasi yang ditinjau dari respon rpm berupa *rise time* sebesar 0,65 detik, *settling time* sebesar 1,5 detik, dan *percent overshoot* sebesar 0%. Sehingga dapat disimpulkan bahwa pemodelan *lookup table field oriented control* memperoleh hasil respon lebih baik dibandingkan dengan pemodelan *field oriented control*, dan *field oriented control vedder*.

Kata kunci: *Transient Analysis; FOC Control; PI*

TRANSIENT ANALYSIS ON BRUSHLESS DIRECT CURRENT MOTOR CONTROLLER USING LOOK UP TABLE FIELD ORIENTED CONTROL

Name : Agus Nurtriartono
NRP : 02111850080003
Supervisor : Dr.Muhammad Nur Yuniarto

ABSTRACT

Brushless Direct Current (BLDC) motor controller is widely used on electric vehicle. Sepuluh Nopember Institute of Technology (ITS) Surabaya is one of the centers for developing electric vehicles in Indonesia develops main components in electric vehicles, one of which is the BLDC motor controller. However BLDC motor still has a disadvantage that a rough starting condition, this happens because there is a torque/current limit which disables the controller when the torque/current reaches the limit.

This study aims to correct this disadvantage by analyzing the transient response of controller on Look up Table Field Oriented Control (FOC) models using Matlab/ Simulink. The steps to be taken are looking for the Longitudinal Vehicle Dynamic equations to obtain the value of torque, power, and maximum rpm to determine the specifications of the BLDC motor. The next is detecting the BLDC Motor parameters and test the stability of the BLDC motor model. Next determines the specifications of the transient response in the form of rise time, settling time, and percent overshoot. The last is modeling the FOC on Look up Table using Matlab/Simulink and tuning the PI parameters to obtain a response that has a transient response specifications value according to the target.

Lookup table field oriented control model with PI default parameter values ($k_p = 1$ and $k_i = 1$) the simulation results are obtained in terms of the rpm response is 0.6445 s rise time, 1.979 s settling time, and 0% percent overshoot. While for the simulation results after tuning the parameters ($k_p = 10$ and $k_i = 0.5$) obtained 0.5522 s rise time, 0.878 s settling time, and 0% percent overshoot. field

oriented control model obtained simulation results in terms of rpm response is 0.6 s rise time, 1s settling time, and 0% percent overshoot. Field oriented control vedder model obtained simulation results in terms of rpm response is 0.65 s rise time, 1.5 s settling time, and 0% percent overshoot. So it can be concluded that the lookup table field oriented control model has better response results than the field oriented control , and field oriented control vedder model.

Keywords: Transient Analysis; FOC Control; PI

KATA PENGANTAR

Puji syukur kehadiran Tuhan Yang Maha Esa karena atas anugerah, berkah dan hidayah-Nya laporan tesis yang berjudul “Analisa *Transient* pada *Controller Motor Brushless Direct Current* dengan Menggunakan *Look up Table Field Oriented Control*” ini dapat diselesaikan.

Dalam kesempatan ini, penulis ingin menyampaikan rasa terima kasih yang sedalam-dalamnya kepada:

1. Bapak Dr. Muhammad Nur Yuniarto selaku dosen pembimbing yang telah banyak memberikan arahan, bimbingan serta pelajaran selama pembuatan dan penyelesaian tesis ini.
2. Orangtua serta seluruh keluarga yang telah banyak memberikan dukungan moral.
3. Seluruh tim Wiksa Daya Pratama dan tim *Stupid but Spirit* yang telah banyak membantu dalam pembuatan tesis ini.
4. Putri Rahajeng Utami yang telah memberi dukungan dan semangat dalam menyelesaikan tesis ini.

Penulis menyadari bahwa laporan tesis ini masih belum sempurna, baik dari analisis yang penulis lakukan maupun dalam penulisan laporan. Semoga laporan ini dapat memberikan manfaat bagi pembaca pada umumnya dan penulis pada khususnya.

Surabaya, Juli 2020

Penulis



Agus Nurtriartono
NRP 02111850080003

(halaman ini sengaja dikosongkan)

DAFTAR ISI

COVER	i
LEMBAR PENGESAHAN	v
ABSTRAK	vii
ABSTRACT	ix
KATA PENGANTAR.....	xi
DAFTAR ISI.....	xiii
DAFTAR GAMBAR.....	xv
DAFTAR TABEL	xixx
BAB 1 PENDAHULUAN	
1.1 Latar Belakang	1
1.2 Rumusan Masalah	4
1.3 Tujuan	4
1.4 Manfaat	4
1.5 Batasan Masalah.....	4
BAB 2 TINJAUAN PUSTAKA	
2.1 <i>Longitudinal Vehicle Dynamic</i>	5
2.2 Motor BLDC	6
2.2.1 Model Matematika Motor BLDC	7
2.2.2 <i>Field Oriented Control</i>	10
2.2.3 Karakteristik respon waktu (Time Respon)	13
2.2.4 PID Kontrol.....	15
2.2.5 Penentuan nilai Konstanta PID <i>Ziegler Nichols</i>	19
2.3 <i>Look up Table</i>	20
2.4 <i>Controller</i> motor BLDC.....	21
BAB 3 METODE PENELITIAN	
3.1 Rancangan Penelitian	25
3.2 Perhitungan <i>Vehicle Dynamic</i> Sepeda Motor Listrik.....	29
3.2.1 Pemodelan <i>Vehicle Dynamic</i>	28
3.3 Metode Pengambilan Data Karakteristik Motor	36
3.3.1 Parameter Motor: Pengukuran Resistansi dan Induktansi Motor	34

3.3.2 Parameter Motor: Konstanta BEMF	35
3.3.3 Parameter Motor: Konstanta Torsi	37
3.3.5 Parameter Motor: Koefisien Gaya Gesek	38
3.3.6 Parameter Motor: Momen Inersia.....	42
3.3.7 Pemodelan BLDC Motor.....	42
3.4 Spesifikasi Respon <i>Transient</i>	44
3.5 Simulasi dengan <i>Matlab/Simulink</i>	46
3.6 <i>Tuning</i> Parameter PI	47
BAB 4 PEMODELAN LOOKUP TABLE FIELD ORIENTED CONTROL	
DAN ANALISA TRANSIENT	
4.1 Pemodelan <i>Lookup Table</i> FOC.....	49
4.1.1 Perancangan Model <i>Lookup Table</i> FOC	49
4.1.2 Perancangan <i>Lookup Table</i>	50
4.1.3 Simulasi <i>Lookup Table</i> FOC	53
4.2 Simulasi FOC Vedder, FOC, dan <i>Lookup Table</i> FOC	55
4.2.1 Simulasi FOC Vedder Ditinjau dari Respon rpm.....	56
4.2.2 Simulasi FOC Ditinjau dari Respon rpm.....	58
4.2.3 Simulasi <i>Lookup Table</i> FOC Dengan Ditinjau dari Respon rpm	59
4.2.4 Perbandingan Hasil Simulasi Ditinjau dari Respon rpm	61
4.3 Simulasi FOC Vedder, FOC, dan <i>Lookup Table</i> FOC	62
4.3.1 Simulasi FOC Vedder Ditinjau dari Respon Torsi.....	62
4.3.2 Simulasi FOC Ditinjau dari Respon Torsi.....	64
4.3.3 Simulasi <i>Lookup Table</i> FOC Dengan Ditinjau dari Respon Torsi	66
4.3.4 Perbandingan Hasil Simulasi Ditinjau dari Respon Torsi	68
4.4 Simulasi FOC Model FOC, dan <i>Lookup Table</i> FOC	69
4.5 Generate Simulasi <i>Lookup Table</i> FOC ke <i>C Code</i>	72
BAB 5 KESIMPULAN DAN SARAN	
5.1 Kesimpulan.....	75
5.2 Saran	76
DAFTAR PUSTAKA	77
LAMPIRAN	79
BIODATA PENULIS	119

DAFTAR GAMBAR

Gambar 1.1. Sistem Proteksi Saat Ini dan Sistem Proteksi yang Sesuai.....	2
Gambar 2.1. Free body diagram sepeda motor 3S	5
Gambar 2.2. Rangkaian ekuivalen motor BLDC	7
Gambar 2.3. Model <i>Inverter</i> 3 Phase dan BLDC Motor	8
Gambar 2.4. Ekuivalen 3 fasa motor BLDC.	9
Gambar 2.5. Diagram bentuk gelombang BEMF motor BLDC	10
Gambar 2.6. Arah Medan magnet yang dihasilkan stator terhadap rotor	11
Gambar 2.7. Block Diagram FOC	11
Gambar 2.8. Vector diagram of <i>Clarke Transformation</i>	12
Gambar 2.9. Vector diagram of <i>Park Transformation</i>	20
Gambar 2.10 Respon <i>Transient</i> dan Respon <i>Steady State</i>	14
Gambar 2.11. Diagram PID Controller	15
Gambar 2.12. Respon dari sitem terhadap perubahan K_p , K_i dan K_d	17
Gambar 2.13. Kurva S Analisa Grafis <i>Ziegler Nichols</i>	18
Gambar 2.14. Look up Table (a) ECU sepeda motor konvensional (b)	19
Gambar 2.15. <i>Controller</i> Wiksa Mark 1.2	20
Gambar 2.16. Pengaturan batas jumlah arus dari baterai ke motor	21
Gambar 2.17. Pengaturan suhu maksimum pada VESC.....	21
Gambar 2.18. Pengaturan tegangan input minimum dan maksimum	22
Gambar 3.1. <i>Free Body Diagram</i> Sepeda Motor Listrik	27
Gambar 3.2. Block Diagram <i>Vehicle Modeling</i>	29
Gambar 3.3. Block Diagram <i>Longitudinal Vehicle Dynamic</i>	29
Gambar 3.4. Hasil dari <i>Vehicle Modeling</i>	30
Gambar 3.5. <i>Zoom</i> hasil dari <i>Vehicle modeling</i>	30
Gambar 3.6. <i>Power</i> hasil dari <i>Vehicle modeling</i>	31
Gambar 3.7. Torsi hasil dari <i>Vehicle modeling</i>	31
Gambar 3.8. Rpm hasil dari <i>Vehicle modeling</i>	32
Gambar 3.9. Pemodelan <i>vehicle dynamic</i> pada sudut kemiringan jalan 8°	32
Gambar 3.10. <i>Power</i> pada sudut kemiringan jalan 8°	33

Gambar 3.11. Torsi pada sudut kemiringan jalan 8°	33
Gambar 3.12. Hasil Uji Performa Motor Listrik 3S pada <i>Dynamometer</i>	34
Gambar 3.13. Skema pengukuran Resistansi dan Induktansi	35
Gambar 3.14. Skema pengukuran konstanta BEMF	36
Gambar 3.15. Respon kecepatan dengan komutasi <i>six-step</i>	38
Gambar 3.16. Skema pengukuran arus dan rpm	38
Gambar 3.17. <i>Block diagram</i> BLDC by <i>six-Step Inverter</i>	40
Gambar 3.18. Parameter BLDC motor	40
Gambar 3.19. Respon pengujian <i>open loop plant</i> BLDC motor	41
Gambar 3.20. Graphik <i>damping ratio vs rise time</i>	42
Gambar 3.21. <i>Block Diagram</i> FOC	43
Gambar 3.22. <i>Look up Table</i> Pada <i>Simulink</i>	44
Gambar 3.23. <i>Tuning</i> Parameter PI	44
Gambar 4.1. Model FOC Dengan <i>Lookup Table</i>	45
Gambar 4.2. Model <i>Simulink Lookup Table</i> FOC	46
Gambar 4.3. Model <i>Vehicle Dynamic</i>	46
Gambar 4.4. Data Torsi Pada Jalan Datar (a) dan Pada Jalan Menanjak (b)	47
Gambar 4.5. Data <i>Lookup Table</i>	48
Gambar 4.6. Proses Memasukan Data <i>Lookup Table</i>	48
Gambar 4.7. Simulasi model <i>simulink Lookup Table</i> FOC	49
Gambar 4.8. Parameter PI <i>default</i> pada <i>Lookup Table</i> FOC	49
Gambar 4.9. Respon rpm terhadap waktu	50
Gambar 4.10. Parameter PI setelah <i>tuning</i> pada <i>Lookup Table</i> FOC	50
Gambar 4.11. Respon rpm terhadap waktu	51
Gambar 4.12. Simulasi model <i>simulink</i> FOC <i>Vedder</i>	52
Gambar 4.13. Respon rpm terhadap waktu dari model <i>simulink</i> FOC <i>Vedder</i>	52
Gambar 4.14. Hasil simulasi model <i>simulink</i> sistem FOC <i>Vedder</i>	53
Gambar 4.15. Simulasi model <i>simulink</i> FOC	53
Gambar 4.16. Respon rpm terhadap waktu dari model <i>simulink</i> FOC	54
Gambar 4.17. Hasil simulasi model <i>simulink</i> FOC	54
Gambar 4.18. Simulasi model <i>simulink Lookup Table</i> FOC	55
Gambar 4.19. Respon rpm terhadap waktu	55

Gambar 4.20. Hasil simulasi model <i>simulink Lookup Table FOC</i>	56
Gambar 4.21. Perbandingan respon rpm terhadap waktu	57
Gambar 4.22. Simulasi model <i>simulink FOC Vedder</i>	58
Gambar 4.23. Hasil simulasi model <i>simulink FOC Vedder</i>	59
Gambar 4.24. <i>Zoom</i> Respon Torsi	59
Gambar 4.25. Simulasi model <i>simulink FOC</i>	60
Gambar 4.26. Hasil simulasi model <i>simulink FOC</i>	60
Gambar 4.27. <i>Zoom</i> Respon Torsi	61
Gambar 4.28. Simulasi model <i>simulink Lookup Table FOC</i>	61
Gambar 4.29. Hasil simulasi model <i>simulink Lookup Table FOC</i>	62
Gambar 4.30. <i>Zoom</i> respon torsi	62
Gambar 4.31. Perbandingan respon torsi terhadap waktu.....	63
Gambar 4.32. <i>Zoom</i> perbandingan respon torsi terhadap waktu.....	64
Gambar 4.33. Simulasi model <i>simulink FOC</i> dengan variasi beban.....	65
Gambar 4.34. Simulasi model <i>simulink Lookup Table FOC</i>	65
Gambar 4.35. Variasi beban pada simulasi model <i>simulink FOC</i>	66
Gambar 4.36. Hasil simulasi model <i>simulink FOC</i>	66
Gambar 4.37. <i>Zoom</i> hasil simulasi	67
Gambar 4.38. Proses <i>Generate code</i> menjadi bahasa C.....	68
Gambar 4.39. Pemilihan target <i>hardware processor</i>	68
Gambar 4.40. Bahasa C dari <i>Generate Code</i>	69

(halaman ini sengaja dikosongkan)

DAFTAR TABEL

Tabel 2.1 <i>Six Step Comutation</i>	8
Tabel 2.2 Formula <i>Ziegler Nichols</i>	18
Tabel 3.1 Tabel pengukuran R dan L motor BLDC	35
Tabel 3.2 Tabel pengukuran untuk konstanta BEMF	37
Tabel 3.3 Tabel pengukuran untuk konstanta gaya gesek	39
Tabel 4.1 Tabel hasil simulasi ditinjau dari respon rpm	57
Tabel 4.2 Tabel perbandingan hasil simulasi ditinjau dari respon torsi.....	64

(halaman ini sengaja dikosongkan)

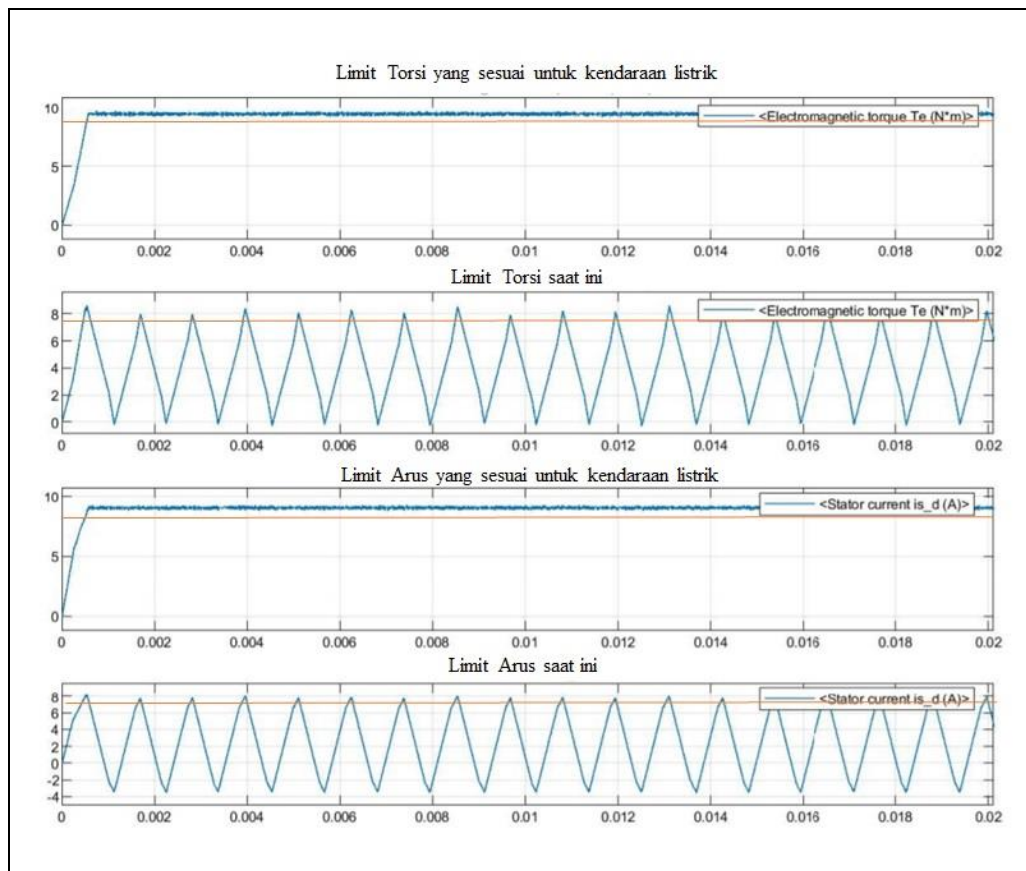
BAB 1

PENDAHULUAN

1.1. Latar Belakang

Motor dan *controller* merupakan komponen penting dalam pembuatan kendaraan listrik, salah satu jenis motor yang umum digunakan pada mobil listrik dunia ialah BLDC motor. Hal ini dikarenakan BLDC motor memiliki nilai efisiensi paling tinggi dibandingkan dengan jenis motor lainnya (Hong-xing Wu dan Shu-mei Cui, 2005) karena BLDC motor tidak menggunakan sikat pada pengaplikasiannya sehingga tidak ada gesekan yang terjadi antara sikat dengan rotor. Sebuah studi mengatakan bahwa motor BLDC merupakan motor yang cocok digunakan untuk aplikasi kendaraan listrik (B. K. Lee, M. Ehsani, 2001). Dalam pengaplikasiannya, BLDC motor membutuhkan *controller*. *Controller* tersebut mempengaruhi nilai dari performa dan efisiensi motor listrik.

Institut Teknologi Sepuluh Nopember (ITS) Surabaya merupakan salah satu pusat pengembangan kendaraan listrik di Indonesia dengan mengembangkan komponen-komponen utama pada kendaraan listrik seperti motor *controller* BLDC. ITS telah membuat beberapa *prototype controller* BLDC salah satunya yaitu *controller* yang digunakan pada sepeda motor listrik, dalam pengembangannya BLDC masih memiliki kelemahan yaitu kondisi *starting* yang kasar dan sistem proteksi berupa limit torsi atau arus yang tidak sesuai untuk aplikasi kendaraan listrik. Sistem proteksi yang digunakan pada controller saat ini memiliki cara kerja dengan menonaktifkan controller pada saat controller bekerja mencapai arus atau torsi limit. Kondisi ini berbahaya ketika kendaraan listrik *starting* atau menanjak dan controller mencapai limit arus atau torsi maka controller akan nonaktif dan akan aktif lagi pada beberapa saat dan hal ini akan membuat kondisi *starting* yang kasar seperti yang ditunjukkan pada Gambar 1.1. Sistem proteksi yang sesuai untuk kendaraan listrik adalah ketika arus atau torsi limit terlampaui, controller akan menahan arus atau torsi pada posisi limit tersebut dan bukan menonaktifkannya.



Gambar 1.1 Sistem proteksi saat ini dan sistem proteksi yang sesuai untuk kendaraan listrik

Kelemahan ini dapat diatasi dengan beberapa solusi seperti memakai spesifikasi *controller* yang lebih tinggi dari kebutuhan. Dengan memakai motor dan *controller over design* maka torsi atau arus tidak bisa mencapai limit. Namun hal ini berdampak pada biaya produksi yang menjadi mahal. Solusi kedua yaitu menggunakan *variable* transmisi pada mobil listrik. Solusi ini pernah digunakan ITS pada kegiatan *Explore* Indonesia yaitu dengan memakai transmisi mobil konvensional pada mobil listrik Blits dan Kasuari. Solusi ini pun memiliki kekurangan yaitu nilai efisiensi yang rendah, hal ini mengakibatkan berkurangnya jarak tempuh yang dapat ditempuh oleh mobil listrik tersebut. Solusi ketiga adalah merubah algoritma *controller*, dengan melakukan pemetaan kebutuhan torsi melalui analisa *vehicle dynamic* agar dapat menentukan rentang torsi operasi yang diperlukan kendaraan listrik. Solusi ketiga merupakan lebih efisien karena tidak memerlukan biaya fabrikasi untuk menerapkannya. Dengan

metode *field oriented control* yang merupakan suatu metode *control* di mana arus stator motor listrik pada tiga fase diidentifikasi sebagai dua komponen ortogonal yaitu komponen yang mendefinisikan fluks *current* (I_d) dan komponen yang mendefinisikan *torque current* (I_q), *current vector* dioptimalkan sehingga menghasilkan torsi maksimal pada setiap posisi rotor. Hal ini tidak dapat dilakukan oleh metode *control trapezoidal*. Pada penelitian ini penulis menambahkan *lookup table* yang berisi data torsi yang didapat dari simulasi model *vehicle dynamic*, dengan *lookup table* membaca *input* posisi *throttle* dan rpm sehingga torsi yang tersalurkan pada motor listrik menjadi optimal dan sesuai dengan kebutuhan. *Lookup table* dipilih agar proses pengaturan parameter dan pengaturan *limit controller* dapat dilakukan lebih mudah. Hal ini dikarenakan seluruh pengaturan ditampilkan dalam satu tampilan *lookup table*.

Penelitian ini bertujuan untuk memperbaiki kelemahan pada kondisi *starting* yang kasar dan sistem proteksi berupa limit torsi atau arus yang tidak sesuai untuk aplikasi kendaraan listrik dengan menganalisa respon *transient controller* pada pemodelan *lookup table field oriented control* dengan menggunakan *Matlab/Simulink*. Adapun langkah – langkah yang akan dilakukan adalah mencari persamaan *longitudinal vehicle dynamic* untuk mendapatkan nilai torsi, *power*, dan rpm maksimum yang digunakan untuk menentukan spesifikasi motor BLDC yang akan dipakai pada motor listrik. Selanjutnya dilakukan deteksi parameter motor BLDC untuk mendapatkan nilai-nilai parameter motor BLDC yang kemudian digunakan untuk mengisi nilai-nilai parameter pada *block diagram* motor BLDC pada *simulink*. Kemudian *block diagram* motor BLDC disimulasi menggunakan model *open loop* untuk mengetahui stabilitas dari model motor BLDC tersebut. Selanjutnya menentukan spesifikasi dari *transient* respon berupa nilai *rise time*, *settling time*, dan *percent overshoot* sebagai tolak ukur yang digunakan untuk mengukur kualitas respon *transient*. Terakhir melakukan pemodelan *field oriented control* dengan *lookup table* menggunakan *Matlab/Simulink* dan *tuning* parameter PI untuk mendapatkan respon *transient* sesuai spesifikasi yang diharapkan.

Berdasarkan latar belakang yang telah dijabarkan diatas, penulis bermaksud untuk melakukan sebuah studi penelitian mengenai *controller* motor BLDC dengan judul,

‘Analisa *Transient* pada *Controller Motor Brushless Direct Current* dengan Menggunakan *Look up Table Field Oriented Control*’

Penelitian ini dilakukan dengan menganalisa respon *controller* BLDC terhadap *transient* menggunakan model *lookup table field oriented control*. Model tersebut merupakan model *field oriented control* yang ditambahkan *lookup table* sebagai torque control pada sistem tersebut dan melakukan tuning parameter PI untuk mendapatkan respon *transient* yang baik.

1.2. Rumusan Masalah

Rumusan masalah tesis ini adalah sebagai berikut:

1. Bagaimana cara mengatasi *delivery torque* yang tidak sesuai kebutuhan?
2. Bagaimana cara mendapatkan performa respon *transient* dengan spesifikasi nilai *rise time*, *settling time*, dan *percent overshoot* yang telah ditentukan?

1.3. Tujuan

Tujuan tesis ini adalah sebagai berikut:

1. Melakukan analisa pemodelan *vehicle dynamic* dan melakukan pemodelan *lookup table field oriented control* menggunakan *Matlab/Simulink*.
2. Melakukan tuning parameter PI untuk mendapatkan performa *transient* dengan spesifikasi nilai *rise time* maksimal sebesar 2,162 detik , *settling time* maksimal sebesar 6,72 detik, dan *percent overshoot* maksimal 5%.

1.4. Manfaat

Manfaat tesis ini adalah sebagai berikut:

1. Memperoleh *power* dan *tourqe* motor BLDC yang sesuai dengan beban.
2. Menjadi referensi perkembangan BLDC *motor controller*.

1.5. Batasan Masalah

Batasan masalah dalam Tesis ini adalah sebagai berikut:

1. Jenis motor yang digunakan BLDC motor dengan *hall effect sensor*.
2. Tegangan yang digunakan 60 – 100 VDC.
3. Pemodelan *lookup table field oriented control* menggunakan *Matlab/Simulink*.

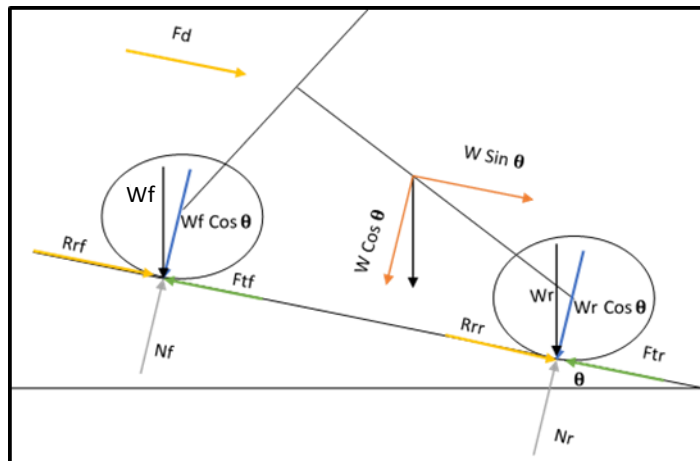
(halaman ini sengaja dikosongkan)

BAB 2

TINJAUAN PUSTAKA

2.1. Longitudinal Vehicle Dynamic

Dengan menganalisa *longitudinal vehicle dynamic* pada kendaraan dapat mengetahui gaya apa saja yang mempengaruhi pada saat kendaraan bergerak pada jalan yang datar maupun pada jalan yang memiliki sudut kemiringan tertentu. Dengan mendapatkan persamaan *longitudinal vehicle dynamic* dapat diketahui *power* dan torsi yang dibutuhkan kendaraan untuk bergerak pada suatu kondisi.



Gambar 2.1 *Free body diagram* sepeda motor 3S

Dimana:

- F_{tf} : Gaya dorong motor axle depan
- F_{tr} : Gaya dorong motor axle belakang
- R_{rf} : Rolling resistance roda depan
- R_{rr} : Rolling resistance roda belakang
- F_d : Gaya drag
- W_f : Gaya berat pada axle depan
- W_r : Gaya berat
- θ : Sudut kemiringan jalan

$$\Sigma F_x = m \cdot a \quad (2.1)$$

$$F_{tf} + F_{tr} - F_d - R_{rf} - R_{rr} - W \cdot \sin\theta = m \cdot a \quad (2.2)$$

$$\Sigma F_y = 0 \quad (2.3)$$

$$W \cos\theta - N = 0 \quad (2.4)$$

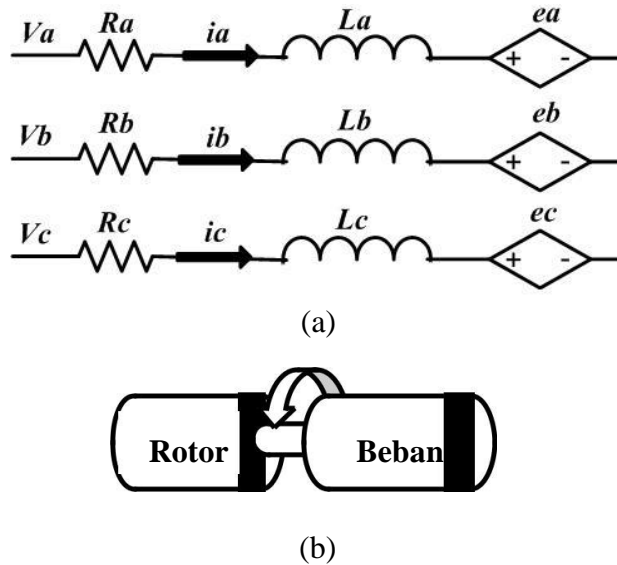
$$W \cos\theta = N \quad (2.5)$$

Pada pemodelan *longitudinal vehicle dynamic* akan mendapatkan data power (Watt) dan putaran motor (rad/s), sehingga torsi dapat ditemukan dengan persamaan berikut:

$$Torsi = \frac{Power \ (Watt)}{putaran \ motor \ (\frac{rad}{s})} \quad (2.6)$$

2.2. Motor BLDC

Motor BLDC sangat banyak digunakan dalam berbagai macam aplikasi, seperti pada kendaraan listrik karena mempunyai kinerja yang bagus dan torsi yang besar serta keandalan dan efisiensi yang tinggi (B. N. Kommula and V. R. Kota, 2015). Berdasarkan sebuah studi, disebutkan bahwa motor BLDC merupakan jenis motor yang mempunyai efisiensi paling tinggi dibandingkan dengan jenis motor yang lain (M. Singh and A. Garg, 2012). Pada prinsipnya motor BLDC sama dengan motor DC biasa. Namun perbedaan utamanya adalah tidak adanya sikat yang digunakan pada motor BLDC, sehingga tidak ada gesekan yang ditimbulkan antara sikat dan rotor sehingga motor BLDC mempunyai nilai efisiensi yang sangat tinggi. Selain itu penggunaan sikat pada motor DC biasa menyebabkan gesekan dengan rotor, sehingga menyebabkan nilai efisiensi berkurang, sikat mengalami keausan dalam jangka waktu lama, dan menimbulkan biaya perawatan yang sangat mahal. Berbeda dengan motor BLDC, biaya perawatan yang dibutuhkan sangatlah murah karena tidak mempunyai sikat. Sehingga dikatakan dalam sebuah studi bahwa motor BLDC merupakan motor yang paling cocok digunakan untuk aplikasi kendaraan listrik (B. N. Kommula and V. R. Kota, 2015).



Gambar 2.2. Rangkaian ekuivalen motor BLDC (a) model elektrik tiga fasa pada sisi stator (b) model mekanik pada sisi rotor (B. N. Kommula and V. R. Kota, 2015)

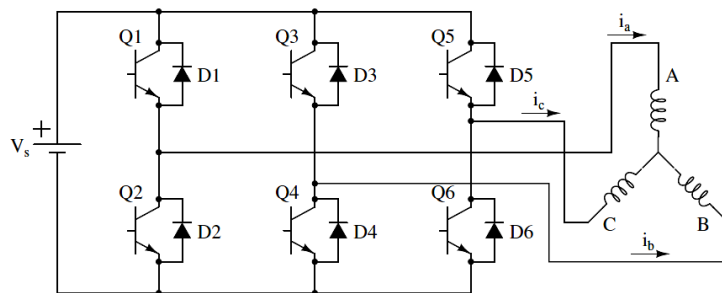
Gambar 2.2. menunjukkan motor BLDC terdiri dari 2 (dua) komponen utama, yaitu rotor yang berupa magnet permanen dan stator yang berupa kumparan yang terhubung pada sebuah *controller*. *Controller* ini dihubungkan ke suatu rangkaian *driver* agar motor BLDC dapat diisi menggunakan level tegangan yang diperlukan. Rangkaian *driver* motor BLDC ditunjukkan oleh Gambar 2.2.(a) *Controller* tersebut berfungsi untuk menggantikan fungsi dari sebuah komutator, yaitu menyediakan energi kumparan pada stator sesuai dengan urutan langkah komutasinya. Untuk mengetahui urutan langkah komutasi, diperlukan proses pendeteksian posisi rotor oleh *controller*. Proses pendeteksian posisi rotor ini bisa menggunakan sensor *hall* maupun *sensorless* (B. N. Kommula and V. R. Kota, 2015). Posisi sensor ini biasanya terpisah sejauh 60° atau 120° siklus elektris.

2.2.1. Model Matematika Motor BLDC

Motor BLDC merupakan jenis *Permanent Magnet Synchronous Motor* (PMSM) atau motor sinkron. Motor sinkron diklasifikasikan menjadi 2 (dua) kategori berdasarkan bentuk gelombang *Back Electro-Motive Force* (BEMF)

1. Satu berbentuk sinusoidal atau *Brushless Alternating Current (BLAC) motor*.
2. Berbentuk trapezoidal atau *Brushless Direct Current (BLDC) motor*.

Model Inverter 3 Phase dan BLDC Motor terlihat pada Gambar 2.3 berikut:



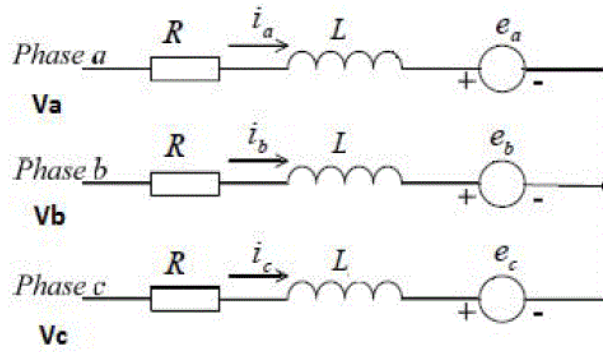
Gambar 2.3. Model Inverter 3 Phase dan BLDC Motor (Stefan B, 2005)

Controller BLDC menggunakan 3 *Half Bridge Mosfet* yang sering disebut dengan Inverter. Pada dasarnya motor BLDC memiliki prinsip kerja elektromagnetik, *sequential comutation* menimbulkan gaya-gaya magnet sehingga dapat berputar. Berikut 6 *step comutation* untuk BLDC Motor:

Tabel 2.1 *Six Step Comutation* (Stefan B, 2005).

Switching Interval	Seq. Number	Pos. Sensors			Switch Closed		Phase Current		
		H1	H2	H3			A	B	C
0° - 60°	0	1	0	0	Q1	Q4	+	-	OFF
60° - 120°	1	1	1	0	Q1	Q6	+	OFF	-
120° - 180°	2	0	1	0	Q3	Q6	OFF	+	-
180° - 240°	3	0	1	1	Q3	Q2	-	+	OFF
240° - 300°	4	0	0	1	Q5	Q2	-	OFF	+
300° - 360°	5	1	0	1	Q5	Q4	OFF	-	+

Pada prinsipnya motor BLDC sama dengan motor DC biasa, perbedaan motor BLDC dengan motor DC adalah pada motor BLDC terdapat 3-fasa belitan pada stator, sebagaimana yang ditunjukkan pada Gambar 2.4. Sehingga persamaan motor BLDC dapat dituliskan sebagai berikut:



Gambar 2.4. Ekuivalen 3 fasa motor BLDC (Stefan B, 2005)

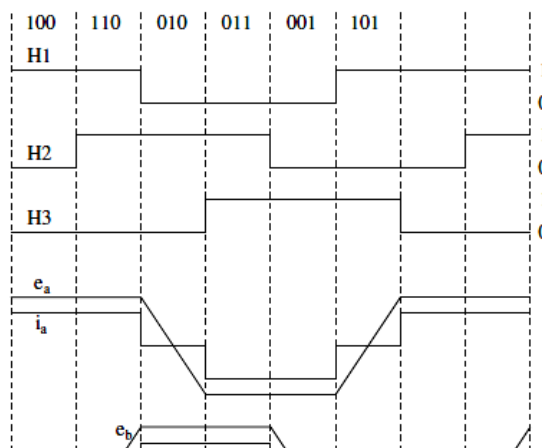
$$v_{ab} = R_s(i_a - i_b) + L_s \frac{d}{dt}(i_a - i_b) + e_a - e_b$$

$$v_{bc} = R_s(i_b - i_c) + L_s \frac{d}{dt}(i_b - i_c) + e_b - e_c \quad (2.7)$$

$$v_{ca} = R_s(i_c - i_a) + L_s \frac{d}{dt}(i_c - i_a) + e_c - e_a$$

$$T_e = T_L + B\omega_m + J \frac{d\omega_m}{dt} \quad (2)$$

Selain menggunakan sensor *Hall*, pendeteksian posisi rotor motor BLDC dapat dilakukan dengan mendeteksi tegangan BEMF pada rangkaian BLDC (Stefan B, 2005). Bentuk gelombang BEMF motor BLDC dapat digambarkan berupa fungsi sepotong-sepotong (*piece-wise function*) berbentuk gelombang *trapezoidal* di mana setiap nilai tegangan BEMF dan fasanya ditentukan berdasarkan posisi rotor. Gambar 2.5. Menunjukkan bentuk gelombang BEMF motor BLDC berdasarkan posisi rotor (Stefan B, 2005).



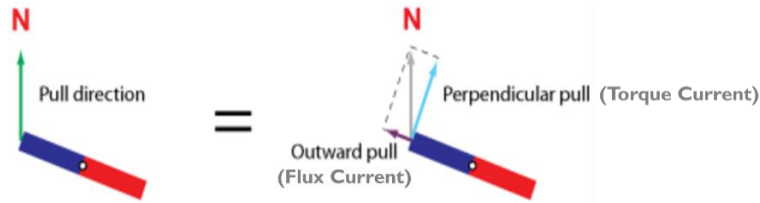
Gambar 2.5. Diagram bentuk gelombang BEMF motor BLDC dan arus pada masing-masing fasa berdasarkan posisi rotor (Stefan B, 2005)

2.2.2. Field Oriented Control

Field Oriented Control (FOC) adalah suatu metode *control* di mana arus stator motor listrik pada tiga fase diidentifikasi sebagai dua komponen ortogonal yaitu komponen yang mendefinisikan fluks *current* (I_d) dan komponen yang mendefinisikan *torque current* (I_q) sebagaimana yang ditunjukkan pada Gambar 2.7. Dengan sistem ini torsi dan fluks dapat diatur secara terpisah. Pada Pengaturan ini, kecepatan dari motor dimonitor oleh suatu sensor. Kecepatan motor yang dikembalikan kemudian dibandingkan dengan kecepatan referensi oleh suatu komparator. Bila ada *error*, kemudian *error* tersebut menjadi *input* dari kontroller. Selanjutnya kontroller memberikan sinyal kepada sistem FOC yang akan diteruskan kerangkaian inverter untuk mengubah tegangan dan arus motor sehingga diperoleh suatu torsi yang diinginkan. Perubahan torsi ini akan mengubah kecepatan motor sehingga bisa mendekati kecepatan referensi.

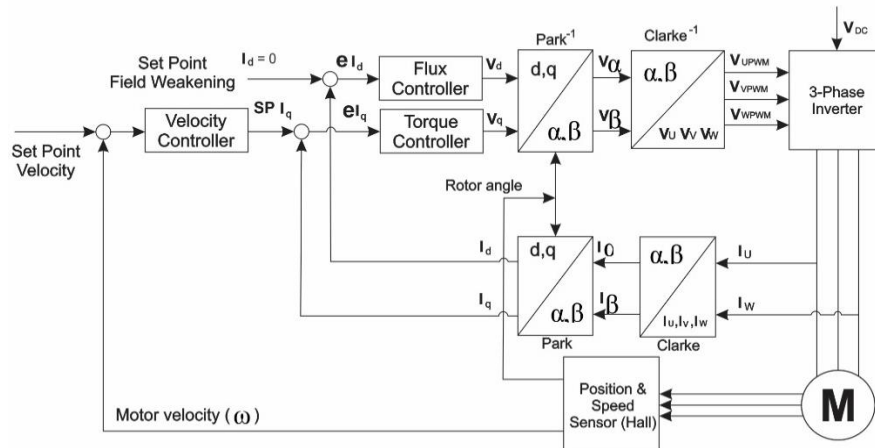
Dengan metode kontrol FOC, *current vector* dioptimalkan sehingga menghasilkan torsi maksimal pada setiap posisi rotor yang dimana tidak dapat dilakukan oleh metode kontrol sebelumnya yaitu metode kontrol trapezoidal. Metode kontrol trapezoidal memiliki kelemahan dimana torsi yang dihasilkan

tidak selalu optimal tergantung pada posisi rotor, sedangkan pada FOC torsi yang dihasilkan selalu dioptimalkan pada setiap posisi rotor.



Gambar 2.6. Arah medan magnet yang dihasilkan pada stator terhadap rotor

Pada FOC *flux current* selalu dibuat 0 sehingga hanya menyisakan *torque current* pada setiap posisi rotor sehingga torsi yang dihasilkan selalu optimal pada setiap posisi rotor seperti yang ditunjukkan Gambar 2.6..



Gambar 2.7. Block Diagram *Field Oriented Control* (Jacob dan Chitra, 2017)

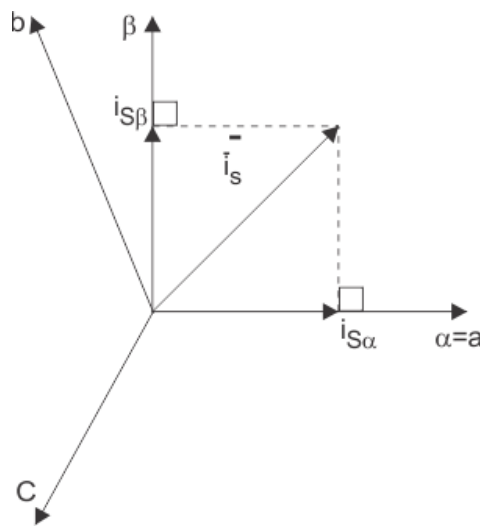
Dilihat dari skema atau diagram blok sistem pengaturan kecepatan motor induksi dengan FOC, inverter beroperasi sebagai sumber arus tiga fasa yang kemudian akan menjadi sumber yang akan menggerakkan motor induksi tiga fasa (Jacob dan Chitra, 2017).

Transformasi Clarke di dalam FOC motor induksi digunakan untuk mentransformasikan arus stator tiga fasa (i_a , i_b , dan i_c) pada bidang stasioner ke arus stator ortogonal dua fasa (i_α dan i_β) pada bidang ortogonal sebagaimana ditunjukkan pada Gambar 2.8. (Shah, 2017). Sedangkan Transformasi Park digunakan untuk mentransformasikan arus stator (i_α dan i_β) ke arus stator dua fasa (i_{ds} dan i_{qs}) pada bidang putar (*rotating reference frame*) seperti yang ditunjukkan

Gambar 2.9. Untuk mentransformasikan arus stator dari sistem tiga fasa (a, b, dan c) ke sistem dua fasa ortogonal (α dan β), serta mengacu pada persamaan *decoupled* di atas, maka secara matematis persamaan Transformasi Clarke dapat dirumuskan kembali sebagai berikut :

$$i_{\alpha} = i_a \quad (2.9)$$

$$i_{\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \quad (2.10)$$



Gambar 2.8. *Vector diagram of Clarke transformation* (www.mathworks.com)

Inverse Transformasi Clarke digunakan untuk mentransformasi balik dari komponen α dan β ke komponen a, b, dan c melalui persamaan berikut :

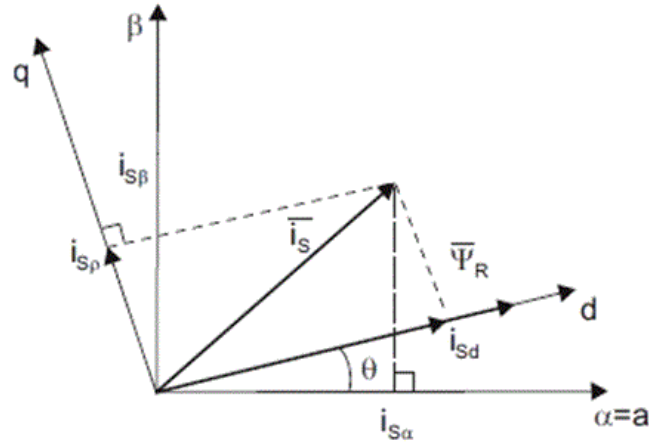
$$i_a = i_{\alpha}$$

$$i_b = -\frac{1}{2} i_{\alpha} + \frac{\sqrt{3}}{2} i_{\beta} \quad (2.11)$$

$$i_c = -\frac{1}{2} i_{\alpha} - \frac{\sqrt{3}}{2} i_{\beta}$$

Untuk mentransformasikan arus stator dari sistem dua fasa ortogonal (α dan β) ke sistem dua fasa (d dan q) menggunakan Transformasi Park, secara matematis dapat dirumuskan sebagai berikut :

$$\begin{aligned}
 i_{ds} &= i_{\alpha} \cdot \cos \theta + i_{\beta} \cdot \sin \theta \\
 i_{dq} &= -i_{\alpha} \cdot \sin \theta + i_{\beta} \cdot \cos \theta
 \end{aligned}
 \tag{2.12}$$



Gambar 2.9. Vector diagram of Park Transformation (www.mathworks.com)

Inverse Transformasi Part digunakan untuk mentransformasi balik dari komponen d dan q ke komponen α dan β melalui persamaan berikut :

$$\begin{aligned}
 i_{\alpha} &= i_{ds} \cdot \cos \theta - i_{qs} \cdot \sin \theta \\
 i_{\beta} &= i_{ds} \cdot \sin \theta + i_{qs} \cdot \cos \theta
 \end{aligned}
 \tag{2.13}$$

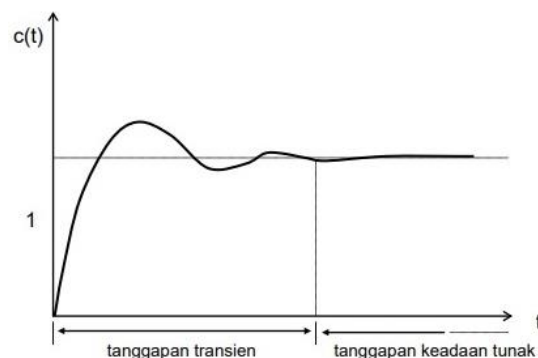
2.2.3. Karakteristik respon waktu (*Time Response*)

Karakteristik *time response* adalah karakteristik respon yang spesifikasi performanya didasarkan pada pengamatan bentuk respon *output* sistem terhadap berubahnya waktu. Secara umum spesifikasi performa respon waktu dapat dibagi atas dua tahapan pengamatan yaitu spesifikasi respon *transient* dan spesifikasi respon *steady state* seperti yang ditunjukkan Gambar 2.10. Spesifikasi respon *transient* adalah spesifikasi respon sistem yang diamati mulai saat terjadi perubahan sinyal *input*/gangguan/beban sampai respon masuk dalam keadaan *steady state*. Tolak ukur yang digunakan untuk mengukur kualitas respon *transient* antara lain *rise time*, *delay time*, *peak time*, *settling time*, dan *percent overshoot*. Spesifikasi respon *steady state* adalah spesifikasi respon sistem yang dimulai saat respon masuk pada keadaan *steady state* sampai waktu tak terbatas.

Tolak ukur yang digunakan untuk mengukur kualitas respon *steady state* diantaranya *% error steady* berupa *error* posisi, *error* kecepatan, dan *error* percepatan.

Tolak ukur yang digunakan untuk mengukur kualitas respon transient antara lain *rise time*, *delay time*, *peak time*, *settling time*, dan *percent overshoot*. Dimana *rise time* merupakan waktu yang diperlukan untuk perubahan dari 10% menjadi 90% nilai akhir, *delay time* merupakan waktu yang diperlukan untuk mencapai setengah dari nilai akhir pada waktu pertama kali, *peak time* merupakan waktu yang diperlukan untuk mencapai *peak* pertama, atau maksimum, *settling time* merupakan besarnya waktu yang dibutuhkan sistem untuk mengkonvergenkan *steady state*, dan *percent overshoot* merupakan jumlah gelombang yang melakukan *overshoot* terhadap *steady state* atau nilai akhir pada waktu puncak, diekspresikan dalam bentuk prosentase terhadap nilai *steady state*.

Risetime, *settling time*, dan *percent overshoot* memberikan informasi mengenai kecepatan dan kualitas respon transient. Besaran-besaran ini dapat membantu perancang untuk mencapai kecepatan yang diinginkan tanpa osilasi atau *overshoot* yang berlebihan.

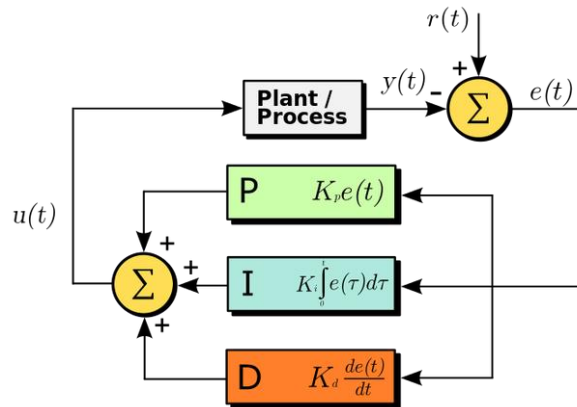


Gambar 2.10. Respon Transient dan Respon *Steady State* (<http://elektro.um.ac.id/>)

2.2.4. Kontrol PID

PID (*Proportional Integral Derivative*) merupakan salah satu jenis kontrol yang paling banyak digunakan oleh industri sekarang ini. PID terdiri dari tiga komponen berupa Proporsional (P), Integral (I), dan Derivatif (D). Ketiga komponen ini saling melengkapi satu sama lain, sehingga kelemahan-kelemahan

pada salah satu komponen dapat ditutupi oleh komponen yang lain. Komponen I dan D tidak dapat berdiri sendiri dan selalu dikombinasikan dengan komponen P, menjadi kontrol PI atau PID. PID bekerja dengan membandingkan kesalahan / *error* yang merupakan selisih dari *process variable* dan *set point*. Kemudian hasil perbandingan itu digunakan sebagai *input signal* menjadi *output signal* ($u(t)$).



Gambar 2.11. *Diagram PID Controller* (www.wikipedia.com)

Berdasarkan Gambar 2.11. diperoleh persamaan *PID Controller* sebagai berikut:

$$mv(t) = K_p e(t) + K_i \int_0^1 e(t) dt + K_d \frac{de(t)}{dt} \quad (2.14)$$

Dimana:

$mv(t)$ = *output* dari pengontrol PID atau *Manipulated Variable*

K_p = konstanta Proporsional

T_i = konstanta Integral

T_d = konstanta Detivatif

$e(t)$ = *error* (selisih antara set point dengan level aktual)

A. Proportional (P)

Penggunaan kontrol P memiliki berbagai keterbatasan karena bersifat statis. Walaupun demikian dalam aplikasi-aplikasi dasar yang sederhana kontrol P ini cukup mampu untuk memperbaiki respon transient khususnya *rise time* dan *setting time*. Pengontrol proporsional

memiliki *output* yang sebanding/proporsional dengan besarnya *error*.

Ciri-ciri pengontrol proporsional:

1. Jika nilai K_p rendah, pengontrol proporsional hanya mampu melakukan koreksi kesalahan yang kecil sehingga akan menghasilkan respon sistem yang lambat (menambah *rise time*).
2. Jika nilai K_p dinaikkan, respon/tanggapan sistem akan semakin cepat mencapai keadaan setimbang (mengurangi *rise time*).
3. Jika nilai K_p besar mengakibatkan sistem bekerja tidak stabil atau respon sistem akan berosilasi.
4. Nilai K_p dapat diatur sedemikian sehingga mengurangi *steady state error*, tetapi tidak dihilangkan.

B. Integratif (I)

Pengontrol Integral berfungsi menghasilkan respon sistem yang memiliki *Error Steady State* = 0. Jika sebuah pengontrol tidak memiliki unsur integrator, pengontrol proporsional tidak mampu memastikan nilai *output* sistem dengan *Error Steady State* = 0. Ciri-ciri pengontrol Integral:

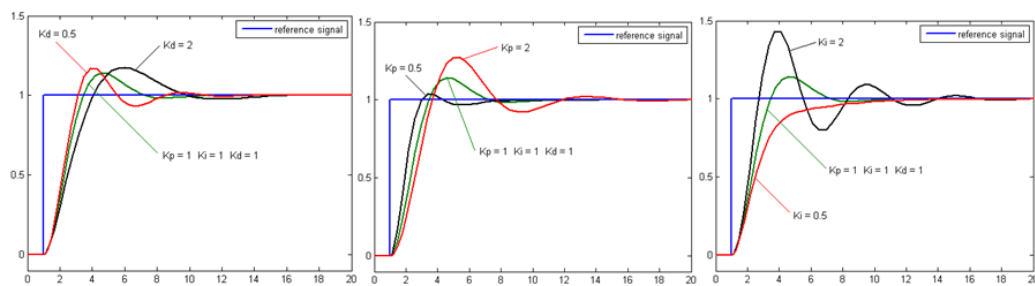
1. *Output* pengontrol integral membutuhkan selang waktu tertentu, sehingga pengontrol integral cenderung memperlambat respon.
2. Ketika *error* bernilai nol, *output* pengontrol akan bertahan pada nilai sebelumnya.
3. Jika *error* tidak bernilai nol, *output* akan menunjukkan kenaikan atau penurunan yang dipengaruhi oleh besarnya *error* dan nilai K_i .
4. Konstanta integral K_i yang bernilai besar akan mempercepat hilangnya *offset*. Tetapi semakin besar nilai konstanta K_i akan mengakibatkan peningkatan osilasi dari sinyal keluaran pengontrol.

C. Derivatif (D)

Kontrol *Derivative* hanya berubah saat ada perubahan *error* sehingga saat *error* statis kontrol ini tidak bereaksi, hal ini pula yang menyebabkan kontroler Derivatif tidak dapat digunakan. Ciri-ciri pengontrol derivatif:

1. Pengontrol tidak dapat menghasilkan *signal output* jika tidak ada perubahan pada *signal input* (berupa perubahan *error*)
2. Jika *error* berubah terhadap waktu, maka *signal output* yang dihasilkan bergantung pada nilai K_d dan laju perubahan *error*.
3. Peningkatan nilai K_d mengakibatkan nilai stabilitas sistem meningkatkan dan mengurangi *overshoot*.

Karakteristik pengontrol PID sangat dipengaruhi oleh kontribusi besar dari ketiga parameter P, I, dan D. Penyetelan konstanta K_p , K_i dan K_d akan mengakibatkan penonjolan sifat dari masing-masing elemen. Satu atau dua dari ketiga konstanta tersebut dapat disetel lebih menonjol dibanding yang lain. Konstanta yang menonjol itulah akan memberikan kontribusi pengaruh pada respon sistem secara keseluruhan.



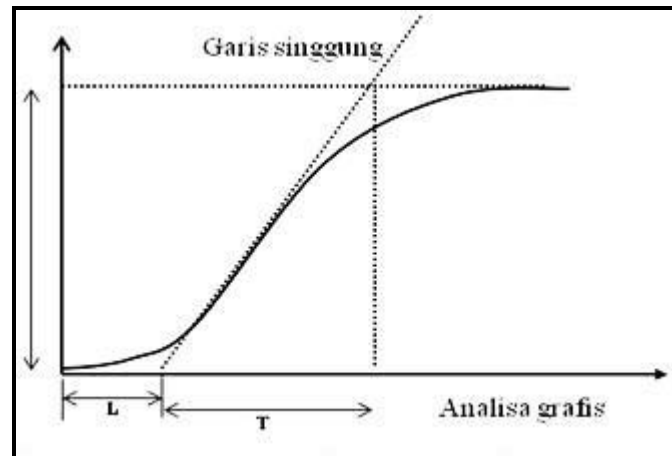
Gambar 2.12. Respon dari sistem terhadap perubahan K_p , K_i dan K_d (www.wikipedia.com)

2.2.5. Penentuan Nilai Konstanta PID Menurut Ziegler Nichols

Aspek yang sangat penting dalam desain kontroler PID ialah penentuan parameter kontroler PID sehingga sistem *close loop* memenuhi kriteria performansi yang diinginkan. Hal ini disebut juga dengan *tuning* kontroler. Terkadang pemodelan matematis suatu *plant* susah untuk dilakukan. Jika hal ini terjadi maka perancangan kontroler PID secara analitis tidak mungkin dilakukan sehingga perancangan kontroler PID harus dilakukan secara eksperimental.

Ziegler – Nichols mengusulkan aturan untuk menentukan nilai K_p , K_i dan K_d berdasarkan pada karakteristik tanggapan peralihan dari *plant* yang diberikan

seperti yang ditunjukkan Table 2.2. Metode pertama Ziegler – Nichols menentukan nilai Kp, Ki, dan Kd:



Gambar 2.13. Kurva Analisa Grafis Ziegler Nichols(www.elektroindonesia.com)

Aturan perpotongan garis lurus terjadi pada kondisi linier dari kurva S respon sistem seperti yang ditunjukkan Gambar 2.13. Ketepatan dalam pengambilan perpotongan ini sangatlah penting karena menentukan parameter T dan L yang menjadi acuan dari *controller*.

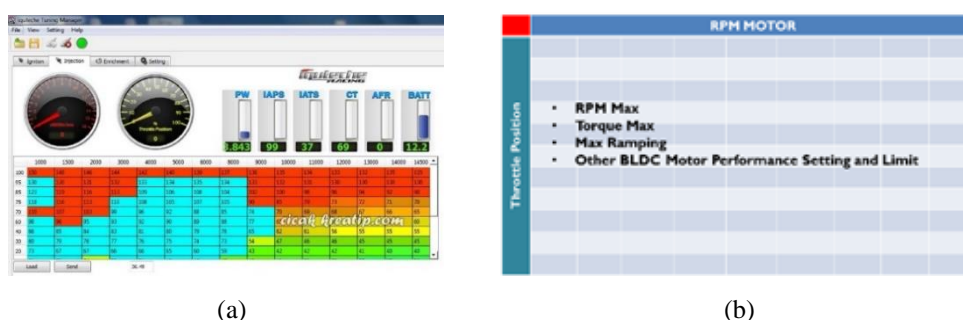
Tabel 1.2 Formula Ziegler Nichols

Tipe Pengendali	Kp	Ki	Kd
P	$\frac{T}{L}$	∞	0
PI	$0.9 \frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2 \frac{T}{L}$	2L	0.5L

2.3. Look up Table

Teknologi kendaraan listrik bergantung pada perangkat yang dikontrol secara elektronik untuk memantau dan mengoperasikan sistem kendaraan. Pada penelitian ini *look up table* dipilih agar *delivery torque* dapat diatur sesuai kebutuhan dan mempermudah dalam proses *setting parameter controller* dan *setting limit controller*. Input dari *look up table* ini adalah TPS (*Throttle Position*

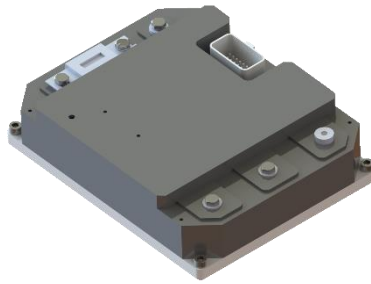
Sensor) dan putaran motor (rpm) dan *output* dari *look up table* ini adalah arus *reference* yang mendeskripsikan torsi. Pada penelitian ini menggunakan *look up table* dengan 2 sumbu yaitu *input* rpm pada sumbu X dan *input throttle position* pada sumbu Y, dimana kedua sumbu tersebut mempresentasikan sebagai beban yang sedang diterima oleh kendaraan seperti yang ditunjukkan Gambar 2.14.(b). *Lookup table* dapat mempermudah dalam *setting parameter* sistem *control* seperti rpm maksimal, torsi maksimal dan *setting limit* dari *controller* tersebut.



Gambar 2.14. *Look up Table* (a) *look up table* pada ECU sepeda motor konvensional (b) rancangan *look up table* pada sistem *control* kendaraan listrik (www.iquteche.com)

2.4. Controller motor BLDC

Pada penelitian ini menggunakan *controller* motor BLDC Wiksa Mark 1.2 seperti yang ditunjukkan Gambar 2.15 yang nantinya dipakai untuk mengimplementasikan hasil pemodelan ke *plant* yang sebenarnya. Kunci keberhasilan sebuah *controller* terletak pada pembacaan *feedback* dari motor BLDC tersebut. *Feedback* yang penting untuk diperhatikan antara lain pembacaan *hall effect sensor*, pembacaan BEMF, pembacaan *throttle*, dan pembacaan arus pada fasa. Parameter utama tersebut sangat mempengaruhi keberhasilan dalam mengontrol motor BLDC. Selanjutnya parameter lain seperti tegangan baterai, *temperature heatsink*, temperatur motor digunakan sebagai fitur proteksi agar *controller* maupun motor tidak mengalami kerusakan secara permanen.



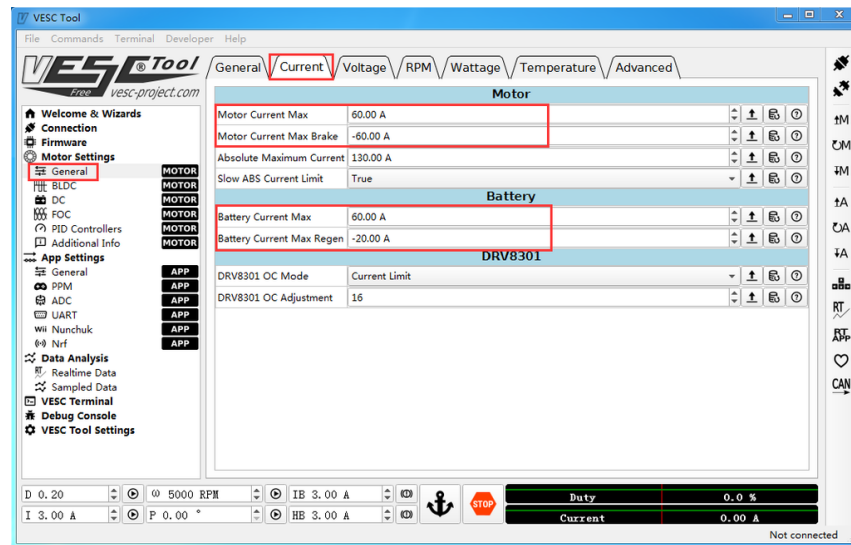
Gambar 2.15. *Controller Wiksa Mark 1.2*

Controller ini menggunakan STM32F405RGTx sebagai *microcontroller*. *Microcontroller* ini memiliki level tegangan 3.3Volt yang termasuk *microcontroller* dengan *low consumption energy*.

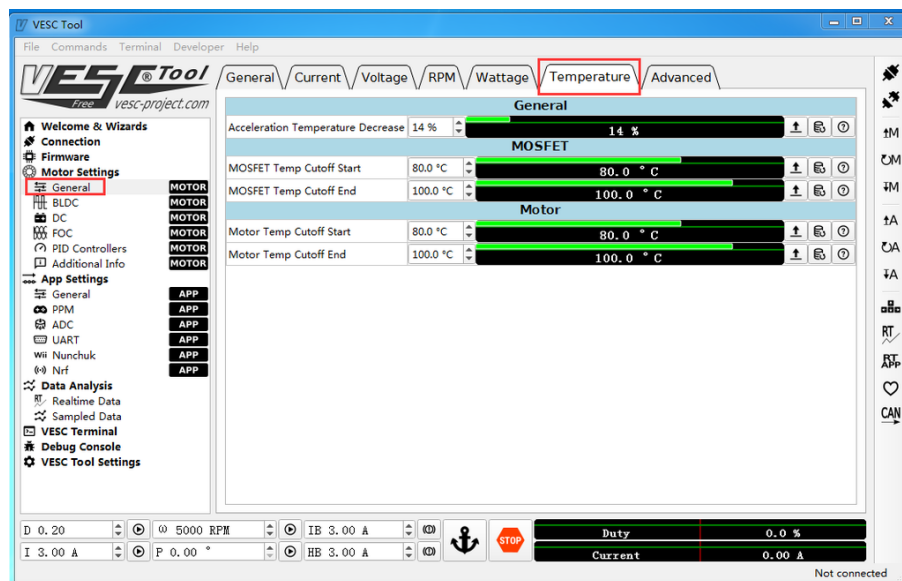
2.5. Vedder Electronic Speed Control.

Vedder Electronic Speed Controller (VESC) merupakan suatu proyek *open source* yang dikembangkan oleh seorang teknisi Swedia bernama Benjamin Vedder. *VESC* adalah sirkuit elektronik yang dapat mengontrol dan mengatur kecepatan motor listrik. Pada awalnya Benjamin Vedder mengembangkan *electronic speed controller* yang diaplikasikan pada *skateboard* listrik. Namun dengan pengembangan teknologi yang dilakukan Benjamin Vedder pada *electronic speed control*, saat ini *VESC* dapat diaplikasikan tidak hanya untuk *skateboard* listrik melainkan sepeda listrik, robot, *scooters* listrik, *go-kart*, dan lain-lain.

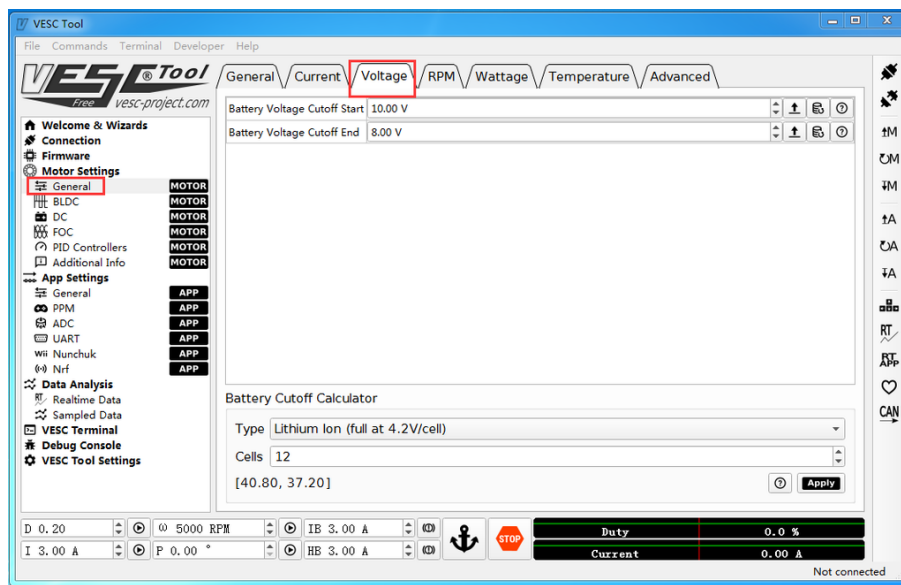
Sistem elektronik *VESC* diproteksi melalui pemrograman dengan cara membatasi berbagai parameter yang ada pada *controller*. Pada *VESC* terdapat beberapa parameter *controller* yang dapat diatur seperti batas jumlah arus dari baterai ke motor seperti ditunjukkan pada Gambar 2.16, suhu maksimum yang diperbolehkan saat *electronic speed control* beroperasi seperti ditunjukkan pada Gambar 2.17, serta nilai tegangan yang masuk dari minimum hingga maksimum untuk memberi proteksi pada baterai agar tidak terjadi *over discharge* seperti yang ditunjukkan pada Gambar 2.18 (Ken Chen, 2018).



Gambar 2.16. Pengaturan batas jumlah arus dari baterai ke motor(Ken Chen, 2018)



Gambar 2.17. Pengaturan suhu maksimum pada VESC (Ken Chen, 2018)



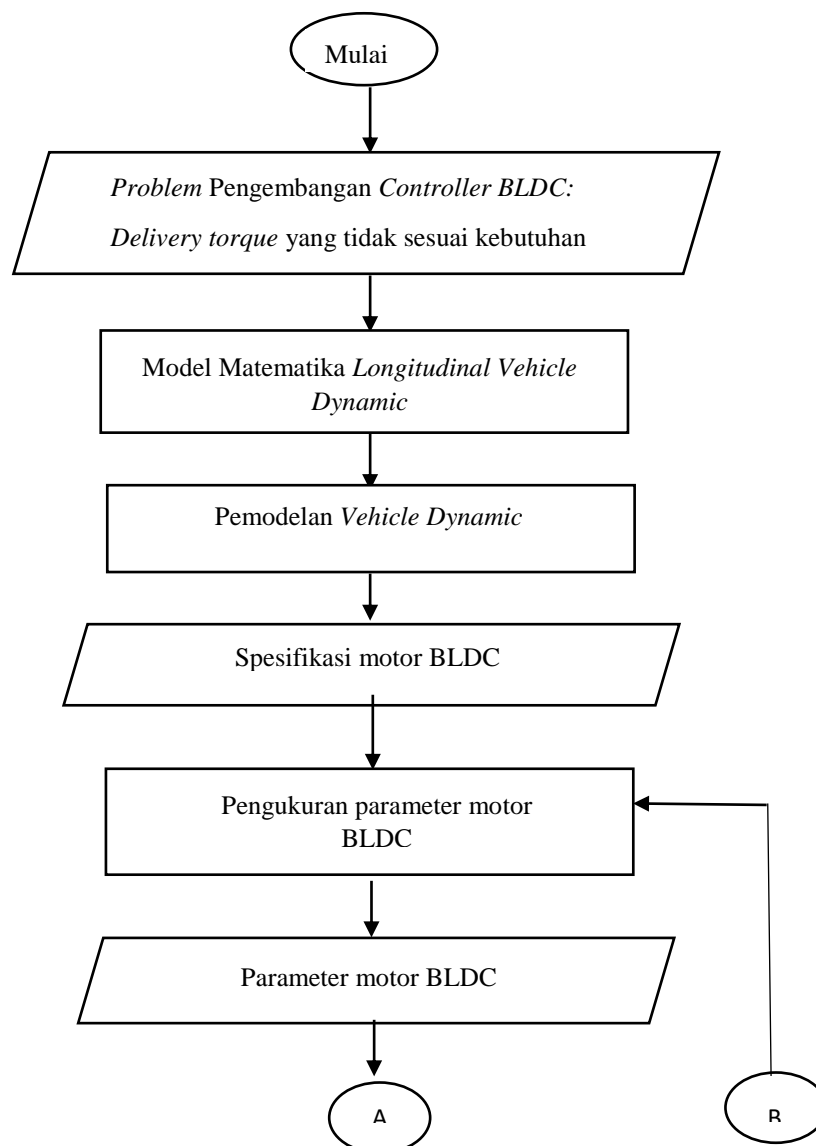
Gambar 2.18. Pengaturan tegangan input minimum dan maksimum pada VESC (Ken Chen, 2018).

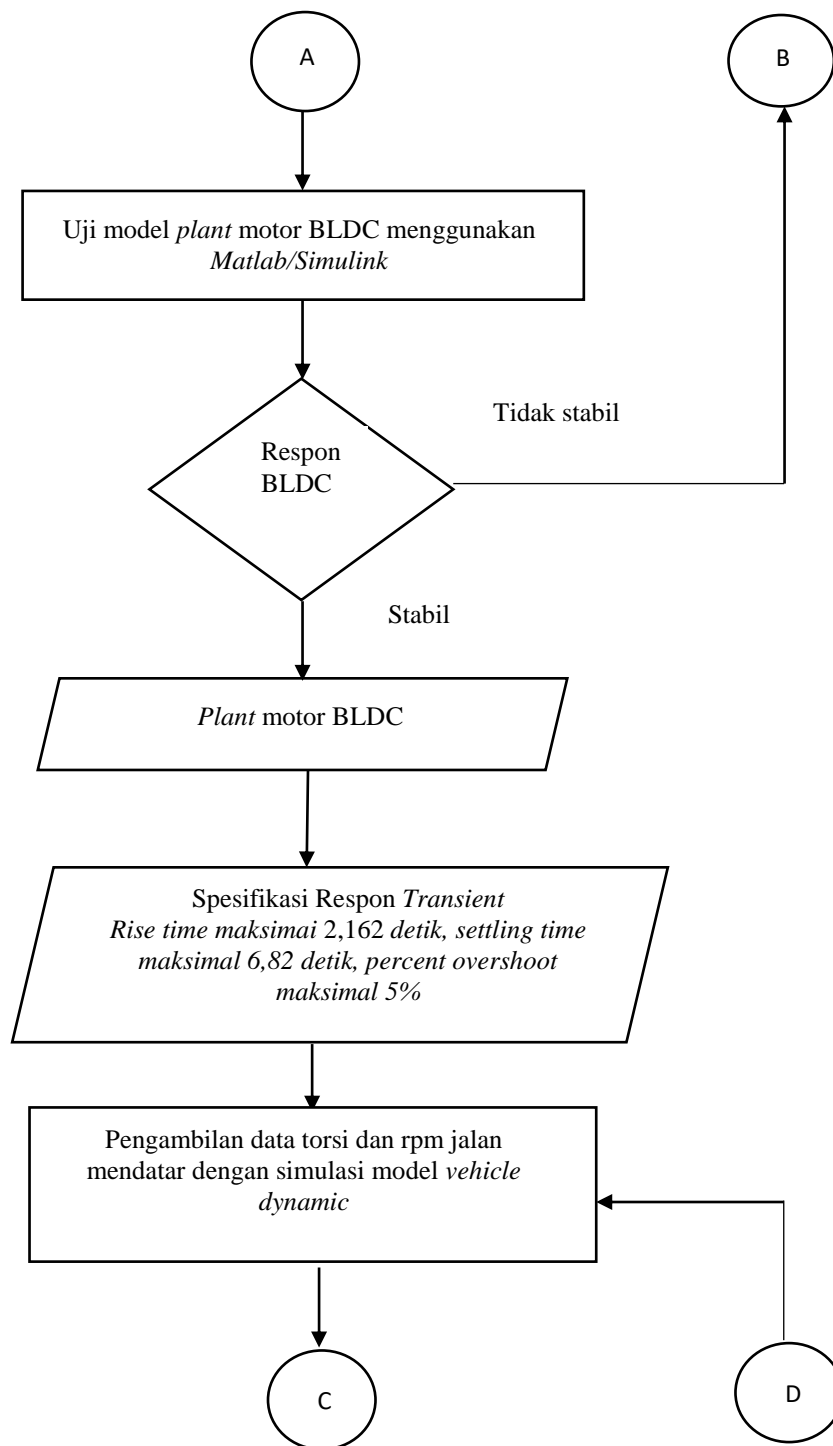
BAB 3

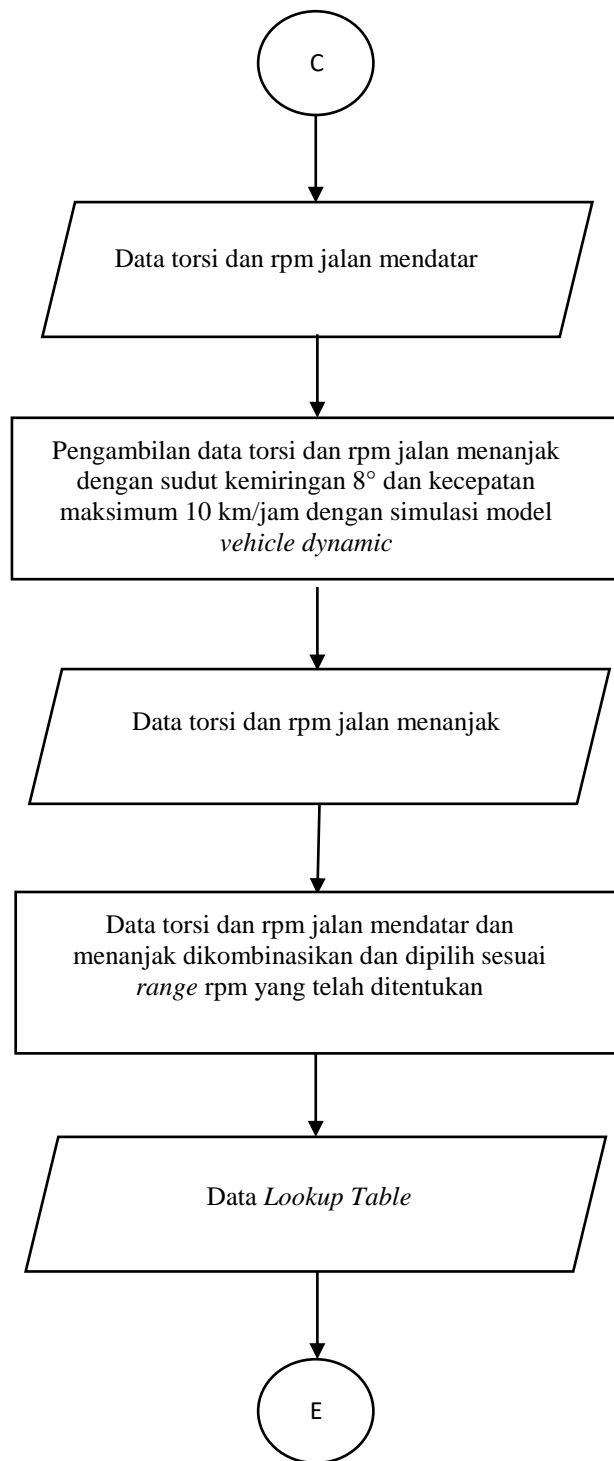
METODE PENELITIAN

3.1. Rancangan Penelitian

Alur pelaksanaan tesis ini sebagai berikut:

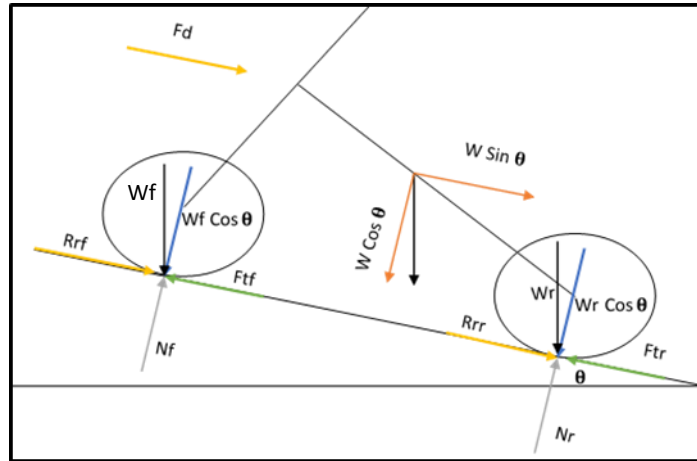






3.2. Perhitungan *Vehicle Dynamic* Sepeda Motor Listrik

Berikut perhitungan *vehicle dynamic* sepeda motor dalam tesis ini:



Gambar 3.1. *Free Body Diagram* Sepeda Motor Listrik

Dimana:

- F_{tf} : Gaya dorong motor axle depan
- F_{tr} : Gaya dorong motor axle belakang
- R_{rf} : *Rolling resistance* roda depan
- R_{rr} : *Rolling resistance* roda belakang
- F_d : Gaya drag
- W_f : Gaya berat pada axle depan
- W_r : Gaya berat
- θ : Sudut kemiringan jalan

Berdasarkan Hukum Newton I di dapatkan persamaan gaya pada sumbu x dan sumbu y sebagai berikut:

$$\Sigma F_x = m \cdot a \quad (3.1)$$

$$F_{tf} + F_{tr} - F_d - R_{rf} - R_{rr} - W \cdot \sin \theta = m \cdot a \quad (3.2)$$

$$\Sigma F_y = 0 \quad (3.3)$$

$$W \cos \theta - N = 0 \quad (3.4)$$

$$W \cos \theta = N \quad (3.5)$$

Dimana:

$$F_t = F_{tf} + F_{tr} \quad (3.6)$$

$$F_d = \frac{1}{2} \rho \cdot C_d \cdot A \cdot V^2 \quad (3.7)$$

$$R_{rt} = R_{rf} + R_{rr} \quad (3.8)$$

$$R_{rt} = \mu_s \cdot N \quad (3.9)$$

$$R_{rt} = \mu_s \cdot W \cos \theta \quad (3.10)$$

Maka didapatkan persamaan gaya untuk menggerakkan sepeda motor sebagai berikut:

$$F_t - F_d - R_{rt} - W \cdot \sin \theta = m \cdot a \quad (3.11)$$

$$F_t = F_d + R_{rt} + W \cdot \sin \theta + m \cdot a \quad (3.12)$$

$$F_t = \frac{1}{2} \rho \cdot C_d \cdot A \cdot V^2 + \mu_s \cdot W \cos \theta + W \cdot \sin \theta + m \cdot a \quad (3.13)$$

Dimana:

$$\rho = 1,23 \text{ Kg/m}^3$$

$$C_d = 0.77$$

$$A = 0,44 \text{ m}^2$$

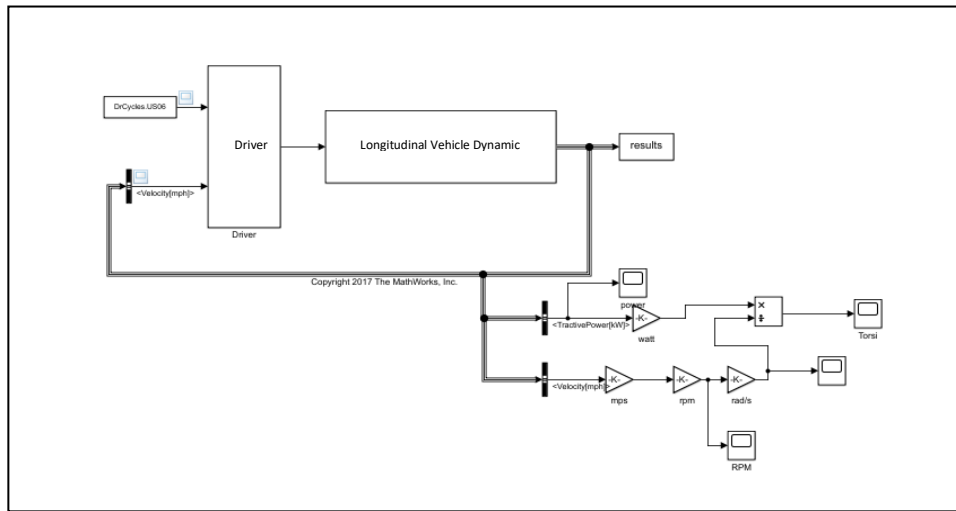
$$\mu_s = 0,004$$

$$m = 220 \text{ kg}$$

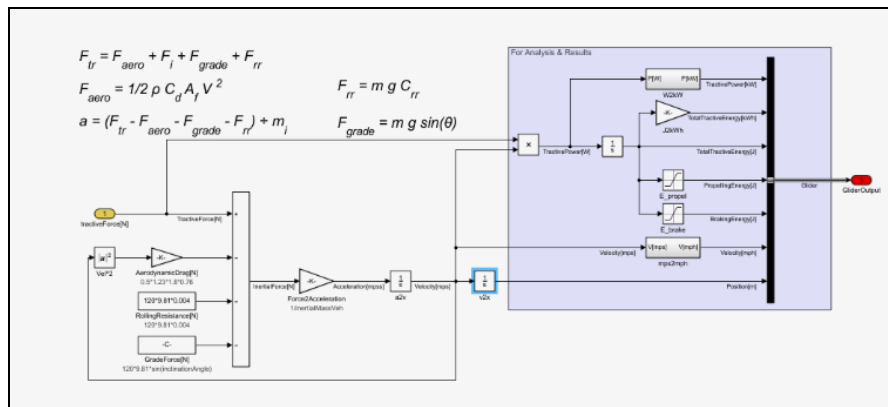
$$g = 9,8 \text{ m/s}^2$$

3.2.1. Pemodelan *Vehicle Dynamic*

Pada tahap awal dilakukan pemodelan *longitudinal vehicle dynamic* dengan menggunakan *vehicle model* pada *simulink* untuk mendapatkan power dan torsi sebagai spesifikasi dari motor listrik sebagai penggerakannya. Nilai parameter pada *longitudinal vehicle dynamic* dimasukkan pada *block diagram longitudinal vehicle dynamic* yang mengimplementasikan sepeda motor listrik pada pemodelan.

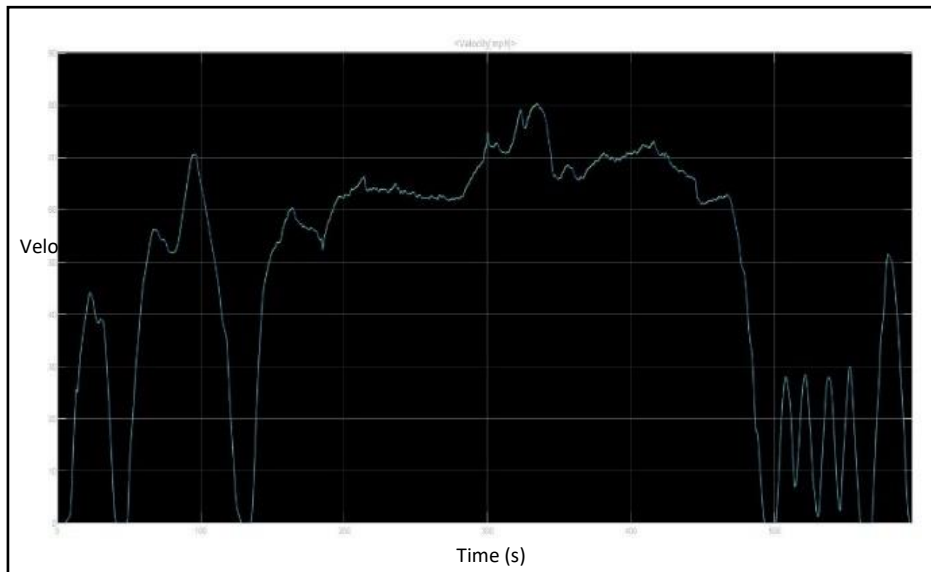


Gambar 3.2. Block Diagram Vehicle Model

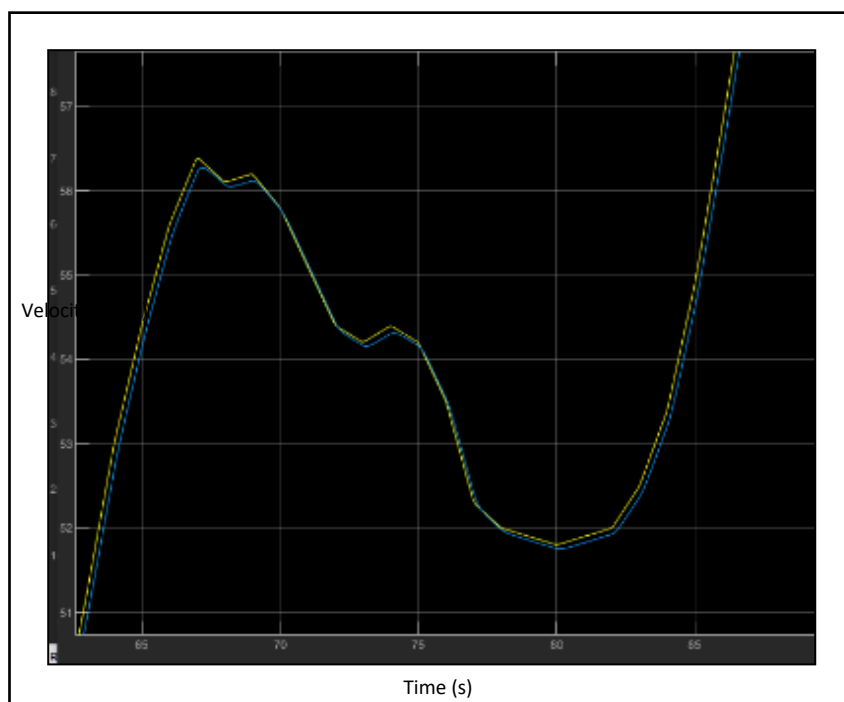


Gambar 3.3. Block diagram Longitudinal Vehicle Dynamic

Pada Gambar 3.2 merupakan *block diagram vehicle modeling* yang menggunakan dasar *block diagram racing lounge vehicle dynamic modeling* yang tersedia pada *Matlab/Simulink* dan Gambar 3.3 adalah *subsystem block diagram* dari *block diagram vehicle model* dimana semua parameter dari *longitudinal vehicle dynamic* dimasukkan pada *subsystem block diagram* ini.

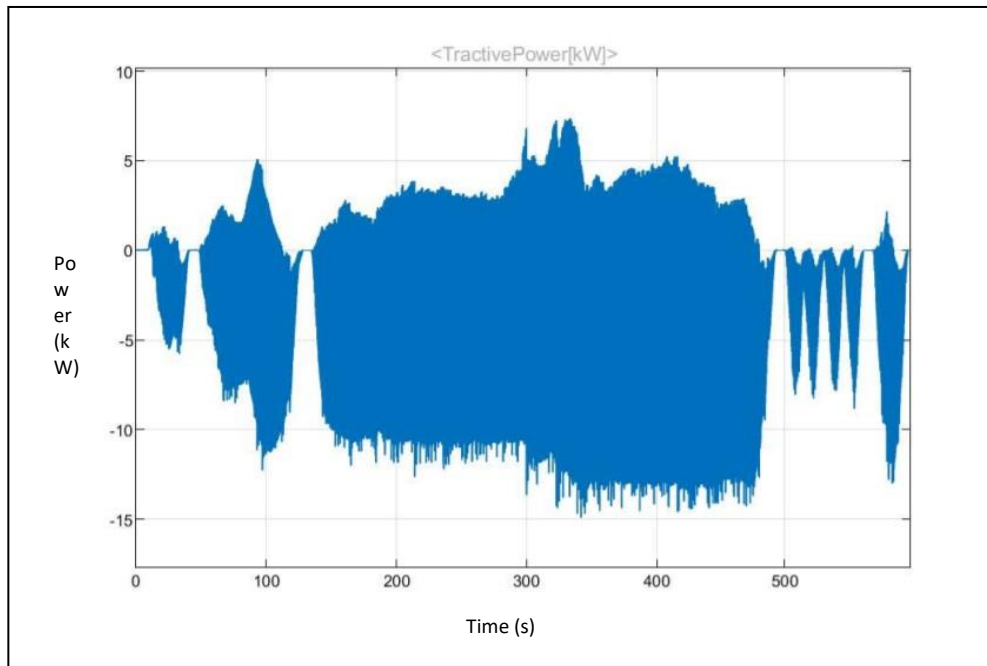


Gambar 3.4. Hasil dari *Vehicle Model*

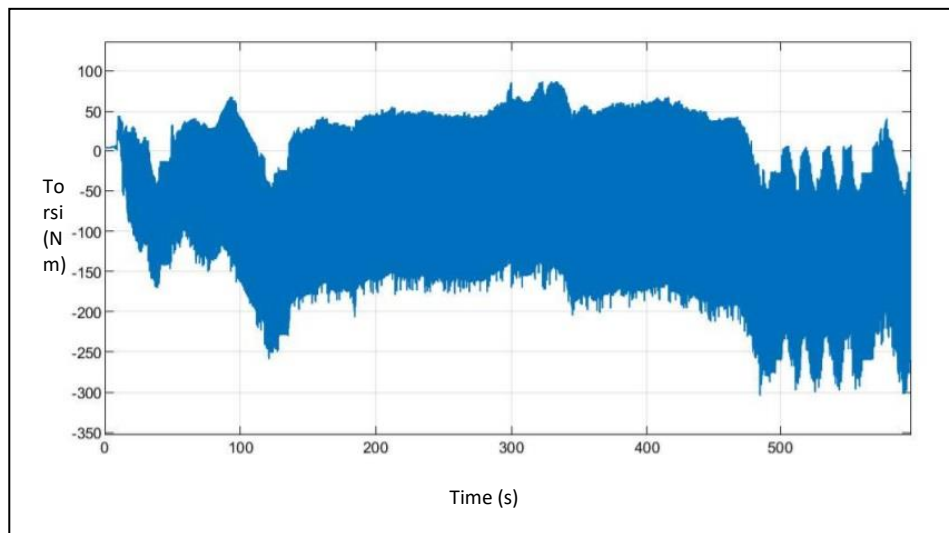


Gambar 3.5. Zoom hasil dari *Vehicle Model*

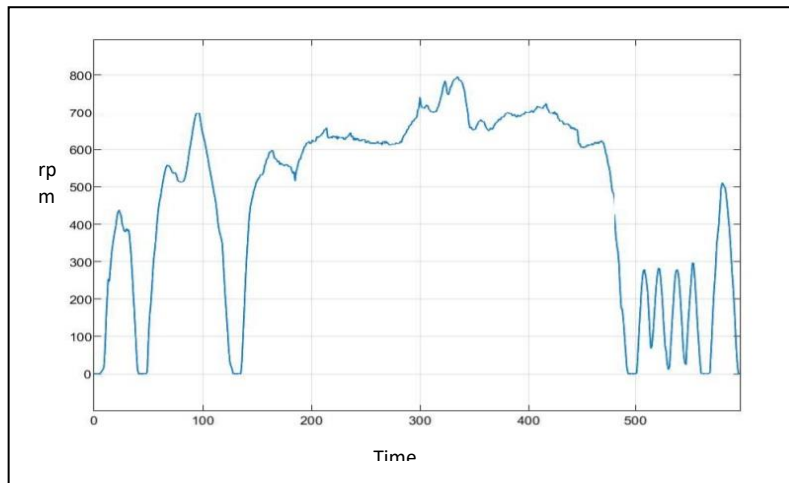
Gambar 3.4 dan Gambar 3.5 adalah hasil dari *vehicle model* dengan input *drive cycle* US06. Garis kuning merupakan *setpoint* yang diberikan dari *drive cycle* dan garis biru merupakan respon dari *vehicle model*.



Gambar 3.6. *Power* hasil dari *Vehicle model*



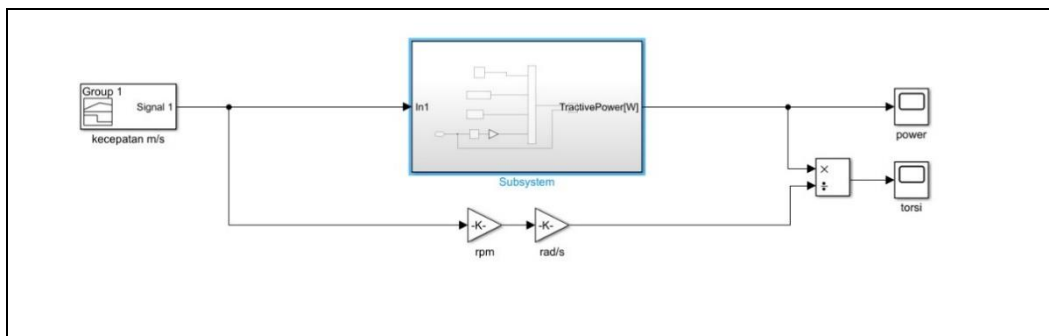
Gambar 3.7. *Torsi* hasil dari *Vehicle model*



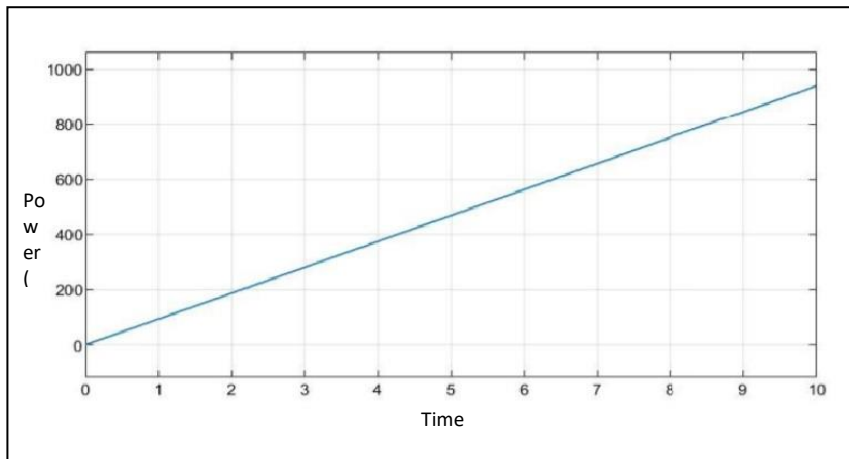
Gambar 3.8. Rpm hasil dari *Vehicle model*

Pada hasil pemodelan *vehicle dynamic* didapatkan power maksimal sebesar 7Kw, torsi maksimal 80Nm, dan rpm maksimal sebesar 800 rpm sebagai salah satu parameter dalam menentukan spesifikasi BLDC motor yang akan digunakan

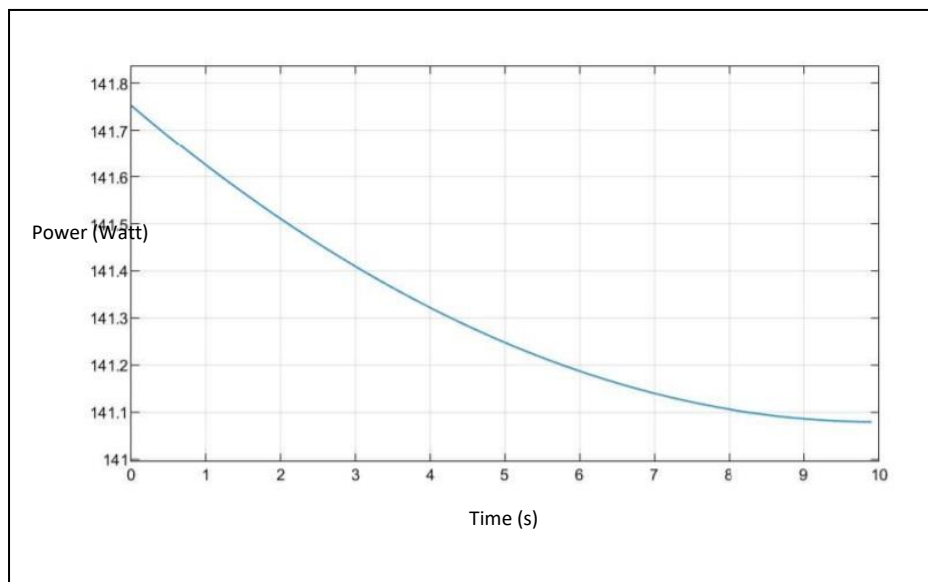
Pemodelan *vehicle dynamic* seperti yang ditunjukkan Gambar 3.9 pada tanjakan dilakukan dengan sudut kemiringan jalan sebesar 8° dengan kecepatan 0-10 km/jam dan percepatan sebesar $0,13 \text{ m/s}^2$ maka akan didapatkan data untuk menunjang perancangan *look up table*.



Gambar 3.9. Pemodelan *vehicle dynamic* pada sudut kemiringan jalan 8°



Gambar 3.10. Power pada kecepatan 0-10 km/jam dengan sudut kemiringan jalan 8°



Gambar 3.11. Torsi pada kecepatan 0-10 km/jam dengan sudut kemiringan jalan 8° .

Pada pemodelan *vehicle dynamic* dan pemodelan vehicle dynamic dengan sudut kemiringan 8° didapat data power, torsi, dan rpm sebagai salah satu pertimbangan dalam menentukan spesifikasi motor BLDC. Dari hasil pemodelan didapatkan spesifikasi motor BLDC yang ditransmisikan dengan *fix ratio* 1:6 yaitu motor BLDC dengan power 5kw *rated*, torsi 24 N.m, dan 3000rpm. Setelah spesifikasi motor BLDC ditentukan, langkah selanjutnya mengukur parameter motor BLDC tersebut.

Pada Gambar 3.12 dapat dilihat hasil dari uji performa sepeda motor listrik 3S sebagai validasi dari penentuan spesifikasi motor yang akan digunakan. Pada dynamometer terukur power motor 5,8kw pada rpm 2872 dan torsi sebesar 20,8 Nm pada rpm 1975 dengan menggunakan fix rasio sebesar 1:4,4.



Gambar 3.12. Hasil uji performa motor listrik 3S pada Dynamometer

Dari hasil *dynamometer* motor listrik 3S, dengan spesifikasi motor BLDC pada motor listrik 3S saat ini belum sesuai dengan kebutuhan *power requirment* dari hasil simulasi model *vehicle dynamic*.

3.3. Metode Pengambilan Data Karakteristik Motor

Berikut metode memperoleh karakteristik motor dalam tesis ini:

3.3.1. Parameter Motor: Pengukuran Resistansi dan Induktansi Motor

Pengukuran resistansi motor dilakukan dengan 2 (dua) cara, yaitu: menggunakan alat ukur LCR meter seperti yang ditunjukkan Gambar 3.13 dan menggunakan perhitungan arus dan tegangan, berdasarkan persamaan hukum Ohm.

$$R = \frac{V}{I} \quad (3.14)$$

Dimana:

R = nilai resistansi (Ω)

V = tegangan *supply* (Volt)

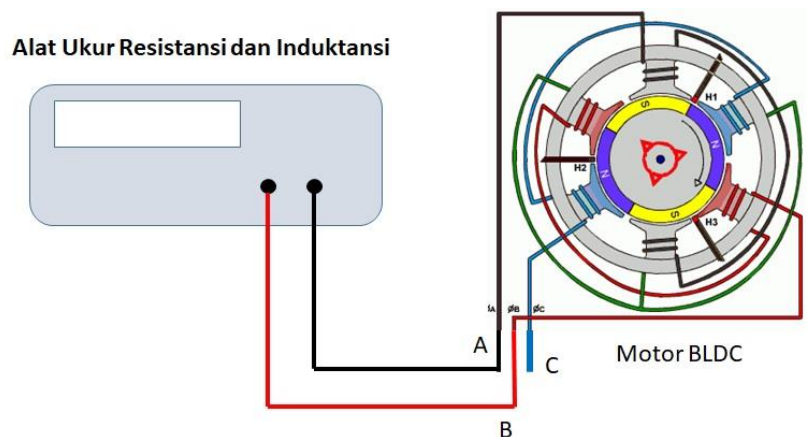
I = arus listrik yang terukur (Ampere).

Pada proses identifikasi ini, dilakukan dengan menggunakan cara yang ke-1, yaitu dengan menggunakan LCR meter. Berdasarkan hasil pengukuran yang telah dilakukan, maka nilai R dan L dari motor BLDC ditunjukkan pada tabel 3.1

Tabel 3.1. Tabel pengukuran R dan L motor BLDC

$R (\Omega)$		$L (\mu H)$	
R_{ab}	0,08	L_{ab}	225,7
R_{bc}	0,09	L_{bc}	184
R_{ca}	0,09	L_{ca}	221,9
Rata-rata	0,0867	Rata-rata	210,533

Karena yang terhubung pada alat ukur adalah dua buah belitan, maka dengan mengasumsikan bahwa nilai resistansi dan induktansi pada masing-masing belitan adalah sama, maka nilai resistansi dan induktansi untuk setiap belitan adalah setengah dari nilai yang terukur. Sehingga didapatkan $R = 0,4335 \Omega$ dan $L = 105,2665 \mu H$.



Gambar 3.13. Skema pengukuran resistansi dan induktansi

3.3.2. Parameter Motor: Konstanta BEMF

Konstanta BEMF merupakan perbandingan antara kecepatan putar mekanik rotor terhadap tegangan BEMF yang dihasilkan kumparan motor. Konstanta BEMF dapat diperoleh dengan persamaan berikut:

$$k_e = \frac{e}{\omega_m} \quad (3.15)$$

Dimana:

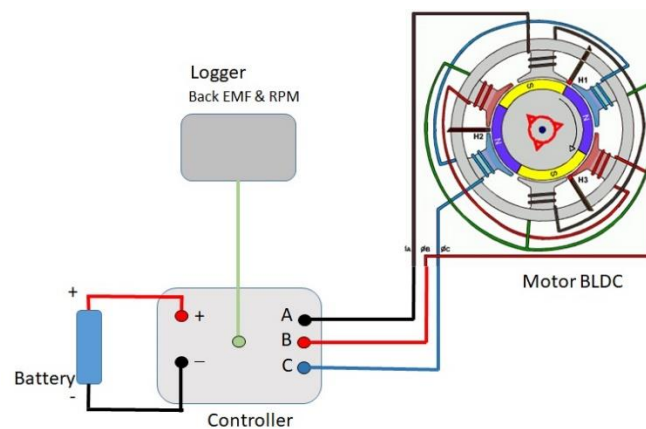
k_e = konstanta BEMF,

e = tegangan BEMF yang dihasilkan kumparan motor (Volt)

ω_m = kecepatan putar mekanik rotor (rad/s).

Langkah-langkah identifikasi konstanta BEMF adalah sebagai berikut.

1. Hubungkan motor BLDC pada sebuah kontroler motor dengan sumber tegangan 12 volt seperti yang ditunjukkan Gambar 3.14.
2. Putar motor BLDC menggunakan algoritma komutasi six-step.
3. Tunggu saat putaran motor mencapai kondisi steady state (dengan asumsi kondisi steady state tercapai saat $t \leq 3$ detik).
4. Kondisikan inverter tidak mensuplai motor dengan cara memberi logik “0” pada setiap gate dari inverter, sehingga motor bergerak dengan sendiri karena efek inersia sampai berhenti.
5. Ukur kecepatan dan tegangan BEMF.
6. Hitung konstanta BEMF.



Gambar 3.14. Skema pengukuran konstanta BEMF

Berdasarkan hasil pengukuran yang dilakukan, didapatkan nilai kecepatan dan tegangan BEMF pada tabel 3.2. berikut.

Tabel 3.2. Tabel pengukuran untuk konstanta BEMF

BEMF (line to line) (Volt)	Speed (RPM)	k_e (V/krpm)
9.796	507.23	19.313
9.667	503.86	19.186
9.362	482.16	19.417
8.704	460.12	18.917
8.624	453.44	19.019
8.222	440.52	18.664
7.981	424.25	18.812
7.917	419.83	18.858
7.499	405.73	18.483
7.483	400.64	18.678
Rata-rata		18.935

Jadi dapat diketahui bahwa nilai konstanta BEMF dari motor BLDC yang digunakan adalah 18,935 volt/krpm.

3.3.3. Parameter Motor: Konstanta Torsi

Konstanta torsi (k_T) merupakan perbandingan antara torsi terhadap arus pada stator. Konstanta torsi dapat dihitung dengan persamaan berikut:

$$k_T = \frac{k_e \times 60}{2\pi} \quad (3.16)$$

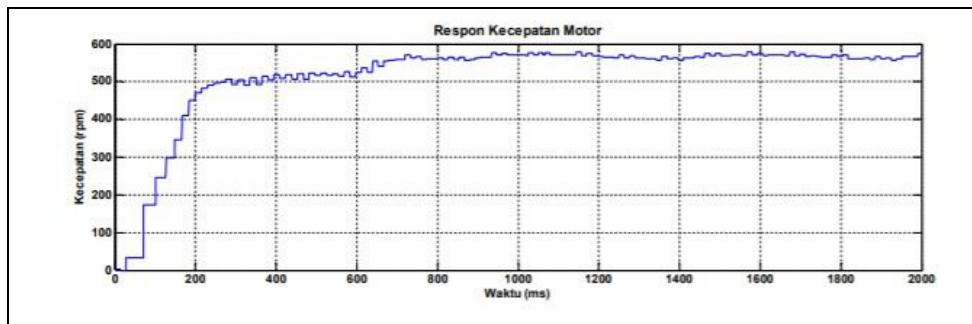
Berdasarkan hasil k_e yang telah didapatkan, maka nilai k_T adalah 0,180815 N.m/A

3.3.4. Parameter Motor: Konstanta Waktu Mekanis

Mechanical time constant (τ_m) merupakan waktu yang dibutuhkan motor untuk mencapai kecepatan 63% dari kecepatan *steady state* saat diberi sinyal *step*. Konstanta waktu mekanis diperoleh dengan melakukan langkah-langkah berikut:

- Putar motor BLDC menggunakan algoritma komutasi *six-step*.
- Logger data kecepatan per waktu (setiap 1ms).
- Plot grafik kecepatan terhadap waktu.
- Waktu saat kecepatan mencapai 63% dari kecepatan *steady state* merupakan konstanta waktu mekanis.

Dari hasil percobaan yang dilakukan, didapatkan plot grafik respon kecepatan yang ditunjukkan oleh gambar 3.10



Gambar 3.15 Respon kecepatan dengan komutasi six-step

Berdasarkan grafik kecepatan yang didapatkan, dapat diketahui nilai konstanta waktu mekanis (τ_m) adalah sekitar 150 mili detik atau 0,15 detik.

3.3.5. Parameter Motor: Koefisien Gaya Gesek

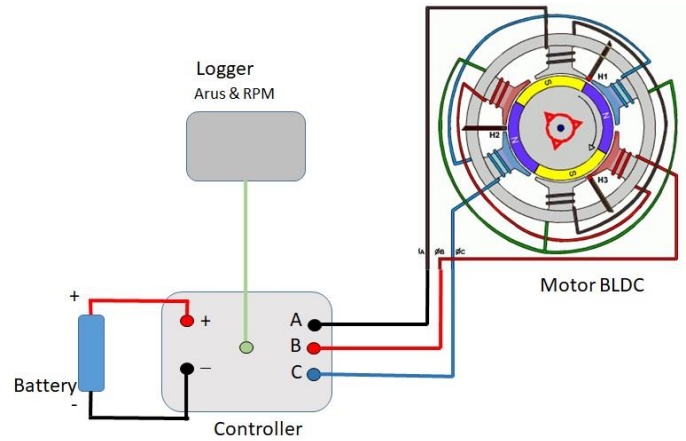
Koefisien gaya gesek dinotasikan sebagai B (N.m.s). Langkah-langkah untuk memperoleh nilai koefisien gaya gesek adalah sebagai berikut :

- Putar motor BLDC menggunakan algoritma komutasi *six-step*.
- Tunggu saat putaran motor mencapai kondisi *steady state* (dengan asumsi kondisi *steady state* tercapai saat $t \geq 1$ detik).
- Hitung torsi elektromagnetik motor menggunakan persamaan berikut :

$$T_e = k_T \times I \quad (3.17)$$

d. Hitung nilai B (N.m.s) berdasarkan persamaan berikut :

$$T_e = B \times \omega_m \quad (3.18)$$



Gambar 3.16. Skema pengukuran arus dan rpm

Dari percobaan yang telah dilakukan, maka didapatkan hasil yang ditunjukkan pada tabel 3.3

Tabel 3.3. Tabel pengukuran untuk konstanta gaya gesek

Arus (ampere)	k_r (N.m/A)	Speed (rad/s)	k_e (N.m.s)
4.399	0.180815	59.418	0.013388
5.854	0.180815	59.110	0.017908
5.835	0.180815	59.168	0.017832
4.446	0.180815	58.769	0.013679
5.005	0.180815	59.899	0.015108
5.166	0.180815	59.057	0.015817
4.982	0.180815	59.562	0.015124
5.401	0.180815	58.791	0.016611
5.532	0.180815	58.959	0.016966
6.231	0.180815	58.835	0.019149
Rata-rata			0.016158

Jadi didapatkan nilai koefisien gaya gesek dari sistem adalah 0,016158 N.m.s.

3.3.6. Parameter Motor: Momen Inersia

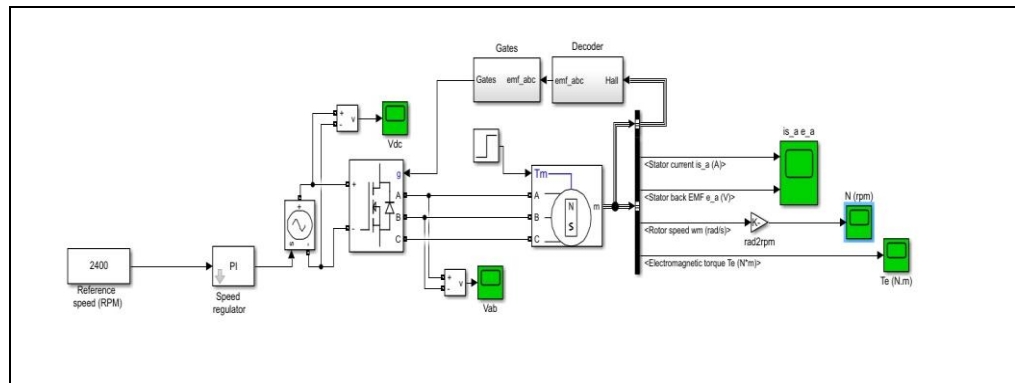
Momen inersia dinotasikan sebagai J (kg.m^2) merupakan sebuah nilai kelembaman motor untuk berotasi pada porosnya. Momen inersia berpengaruh pada saat motor dalam keadaan transien baik itu akselerasi maupun deselerasi. Karena konstanta waktu elektrik jauh lebih kecil dibandingkan dengan konstanta waktu mekanis, maka momen inersia dapat dihitung dengan mengabaikan efek dari induktor dengan persamaan berikut:

$$J = \frac{\tau_m(R_{LL}B + k_e k_t)}{R_{LL}} \quad (3.19)$$

R_{LL} merupakan nilai resistansi *line-to-line* atau dua kali nilai R fasa. Sehingga dapat diketahui nilai momen inersia dari sistem adalah 0,059009 kg.m^2 . Berdasarkan hasil pengukuran parameter motor diatas maka dapat digunakan untuk mengisi parameter motor pada *Matlab/Simulink*.

3.3.7. Pemodelan BLDC Motor

Setelah seluruh parameter motor BLDC didapat selanjutnya dilakukan uji pemodelan motor BLDC seperti yang ditunjukkan Gambar 3.17 menggunakan *Matlab/Simulink*, dimana seluruh parameter motor BLDC diinputkan ke plant motor BLDC seperti yang ditunjukkan Gambar 3.18 pada model untuk memastikan plant motor BLDC sudah stabil, karena untuk mengontrol sebuah plant harus dipastikan *plant* tersebut stabil seperti yang ditunjukkan Gambar 3.19.



Gambar 3.17. Block diagram BLDC by six-Step Inverter open loop

Parameter	Nilai	Satuan
R	0,04335	Ω
L	105,2665	μH
k_e	18,935	volt/krpm
k_t	0,180815	N.m/A
τ_m	0,15	s
B	0,016158	N.m.s
J	0,059009	Kg.m ²
Jumlah pasang kutub	4	-

Block Parameters: Permanent Magnet Synchronous Machine

Permanent Magnet Synchronous Machine (mask) (link)

Implements a three-phase or a five-phase permanent magnet synchronous machine. The stator windings are connected in star to an internal neutral point.

The three-phase machine can have sinusoidal or trapezoidal back EMF waveform. The rotor can be round or salient pole for the sinusoidal machine, it is round when the machine is trapezoidal. Preset models are available for the sinusoidal back EMF machine.

The five-phase machine has a sinusoidal back EMF waveform and round rotor.

Configuration Parameters Advanced

Stator phase resistance R_s (ohm): 0.04335

Stator phase inductance L_s (H): 1.052665e-6

Machine constant

Specify: Flux linkage established by magnets (V.s)

Flux linkage: 0.175

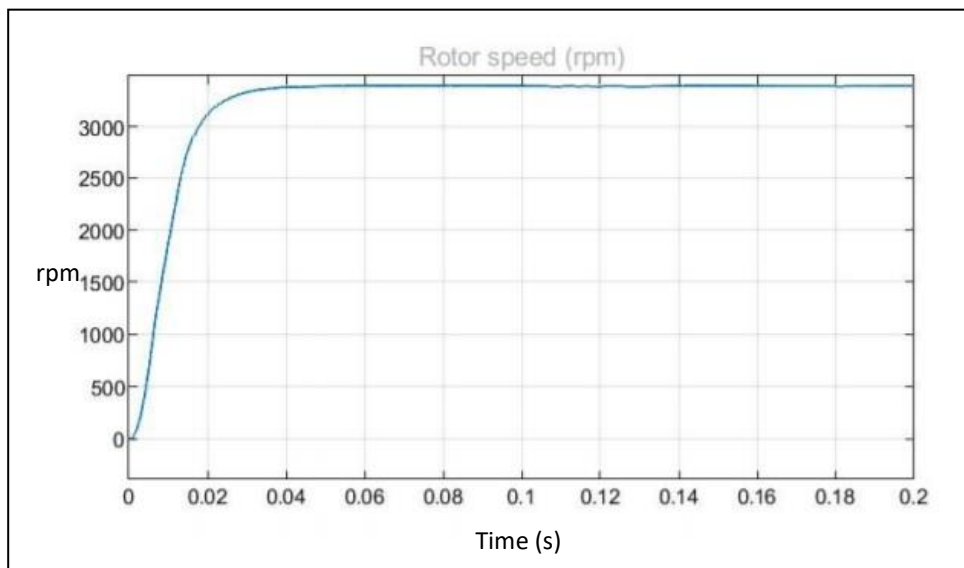
Back EMF flat area (degrees): 120

Inertia, viscous damping, pole pairs, static friction [X(kg.m²), D(N.m.s), pD, T(N.m)]: [0.059009, 0.016158, 4, 0.1772]

Initial conditions [w(rad/s), theta(deg), iA(A)]: [0, 0, 0]

OK Cancel Help Apply

Gambar 3.18. Parameter motor BLDC



Gambar 3.19. Respon pengujian open loop plant motor BLDC

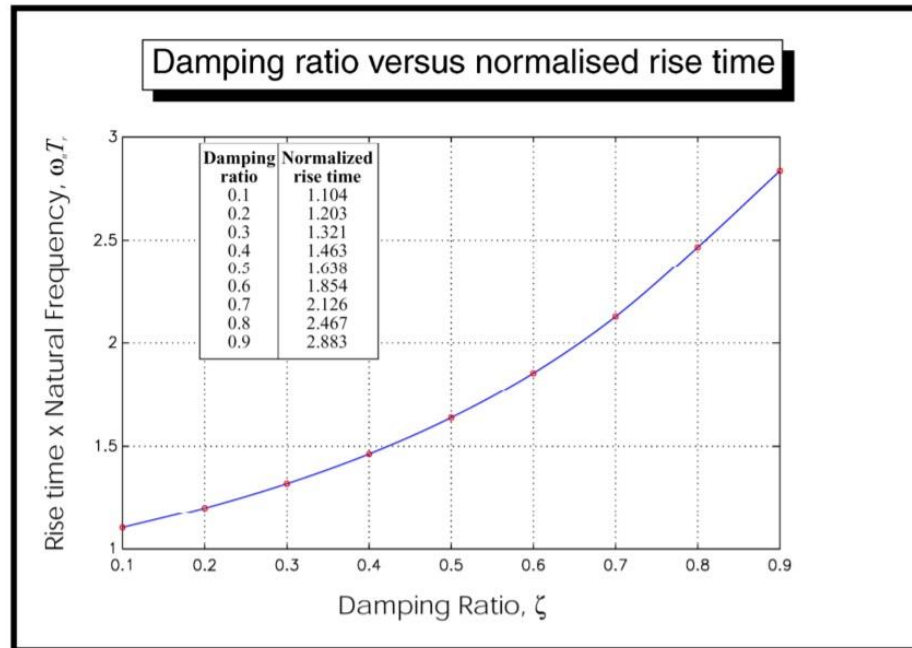
3.4. Spesifikasi Respon *Transient*

Spesifikasi respon *transient* adalah spesifikasi respon sistem yang diamati mulai saat terjadi perubahan sinyal input/gangguan/beban sampai respon masuk dalam keadaan steady state. Tolak ukur yang digunakan untuk mengukur kualitas respon *transient* antara lain *rise time*, *settling time*, dan *percent overshoot*. *Risetime*, *settling time*, dan *peak time* memberikan informasi mengenai kecepatan dan "kualitas" respon transient. Nilai – nilai ini dapat membantu untuk mencapai kecepatan yang diinginkan tanpa osilasi atau *overshoot* yang berlebihan.. Nilai dari *risetime*, *settlingtime*, dan *percent overshoot* didapat dari hasil pendekatan perhitungan sebagai dasar dalam menentukan nilai target dari *risetime*, *settlingtime*, dan *percent overshoot* tersebut, dimana nilai spesifikasi transient dari hasil simulasi harus lebih baik dari nilai target. Nilai *percent overshoot* yang diijinkan dalam system ini maksimal 5%, kemudian dari *percent overshoot* yang telah ditetapkan maka akan didapatkan nilai *rise time* dan *settling time*.

$$\zeta = \frac{-\ln(\frac{\%OS}{100})}{\sqrt{\pi^2 + \ln^2(\frac{\%OS}{100})}} \quad (3.20)$$

$$\zeta = \frac{-\ln(\frac{5}{100})}{\sqrt{3,14^2 + \ln^2(\frac{5}{100})}} \quad (3.21)$$

$$\zeta = 0,7 \quad (3.22)$$



Gambar 3.20. Graphik *damping ratio* vs *rise time* (Norman S. Nise)

Dari Gambar 3.20 dengan nilai *damping ratio* diketahui nilai dari *rise time* sebesar 2,126 detik. Untuk $0.866 < \zeta < 0.5$ maka,

$$Tr = \frac{1,8}{w_n} \quad (3.23)$$

$$2,126 = \frac{1,8}{w_n} \quad (3.24)$$

$$w_n = 0,85 \quad (3.25)$$

Dengan mengetahui w_n maka nilai *settling time* akan didapatkan

$$Ts = \frac{4}{\zeta w_n} \quad (3.26)$$

$$Ts = \frac{4}{0,7 \cdot 0,85} \quad (3.27)$$

$$Ts = 6,72 \quad (3.28)$$

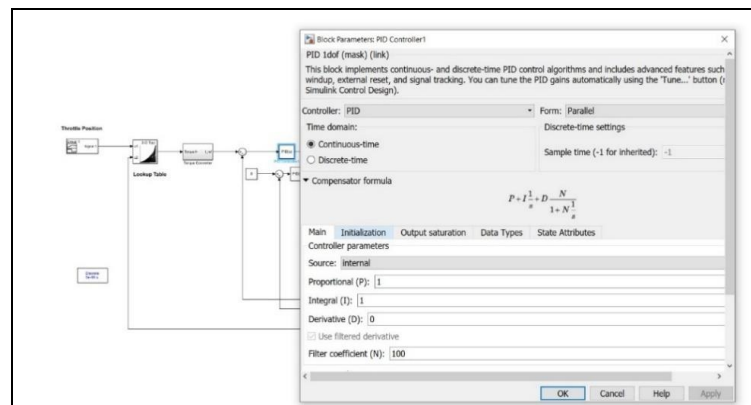
Pada penelitian ini nilai *risetime*, *settling time*, dan *percent overshoot* di tentukan sebagai spesifikasi dari respon transient yang harus dicapai, dimana target dari nilai *risetime* 2,126 detik, *settlingtime* 6,72 detik, dan *percent overshoot* maksimal 5%

Row	Column	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(1)	10	9.16	9.15	9.14	9.13	9.13	9.13	9.12	9.12	9.11	9.11	9.1
(2)	20	9.53	9.49	9.45	9.43	9.4	9.39	9.37	9.34	9.33	9.3	9.29
(3)	30	10.15	10.05	9.97	9.93	9.86	9.82	9.79	9.72	9.69	9.62	9.59
(4)	40	11.03	10.83	10.69	10.62	10.49	10.43	10.37	10.25	10.19	10.08	10.02
(5)	50	12.04	11.84	11.62	11.51	11.31	11.21	11.11	10.93	10.84	10.66	10.58
(6)	60	13.38	13.08	12.76	12.6	12.31	12.17	12.03	11.76	11.63	11.37	11.25
(7)	70	14.93	14.53	14.1	13.89	13.49	13.29	13.1	12.73	12.56	12.21	12.05
(8)	80	16.62	16.22	15.65	15.38	14.85	14.6	14.35	13.87	12.63	13.18	12.97
(9)	90	17.07	16.4	16.07	15.76	15.15	14.85	14.29	14.01	13.5	13.0	12.5
(10)	100	23.54	22	21	20	19	18	17	16	15	14	13

Gambar 3.22. Look up Table pada Simulink

3.6. Tuning Parameter PI

Pada proses ini dilakukan *tuning* parameter PI pada pemodelan *lookup table field oriented control* seperti yang ditunjukkan Gambar 3.23 untuk mendapatkan performa respon transient yang sesuai spesifikasi yang telah ditentukan. Kontrol P ini cukup mampu untuk memperbaiki respon transient khususnya *rise time* dan *setting time* sedangkan pengontrol Integral berfungsi menghasilkan respon sistem yang memiliki *Error Steady State* = 0. Jika sebuah pengontrol tidak memiliki unsur integrator, pengontrol proporsional tidak mampu memastikan nilai *output* sistem dengan *Error Steady State* = 0



Gambar 3.23. Tuning parameter PI

(halaman ini sengaja dikosongkan)

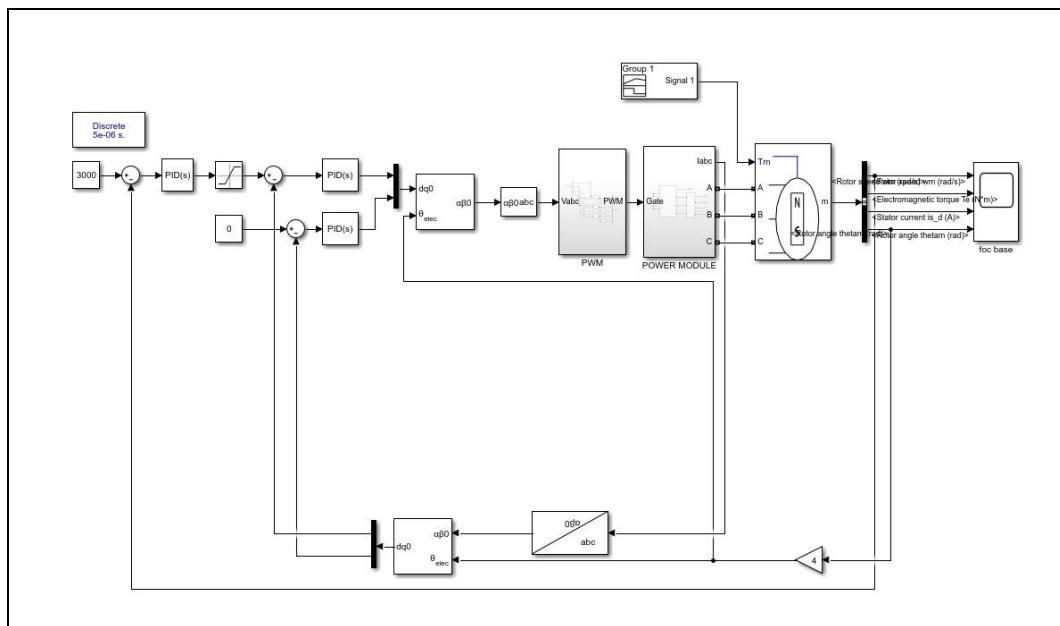
BAB 4

PEMODELAN *LOOKUP TABLE FIELD ORIENTED CONTROL* DAN ANALISA *TRANSIENT*

4.1. Pemodelan dan Simulasi *Lookup Table Field Oriented Control*

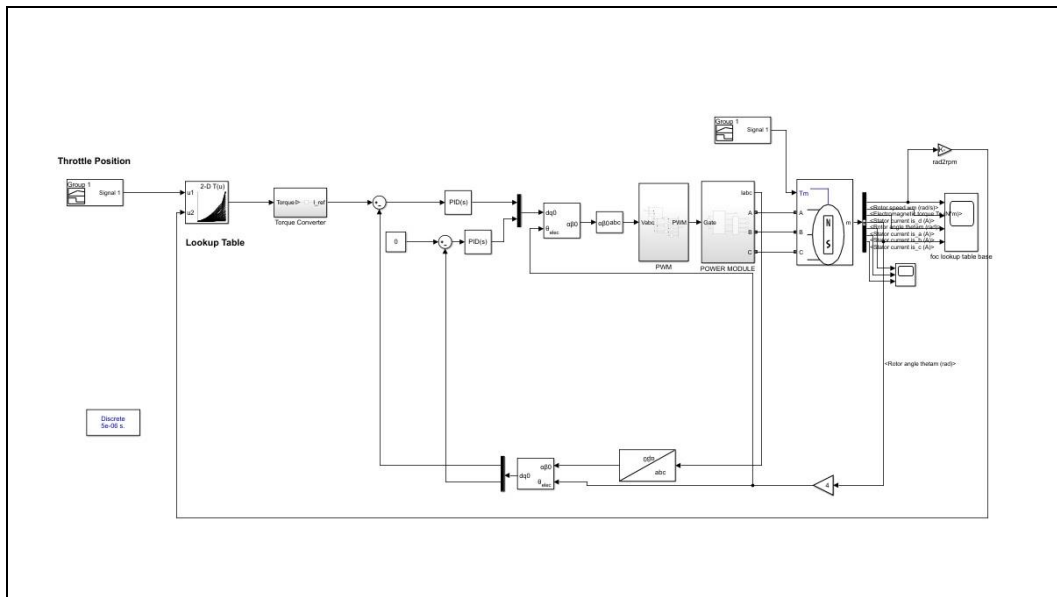
4.1.1. Perancangan Model *Lookup Table Field Oriented Control*

Dasar pemodelan yang digunakan pada tesis ini adalah model *field oriented control* seperti yang ditunjukkan pada Gambar 4.1 yang kemudian dimodifikasi dengan menambahkan *lookup table* sehingga diperoleh perancangan model *lookup table field oriented control* seperti yang ditunjukkan pada Gambar 4.2.



Gambar 4.1. Model *Field Oriented Control* (www.mathworks.com)

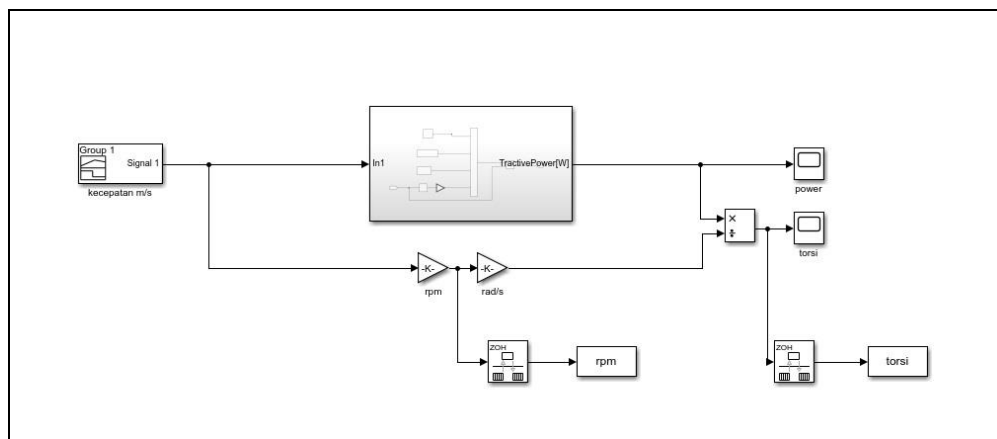
Dalam perancangan *lookup table field oriented control*, *speed control* yang terdapat pada model *field oriented control* dimodifikasi menjadi *torque control* berupa tabel torsi. Tabel torsi tersebut menunjukkan nilai torsi yang dibutuhkan oleh kendaraan pada kondisi tertentu berdasarkan pembacaan *throttle position sensor* dan rpm motor seperti ditunjukkan pada Gambar 4.2.



Gambar 4.2. Model *simulink Lookup Table Field Oriented Control*

4.1.2. Perancangan *Lookup Table*

Pada perancangan *lookup table* data torsi didapatkan dari hasil pemodelan *vehicle dynamic* seperti yang ditunjukkan pada Gambar 4.3. Pada penelitian ini, pemodelan *vehicle dynamic* dilakukan pada dua kondisi jalan yaitu jalan datar dan menanjak.



Gambar 4.3. Model *Vehicle Dynamic*

Dalam perancangan *lookup table* dilakukan pengambilan data torsi pada setiap kecepatan dari pemodelan *vehicle dynamic* pada jalan datar, sehingga diperoleh data torsi dari kecepatan rendah hingga kecepatan tinggi seperti yang

ditunjukkan pada Gambar 4.4 (a), sedangkan pengambilan data torsi pada jalan menanjak dilakukan dengan menggunakan sudut kemiringan jalan sebesar 8° , kecepatan 0-10km/jam, dan percepatan sebesar $0,13\text{m/s}^2$ sehingga diperoleh data torsi seperti yang ditunjukkan pada Gambar 4.4 (b).

RPM Motor	Kecepatan(TPS)									
	100	90	80	70	60	50	40	30	20	10
73.81	24.33	18.12	16.22	14.53	13.08	11.84	10.83	10.05	9.49	9.15
147.63	23.79	17.76	15.93	14.32	12.92	11.73	10.76	10.01	9.47	9.15
221.44	23.27	17.41	15.65	14.10	12.76	11.62	10.69	9.97	9.45	9.14
295.25	22.76	17.07	15.38	13.89	12.61	11.52	10.63	9.93	9.44	9.14
369.07	22.26	16.73	15.11	13.69	12.46	11.41	10.56	9.89	9.42	9.14
442.88	21.77	16.40	14.85	13.49	12.31	11.31	10.49	9.86	9.40	9.13
516.69	21.29	16.07	14.60	13.29	12.17	11.21	10.43	9.82	9.39	9.13
590.51	20.82	15.76	14.35	13.10	12.03	11.11	10.37	9.79	9.37	9.12
664.32	20.36	15.45	14.10	12.92	11.89	11.02	10.31	9.75	9.36	9.12
738.13	19.91	15.15	13.87	12.73	11.76	10.93	10.25	9.72	9.34	9.12
811.95	19.47	14.85	13.63	12.56	11.63	10.84	10.19	9.69	9.33	9.11
885.76	19.05	14.57	13.41	12.38	11.50	10.75	10.13	9.65	9.31	9.11
959.57	18.63	14.29	13.18	12.21	11.37	10.66	10.08	9.62	9.30	9.11
1033.39	18.23	14.01	12.97	12.05	11.25	10.58	10.02	9.59	9.29	9.10
1107.20	17.84	13.75	12.76	11.89	11.13	10.49	9.97	9.56	9.27	9.10
1181.01	17.45	13.49	12.56	11.73	11.02	10.41	9.92	9.53	9.26	9.10
1254.83	17.08	13.24	12.36	11.58	10.91	10.34	9.87	9.51	9.25	9.09
1328.64	16.72	13.00	12.17	11.43	10.80	10.26	9.82	9.48	9.24	9.09
1402.45	16.37	12.76	11.98	11.29	10.69	10.19	9.78	9.45	9.22	9.09
1476.27	16.03	12.53	11.80	11.15	10.59	10.12	9.73	9.43	9.21	9.08
1550.08	15.70	12.31	11.62	11.02	10.49	10.05	9.69	9.40	9.20	9.08
1623.89	15.38	12.10	11.45	10.89	10.40	9.98	9.64	9.38	9.19	9.08
1697.71	15.08	11.89	11.29	10.76	10.31	9.92	9.60	9.36	9.18	9.08
1771.52	14.78	11.69	11.13	10.64	10.22	9.86	9.56	9.33	9.17	9.07
1845.33	14.49	11.50	10.98	10.53	10.13	9.80	9.53	9.31	9.16	9.07
1919.15	14.22	11.31	10.83	10.41	10.05	9.74	9.49	9.29	9.15	9.07
1992.96	13.95	11.13	10.69	10.31	9.97	9.69	9.45	9.27	9.14	9.07
2066.77	13.70	10.96	10.56	10.20	9.90	9.63	9.42	9.25	9.14	9.06
2140.59	13.46	10.80	10.43	10.10	9.82	9.58	9.39	9.24	9.13	9.06
2214.40	13.23	10.64	10.31	10.01	9.75	9.54	9.36	9.22	9.12	9.06
2288.21	13.01	10.49	10.19	9.92	9.69	9.49	9.33	9.20	9.11	9.06
2362.03	12.79	10.35	10.08	9.83	9.62	9.45	9.30	9.19	9.11	9.06
2435.84	12.60	10.22	9.97	9.75	9.56	9.40	9.27	9.17	9.10	9.06
2509.65	12.41	10.09	9.87	9.68	9.51	9.36	9.25	9.16	9.09	9.05
2583.47	12.23	9.97	9.78	9.60	9.45	9.33	9.22	9.14	9.09	9.05
2657.28	12.06	9.86	9.69	9.53	9.40	9.29	9.20	9.13	9.08	9.05
2731.09	11.91	9.75	9.60	9.47	9.36	9.26	9.18	9.12	9.08	9.05
2804.91	11.76	9.65	9.53	9.41	9.31	9.23	9.16	9.11	9.07	9.05
2878.72	11.63	9.56	9.45	9.36	9.27	9.20	9.14	9.10	9.07	9.05
2952.53	11.50	9.48	9.39	9.31	9.24	9.18	9.13	9.09	9.06	9.05
3026.35	11.39	9.40	9.33	9.26	9.20	9.15	9.11	9.08	9.06	9.05

(a)

(b)

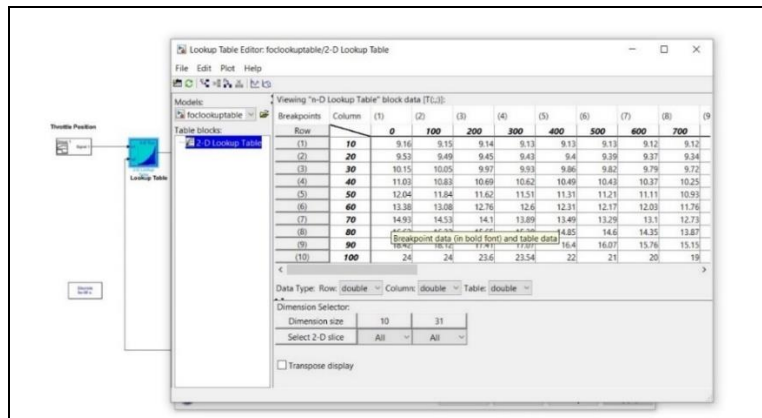
Gambar 4.4. Data torsi pada jalan datar (a) dan data torsi pada jalan menanjak dengan sudut kemiringan 8° (b)

Setelah data torsi dari hasil pemodelan *vehicle dynamic* pada jalan datar dan menanjak didapatkan kemudian dilakukan pemilihan data torsi sesuai *range* rpm setiap kelipatan 100 dari rpm 0-3000. Selanjutnya kedua data torsi yang didapatkan pada jalan datar dan menanjak dikombinasikan sehingga diperoleh data *lookup table* sesuai dengan kebutuhan kendaraan listrik seperti yang ditunjukkan pada Gambar 4.5.

Throttle Position Sensor											
RPM	0	10.00	20.00	30.00	40.00	50.00	60.00	70.00	80.00	90.00	100.00
	0	9.16	9.53	10.15	11.03	12.04	13.38	14.93	16.62	18.42	24
	100.00	9.15	9.49	10.05	10.83	11.84	13.08	14.53	16.22	18.12	23.60
	200.00	9.14	9.45	9.97	10.69	11.62	12.76	14.10	15.65	17.41	23.56
	300.00	9.13	9.43	9.93	10.62	11.51	12.60	13.89	15.38	17.07	23.54
	400.00	9.13	9.40	9.86	10.49	11.31	12.31	13.49	14.85	16.40	23.53
	500.00	9.13	9.39	9.82	10.43	11.21	12.17	13.29	14.60	16.07	17.74
	600.00	9.12	9.37	9.79	10.37	11.11	12.03	13.10	14.35	15.76	17.35
	700.00	9.12	9.34	9.72	10.25	10.93	11.76	12.73	13.87	15.15	16.59
	800.00	9.11	9.33	9.69	10.19	10.84	11.63	12.56	13.63	14.85	16.23
	900.00	9.11	9.30	9.62	10.08	10.66	11.37	12.21	13.18	14.29	15.53
	1000.00	9.10	9.29	9.59	10.02	10.58	11.25	12.05	12.97	14.01	15.19
	1100.00	9.10	9.27	9.56	9.97	10.49	11.13	11.89	12.76	13.75	14.86
	1200.00	9.09	9.25	9.51	9.87	10.34	10.91	11.58	12.36	13.24	14.23
	1300.00	9.09	9.24	9.48	9.82	10.26	10.80	11.43	12.17	13.00	13.93
	1400.00	9.09	9.22	9.45	9.78	10.19	10.69	11.29	11.98	12.76	13.64
	1500.00	9.08	9.20	9.40	9.69	10.05	10.49	11.02	11.62	12.31	13.08
	1600.00	9.08	9.19	9.38	9.64	9.98	10.40	10.89	11.45	12.10	12.82
	1700.00	9.07	9.17	9.33	9.56	9.86	10.22	10.64	11.13	11.69	12.32
	1800.00	9.07	9.16	9.31	9.53	9.80	10.13	10.53	10.98	11.50	12.08
	1900.00	9.07	9.15	9.29	9.49	9.74	10.05	10.41	10.83	11.31	11.85
	2000.00	9.06	9.14	9.25	9.42	9.63	9.90	10.20	10.56	10.96	11.42
	2100.00	9.06	9.13	9.24	9.39	9.58	9.82	10.10	10.43	10.80	11.22
	2200.00	9.06	9.12	9.22	9.36	9.54	9.75	10.01	10.31	10.64	11.02
	2300.00	9.06	9.11	9.19	9.30	9.45	9.62	9.83	10.08	10.35	10.66
	2400.00	9.06	9.10	9.17	9.27	9.40	9.56	9.75	9.97	10.22	10.50
	2500.00	9.05	9.09	9.16	9.25	9.36	9.51	9.68	9.87	10.09	10.34
	2600.00	9.05	9.08	9.13	9.20	9.29	9.40	9.53	9.69	9.86	10.05
	2700.00	9.05	9.08	9.12	9.18	9.26	9.36	9.47	9.60	9.75	9.92
	2800.00	9.05	9.07	9.11	9.16	9.23	9.31	9.41	9.53	9.65	9.80
	2900.00	9.05	9.06	9.09	9.13	9.18	9.24	9.31	9.39	9.48	9.58
	3000.00	9.05	9.06	9.08	9.11	9.15	9.20	9.26	9.33	9.40	9.49

Gambar 4.5. Data *Lookup Table*

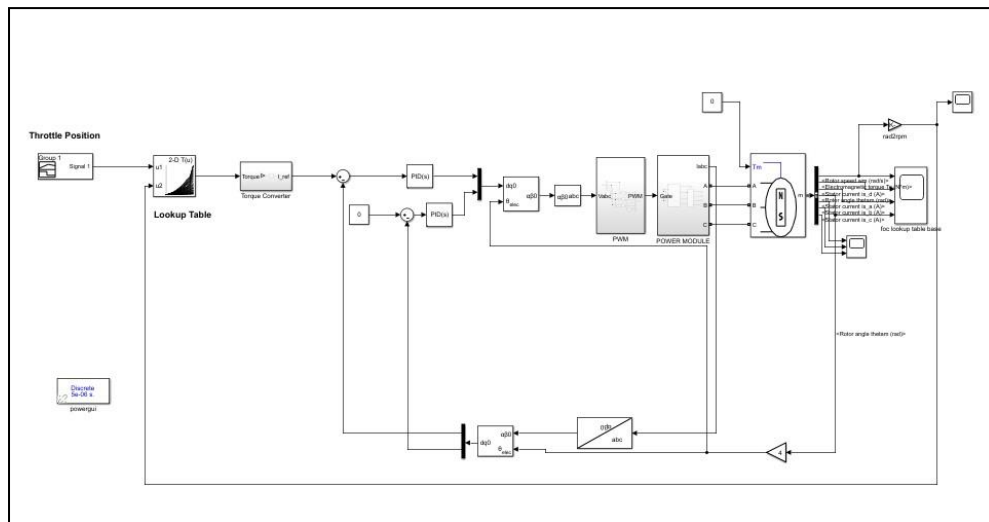
Setelah data *lookup table* didapatkan, data tersebut diolah dengan memasukkan ke dalam *block 2D-lookup table* pada model *lookup table field oriented control* seperti yang ditunjukkan Gambar 4.6. Selanjutnya *block 2D-lookup table* tersebut dapat digunakan dalam simulasi model *lookup table field oriented control*.



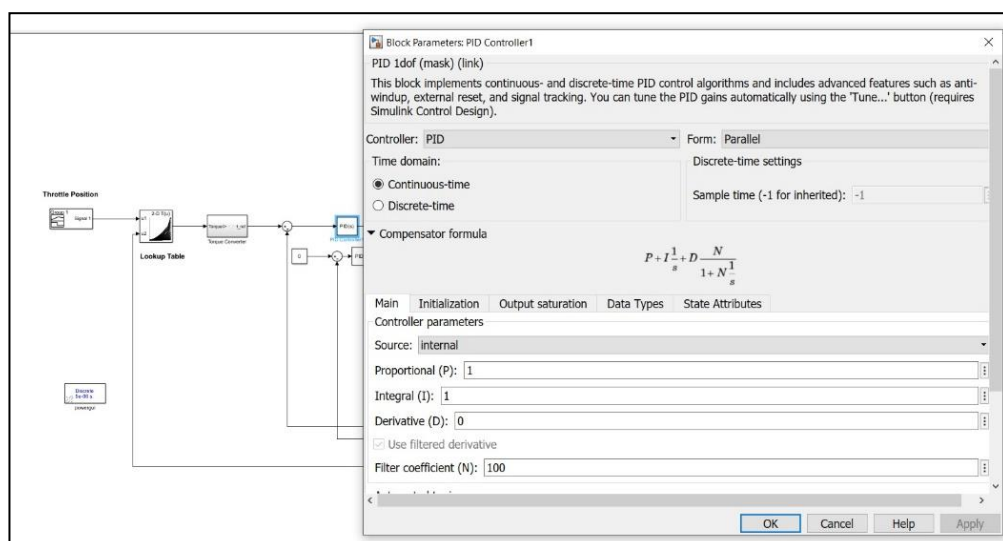
Gambar 4.6. Proses memasukan data *Lookup Table* pada *Block 2D-Lookup Table Simulink*

4.1.3. Simulasi *Lookup Table Field Oriented Control*

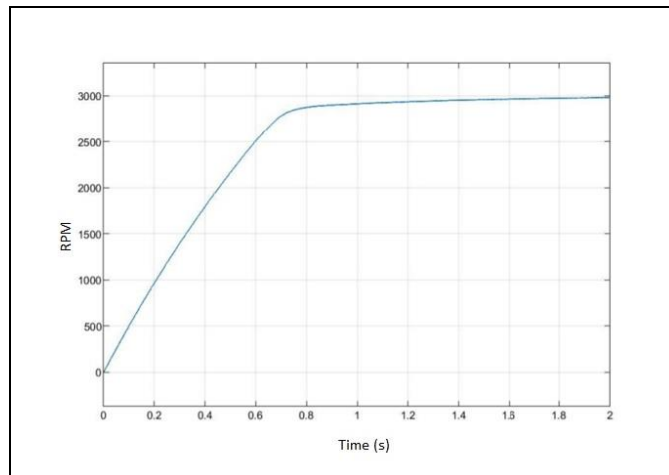
Pada tesis ini dilakukan simulasi terhadap model *simulink* dari *lookup table field oriented control* untuk mengetahui respon model tersebut. Model *field oriented control* yang telah dimodifikasi dengan menambahkan *lookup table* didalamnya seperti yang ditunjukkan Gambar 4.7 disimulasi untuk mengetahui hasil respon terhadap model tersebut. Pemodelan ini menggunakan nilai parameter PI *default* seperti yang ditunjukkan Gambar 4.8 untuk mengetahui respon awal dari model ini yang selanjutnya akan dilakukan *tuning* parameter PI seperti yang ditunjukkan Gambar 4.10 untuk mendapatkan respon terbaik.



Gambar 4.7. Simulasi model *simulink* *Lookup Table Field Oriented Control*

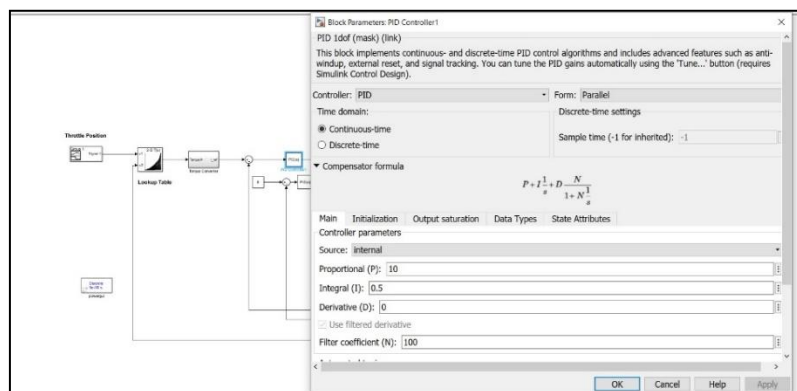


Gambar 4.8. Parameter PI *default* pada *Lookup Table Field Oriented Control*

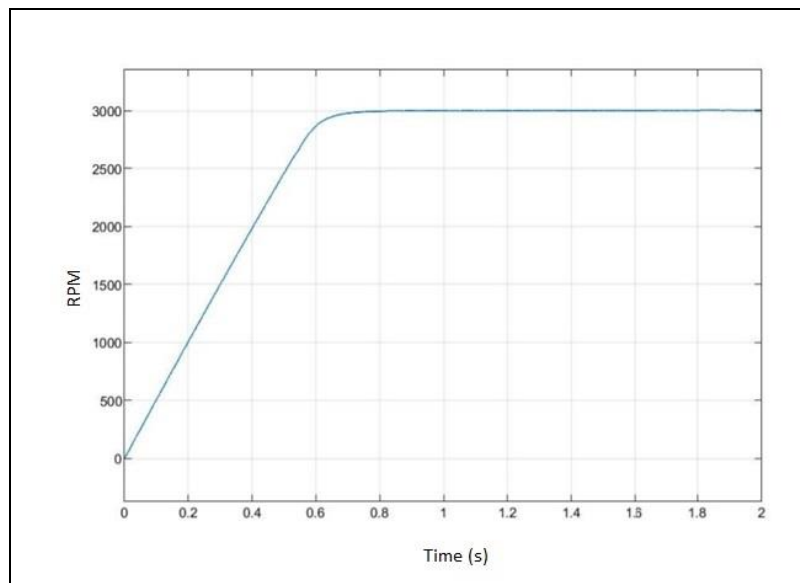


Gambar 4.9. Respon rpm terhadap waktu dari pemodelan *simulink Lookup Table Field Oriented Control*

Gambar 4.9 menunjukkan hasil simulasi model *simulink lookup table field oriented control* dengan nilai k_p sebesar 1 dan nilai k_i 1 diperoleh nilai *rise time* sebesar 0.6445 detik, nilai *settling time* sebesar 1,979 detik, dan nilai *percent overshoot* sebesar 0%. Setelah respon rpm terhadap waktu didapatkan dari simulasi model *lookup table field oriented control* kemudian dilakukan *tuning* nilai parameter PI. *Tuning* parameter PI dilakukan dengan merubah nilai k_p dan k_i pada model untuk memperoleh nilai respon yang lebih baik. Nilai k_p yang tepat dapat memperbaiki respon *transient* khususnya nilai *rise time*, sedangkan nilai k_i yang tepat dapat memperbaiki respon *transient* khususnya nilai *settling time*. *Tuning* parameter PI dilakukan dengan cara *trial and error* hingga mendapatkan respon terbaik.



Gambar 4.10. Parameter PI setelah *tuning* pada *Lookup Table Field Oriented Control*



Gambar 4.11. Respon rpm terhadap waktu dari pemodelan *simulink Lookup Table Field Oriented Control* dengan nilai parameter PI setelah *tuning*

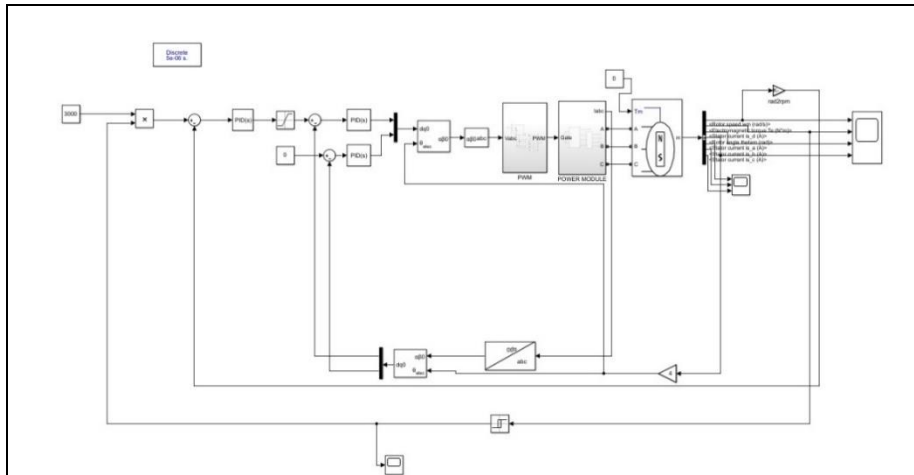
Gambar 4.11 menunjukkan hasil respon rpm terhadap waktu yang diperoleh dari model *simulink look up table field oriented control*. Adapun hasil simulasi yang diperoleh dari model *simulink look up table field oriented control* dengan nilai parameter PI setelah proses *tuning* seperti yang ditunjukkan pada gambar 4.10 dengan nilai $k_p = 10$ dan $k_i = 0,5$ menghasilkan nilai *rise time* sebesar 0.5522 detik, *settling time* sebesar 0.878 detik, dan *percent overshoot* sebesar 0%. Nilai spesifikasi respon *transient* yang dihasilkan dari pemodelan *look up table field oriented control* lebih baik dari nilai *rise time*, *settling time*, dan *percent overshoot* target.

4.2. Simulasi *Field Oriented Control Vedder*, *Field Oriented Control*, dan *Lookup Table Field Oriented Control* Ditinjau dari Respon rpm

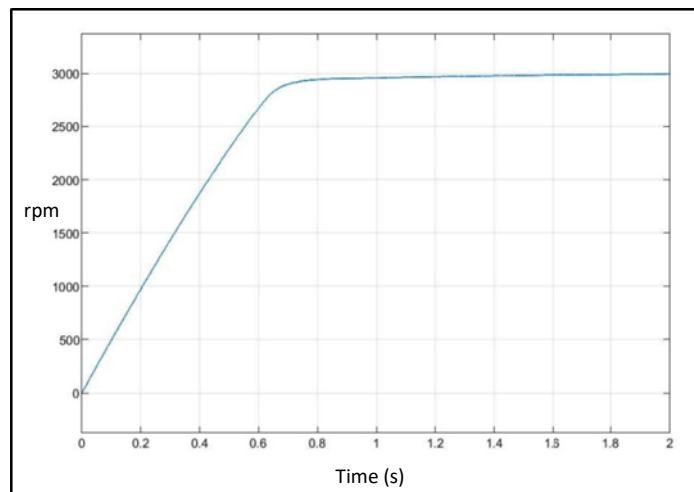
Tesis ini membandingkan dari hasil simulasi beberapa model seperti *field oriented control vedder*, *field oriented control*, dan *lookup table field oriented control* ditinjau dari respon rpm. Simulasi dilakukan dengan tidak memberikan beban pada model untuk memperoleh rpm maksimal.

4.2.1. Simulasi *Field Oriented Control Vedder* Ditinjau dari Respon rpm

Pada *controller* saat ini menggunakan *field oriented control vedder* seperti yang ditunjukkan Gambar 4.12, di mana terdapat limit torsi atau arus yang digunakan menjadi salah satu fitur proteksi pada *controller*. Sistem proteksi ini menonaktifkan *controller* apabila torsi atau arus mencapai limit, namun selama torsi atau arus tidak mencapai posisi limit *controller* tetap berfungsi normal.



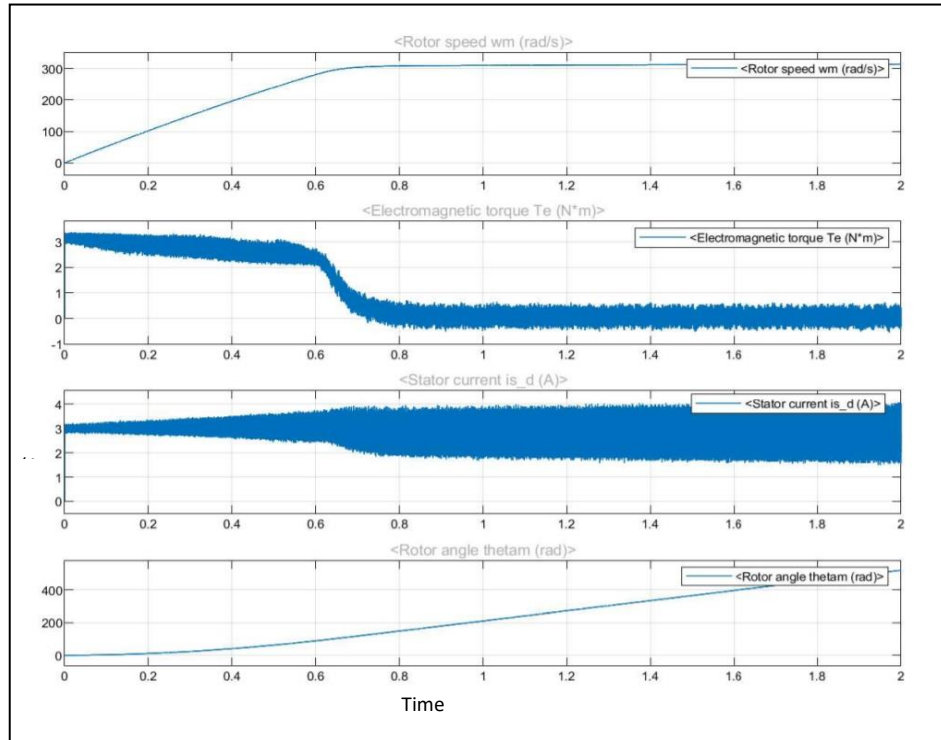
Gambar 4.12. Simulasi model *simulink Field Oriented Control Vedder*



Gambar 4.13. Respon rpm terhadap waktu dari model *simulink Field Oriented Control Vedder*

Gambar 4.13 menunjukkan hasil respon rpm terhadap waktu yang diperoleh dari model *simulink field oriented control vedder*. Hasil simulasi yang

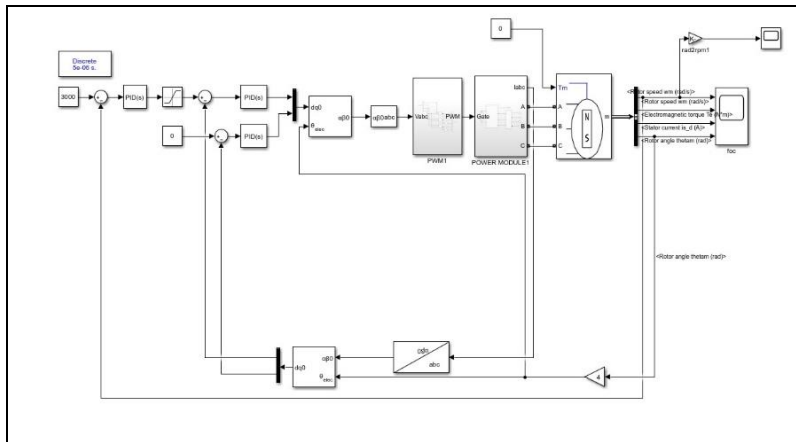
diperoleh adalah nilai *rise time* sebesar 0,65 detik, *settling time* sebesar 1,5 detik, dan *percent overshoot* sebesar 0%.



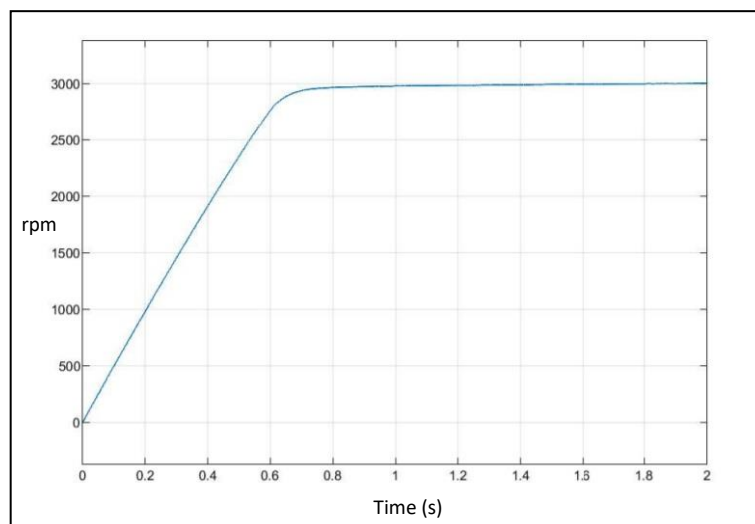
Gambar 4.14. Hasil simulasi model *simulink* sistem *Field Oriented Control Vedder*

Gambar 4.14 menunjukkan nilai torsi dan arus yang dihasilkan dari pemodelan *field oriented control vedder*. Pada pemodelan ini nilai torsi yang dihasilkan sebesar 3 Nm dan nilai arus yang dihasilkan pada stator sebesar 3 A. Simulasi sistem *control vedder* masih berjalan dikarenakan torsi yang dihasilkan belum mencapai limit torsi yang di atur pada torsi maksimal sebesar 24 Nm.

4.2.2. Simulasi Model *Field Oriented Control* Ditinjau dari Respon rpm

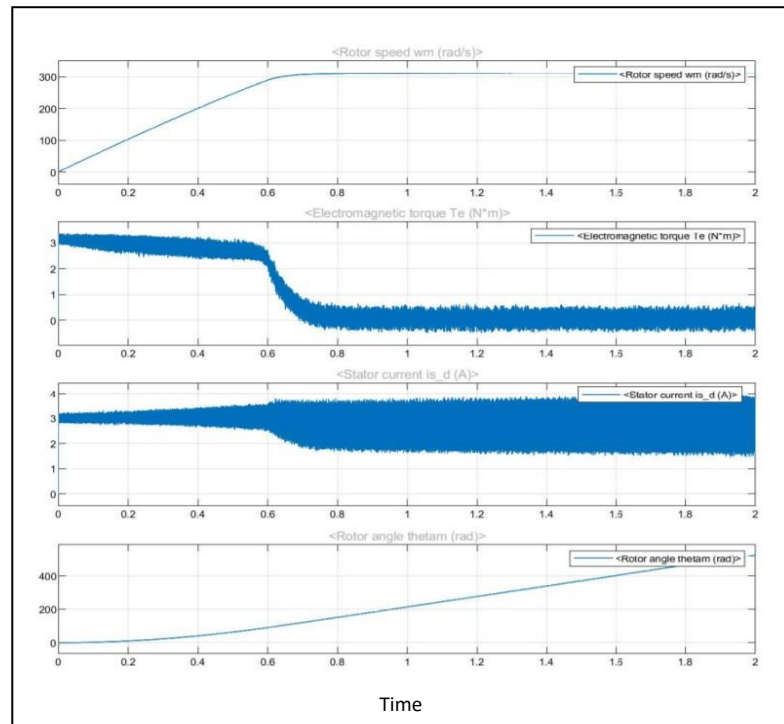


Gambar 4.15. Simulasi model *simulink Field Oriented Control*



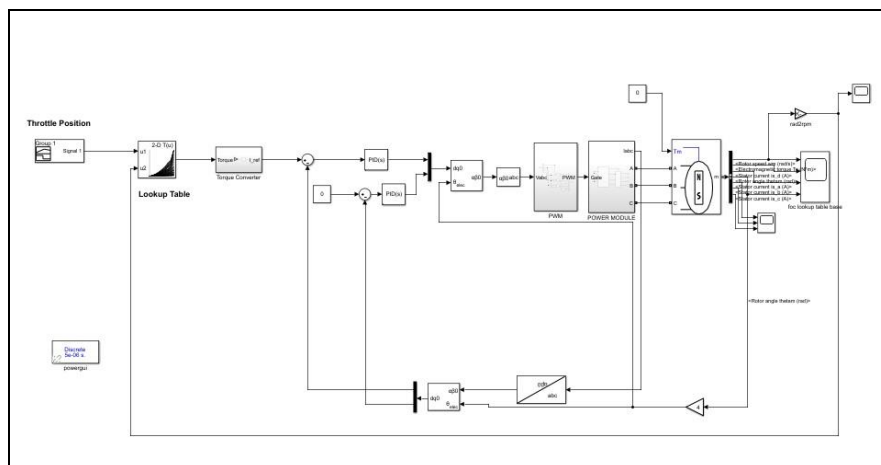
Gambar 4.16. Respon rpm terhadap waktu dari model *simulink Field Oriented Control*

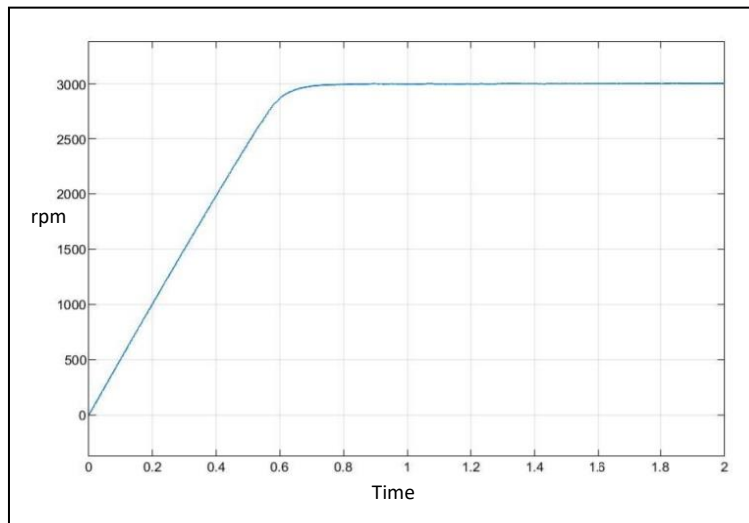
Gambar 4.16 menunjukkan hasil respon rpm terhadap waktu yang diperoleh dari model *simulink field oriented control* seperti yang ditunjukkan Gambar 4.15 adalah nilai *rise time* sebesar 0,6 detik, *settling time* sebesar 1 detik, dan *percent overshoot* sebesar 0%.



Gambar 4.17 menunjukkan nilai torsi dan arus yang dihasilkan dari pemodelan *field oriented control*. Pada pemodelan tersebut nilai torsi yang dihasilkan sebesar 3 Nm dan nilai arus yang dihasilkan pada stator sebesar 3 A.

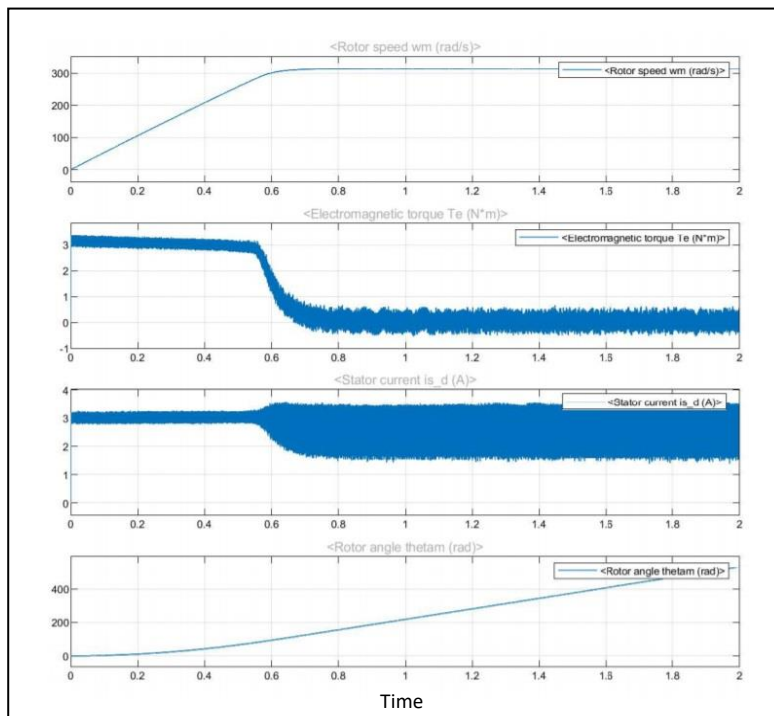
4.2.3. Simulasi Model *Lookup Table Field Oriented Control* Ditinjau dari Respon rpm





Gambar 4.19. Respon rpm terhadap waktu dari model *simulink Lookup Table Field Oriented Control*

Gambar 4.19 menunjukkan hasil respon rpm terhadap waktu yang diperoleh dari model *simulink lookup table field oriented control* seperti yang ditunjukkan Gambar 4.18 adalah nilai *rise time* sebesar 0,552 detik, *settling time* sebesar 0,878 detik, dan *percent overshoot* sebesar 0%.

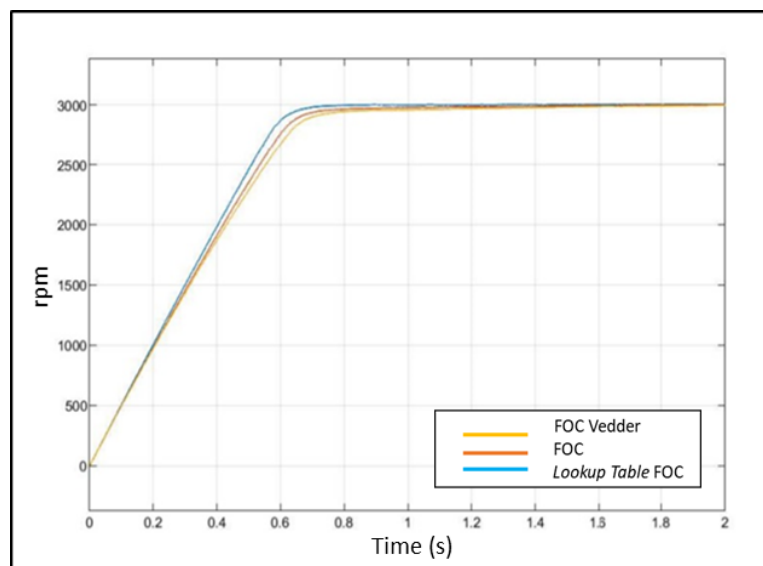


Gambar 4.20. Hasil simulasi model *simulink Lookup Table Field Oriented Control*

Gambar 4.20 menunjukkan nilai torsi dan arus yang dihasilkan dari pemodelan *field oriented control* dimana nilai torsi yang dihasilkan sebesar 3 Nm dan nilai arus yang dihasilkan pada stator sebesar 3 A.

4.2.4. Perbandingan Hasil Simulasi Ditinjau dari Respon rpm

Gambar 4.21 merupakan perbandingan hasil simulasi model *field oriented control vedder* yang ditunjukkan oleh garis berwarna kuning, *field oriented control* yang ditunjukkan oleh garis berwarna merah, dan *lookup table field oriented control* yang ditunjukkan oleh garis yang berwarna biru ditinjau dari respon rpm. Simulasi dari *lookup table field oriented control* memiliki hasil dengan nilai *rise time*, *settling time*, dan *percent overshoot* paling baik dibanding model *field oriented control vedder* dan *field oriented control* seperti yang ditunjukkan pada Tabel 4.1.



Gambar 4.21. Perbandingan respon rpm terhadap waktu

Tabel 4.1. Tabel hasil simulasi ditinjau dari respon rpm.

Sistem kontrol	<i>Rise Time</i> (s)	<i>Settling Time</i> (s)	<i>Percent Overshoot</i> (%)
Target	2,162	6,172	5
Vedder	0,65	1,5	0
FOC	0,6	1	0
<i>Lookup Table</i> FOC	0,5522	0,878	0

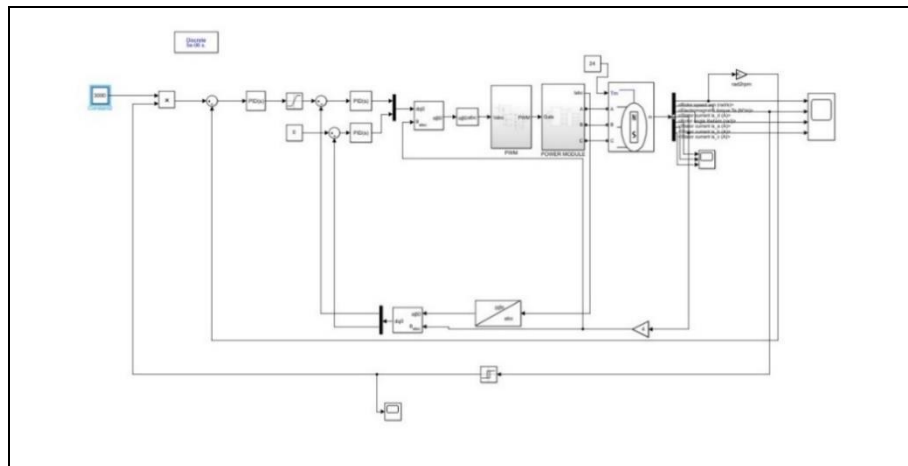
Dari hasil simulasi ditinjau dari respon rpm seperti yang ditunjukkan pada gambar 4.21 dan tabel 4.1, simulasi model *lookup table field oriented control* memiliki nilai *rise time*, *settling time*, dan *percent overshoot* yang lebih baik dibandingkan dengan respon rpm dari model *field oriented control vedder* dan *field oriented control*. Hal ini menunjukkan *lookup table* yang ditambahkan pada model *field oriented control* bekerja dengan baik sebagai pengganti dari *speed control* sehingga menghasilkan respon dengan nilai *rise time*, *settling time*, dan *percent overshoot* yang lebih baik dibandingkan dengan model *field oriented control vedder* dan *field oriented control*.

4.3. Simulasi Model *Field Oriented Control Vedder*, *Field Oriented Control*, dan *Lookup Table Field Oriented Control* Ditinjau dari Respon Torsi

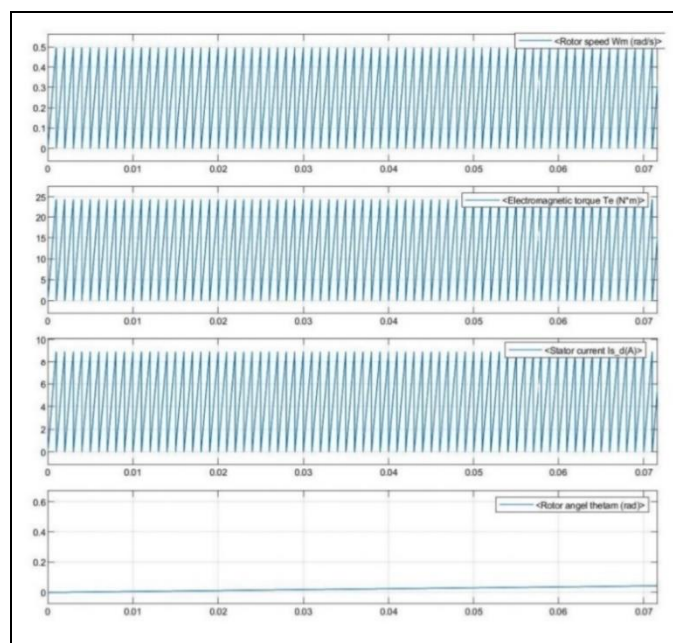
Tesis ini melakukan perbandingan dari hasil simulasi beberapa model seperti *field oriented control vedder*, *field oriented control*, dan *lookup table field oriented control* ditinjau dari respon torsi dengan memberikam beban sebesar 24 Nm pada model.

4.3.1. Simulasi *Field Oriented Control Vedder* Ditinjau dari Respon Torsi.

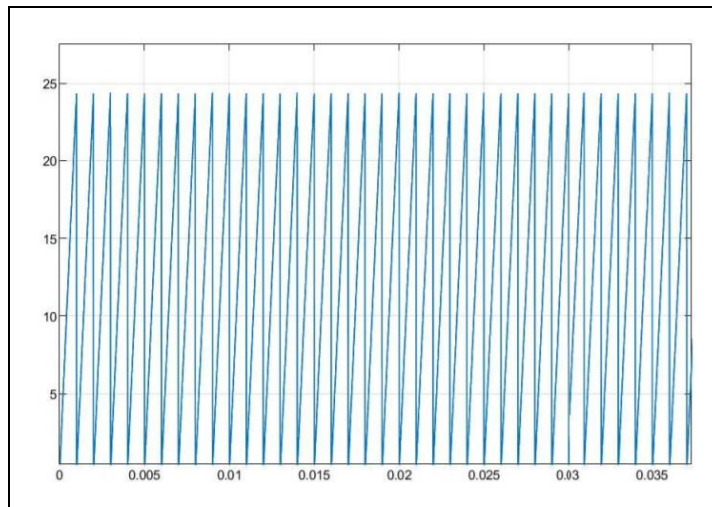
Pada *field oriented control vedder* seperti yang ditunjukkan Gambar 4.22 terdapat limit torsi atau arus yang digunakan menjadi salah satu fitur proteksi pada *controller* ini. Sistem proteksi ini menonaktifkan *controller* jika torsi atau arus mencapai limit seperti yang ditunjukkan Gambar 4.23.



Gambar 4.22 Simulasi model *simulink* Field Oriented Control Vedder



Gambar 4.23. Hasil simulasi model *simulink* Field Oriented Control Vedder

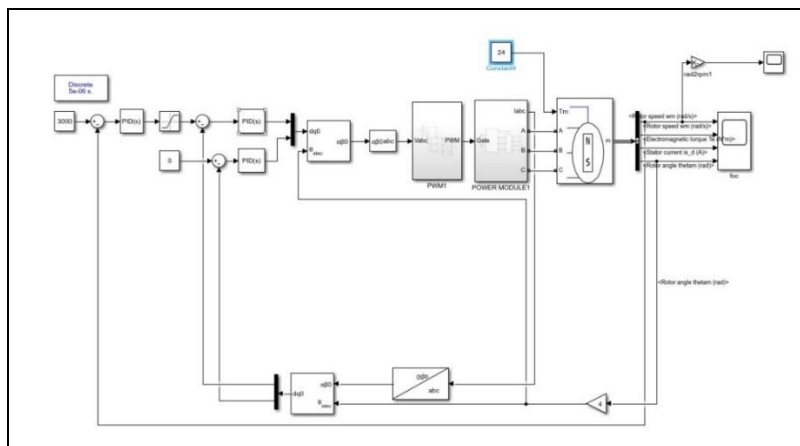


Gambar 4.24. *Zoom Respon Torsi*

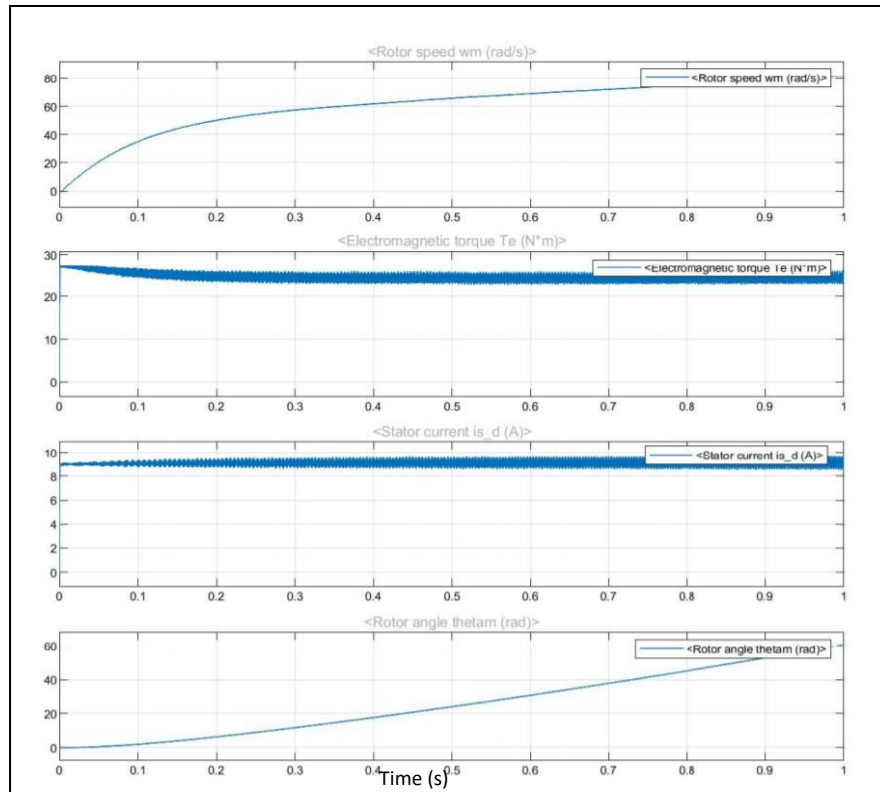
Gambar 4.24 menunjukkan torsi yang dihasilkan pada simulasi model *simulink field oriented control vedder* mencapai limit torsi, sehingga sistem control akan nonaktif dan akan aktif lagi beberapa saat kemudian. Kejadian ini akan terus berulang ketika torsi mencapai limit. Adapun hasil simulasi yang diperoleh yaitu nilai *rise time* sebesar 0,001 detik, nilai *settling time* dan *percent overshoot* tidak tercapai.

4.3.2. Simulasi Model *Field Oriented Control* Ditinjau dari Respon Torsi.

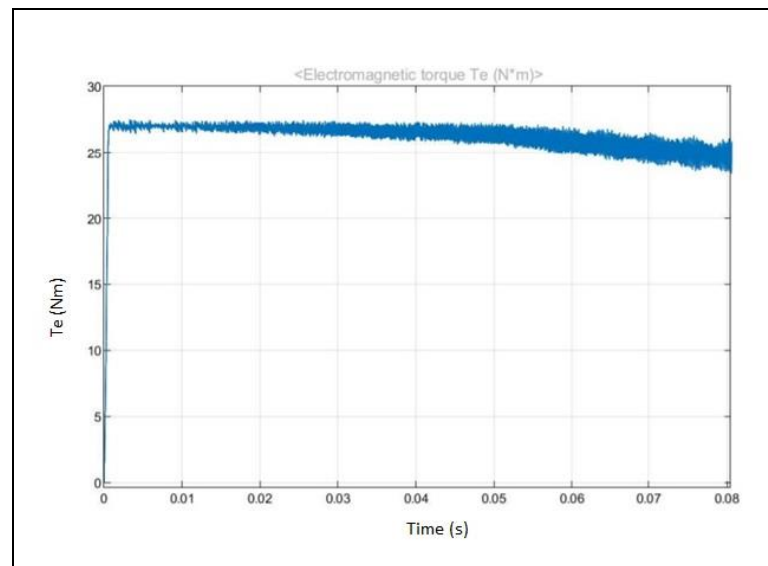
Pada tesis ini melakukan simulasi dari model *field oriented control* seperti yang ditunjukkan Gambar 4.25 dengan pembebanan sebesar 24 Nm. Hasil dari simulasi ini ditinjau dari respon torsi terhadap beban yang diberikan.



Gambar 4.25. Simulasi model *simulink* Field Oriented Control



Gambar 4.26. Hasil simulasi model *simulink Field Oriented Control*



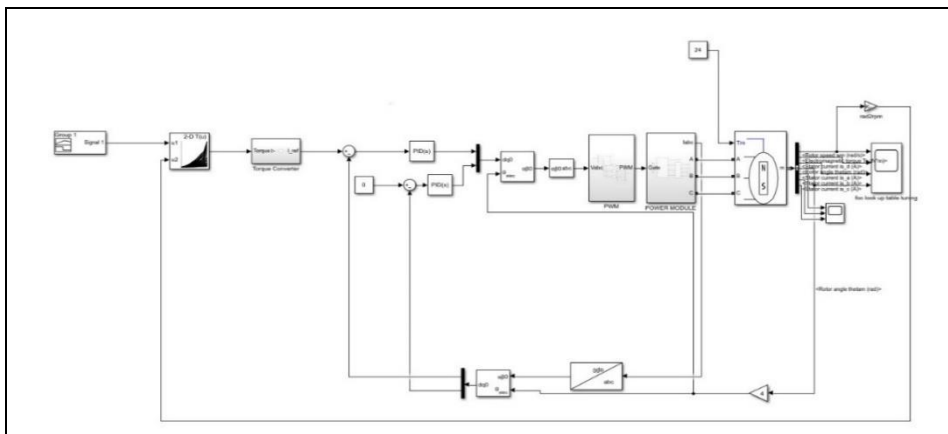
Gambar 4.27. *Zoom Respon Torsi*

Gambar 4.26 menunjukkan hasil respon torsi terhadap waktu yang diperoleh dari model *simulink field oriented control*. Hasil simulasi yang diperoleh adalah nilai *rise time* sebesar 0,001 detik, *settling time* sebesar 0,003

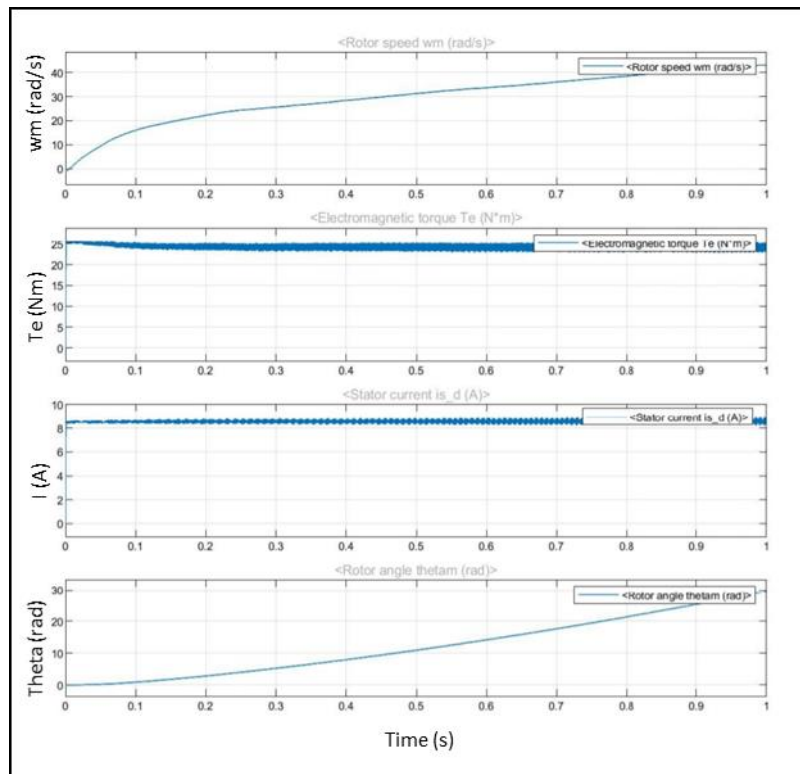
detik, dan *percent overshoot* sebesar 12,5% seperti yang ditunjukkan Gambar 4.27.

4.3.3. Simulasi Model *Lookup Table Field Oriented Control* Ditinjau dari Respon Torsi

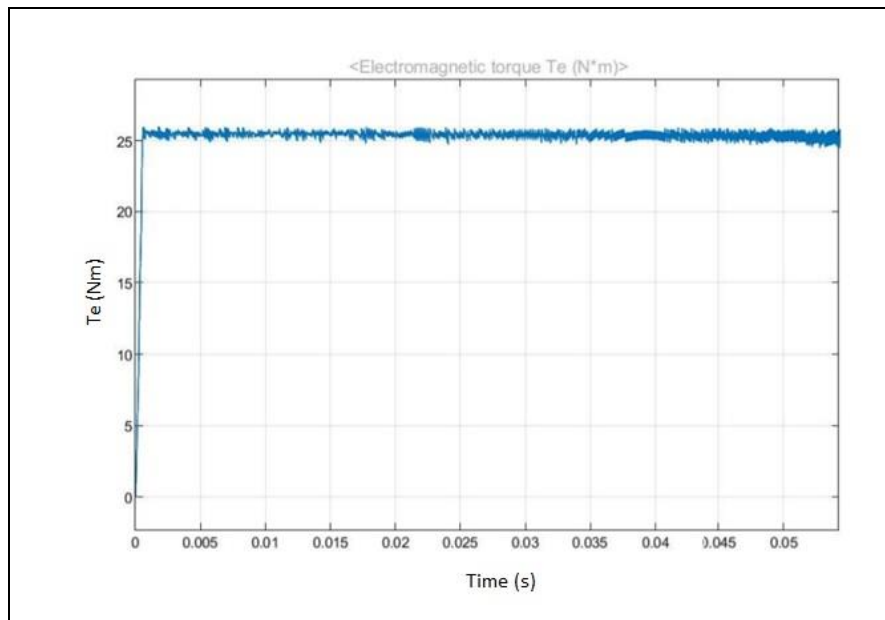
Tesis ini melakukan simulasi dengan model *lookup table field oriented control* seperti yang ditunjukkan Gambar 4.28 dengan pembebanan sebesar 24 Nm. Hasil dari simulasi ini ditinjau dari respon torsi terhadap beban yang diberikan.



Gambar 4.28. Simulasi model *simulink Lookup Table Field Oriented Control*



Gambar 4.29. Hasil simulasi model *simulink lookup table field oriented control*



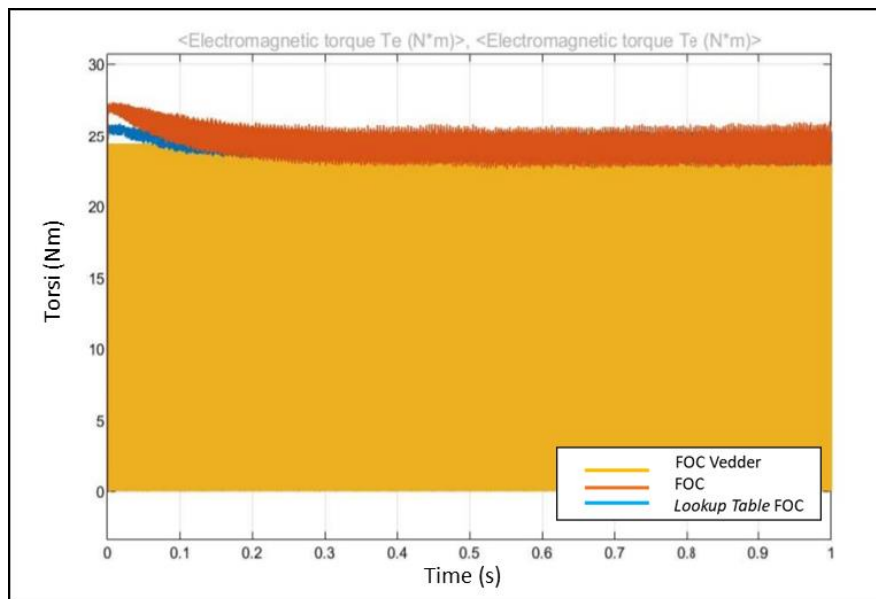
Gambar 4.30. *Zoom respon torsi*

Gambar 4.29 menunjukkan hasil respon torsi terhadap waktu yang diperoleh dari model *simulink field oriented control*. Hasil simulasi yang

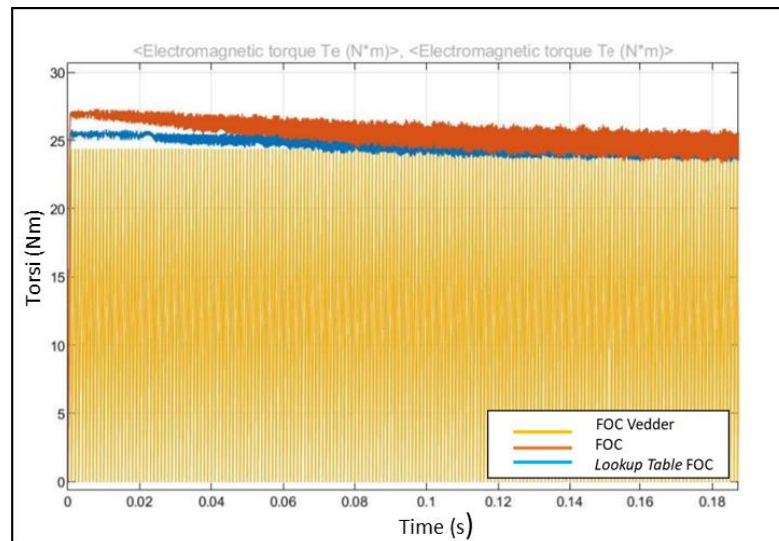
diperoleh adalah nilai *rise time* sebesar 0,001 detik, *settling time* sebesar 0,002 detik, dan *percent overshoot* sebesar 4% seperti yang ditunjukkan Gambar 4.30.

4.3.4. Perbandingan Hasil Simulasi Ditinjau dari Respon Torsi

Gambar 4.31 dan 4.32 merupakan perbandingan hasil simulasi model *field oriented control vedder* ditunjukkan oleh garis berwarna kuning, *field oriented control* ditunjukkan oleh garis berwarna merah, dan *lookup table field oriented control* yang ditunjukkan oleh garis berwarna biru ditinjau dari respon torsi. Hasil simulasi *lookup table field oriented control* yaitu nilai *rise time*, *settling time*, dan *percent overshoot* memiliki nilai yang lebih baik dibanding model *field oriented control vedder* dan *field oriented control* seperti yang ditunjukkan pada Tabel 4.2.



Gambar 4.31. Perbandingan respon torsi terhadap waktu



Gambar 4.32. *Zoom* perbandingan respon torsi terhadap waktu

Tabel 4.2. Tabel perbandingan hasil simulasi ditinjau dari respon torsi.

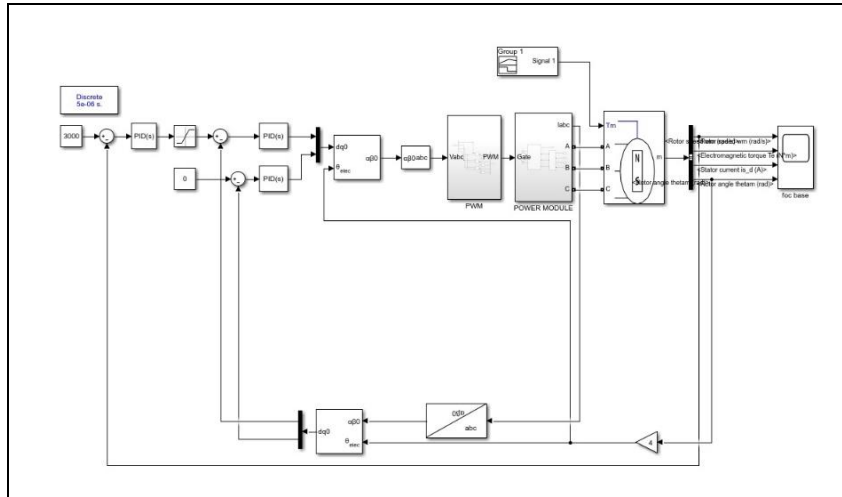
Sistem Kontrol	<i>Rise Time</i> (s)	<i>Settling Time</i> (s)	<i>Percent Overshoot</i> (%)
Target	2,162	6,172	5
Vedder	0,001	∞	∞
FOC	0,001	0,003	12,5
<i>Lookup Table</i> FOC	0,001	0,002	4

Dari hasil simulasi ditinjau dari respon torsi seperti yang ditunjukkan pada Gambar 4.21 dan Tabel 4.1, simulasi model *lookup table field oriented control* memiliki respon torsi dengan nilai *percent overshoot* yang lebih kecil dibandingkan dengan *field oriented control vedder* dan *field oriented control*. Hal ini menunjukkan *lookup table* pada model *lookup table field oriented control* memiliki keakuratan respon torsi dalam mencapai *set point* yang lebih baik dibandingkan model *field oriented control vedder* dan *field oriented control*.

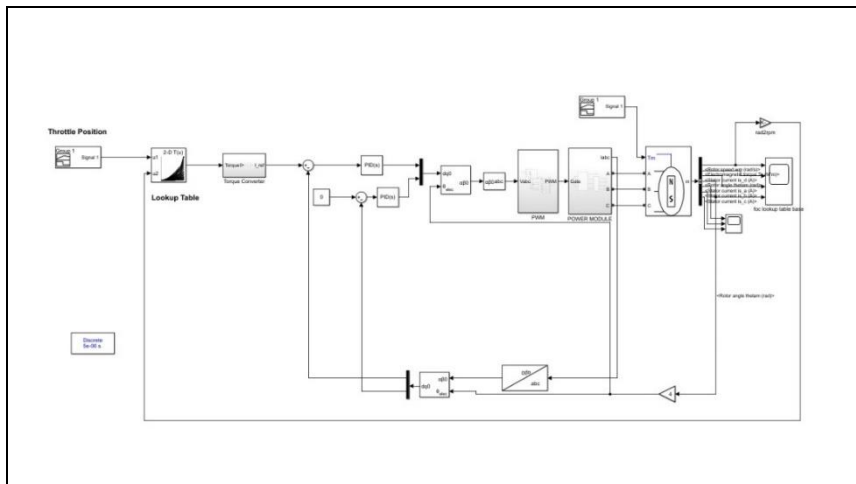
4.4. Simulasi Model *Field Oriented Control* dan *Lookup Table Field Oriented Control* Ditinjau dari Respon Torsi dengan Variasi Beban

Tesis ini melakukan simulasi terhadap model *field oriented control* seperti yang ditunjukkan Gambar 4.33 dan *lookup table field oriented control* seperti yang ditunjukkan Gambar 4.34 dengan memberikan beban yang bervariasi.

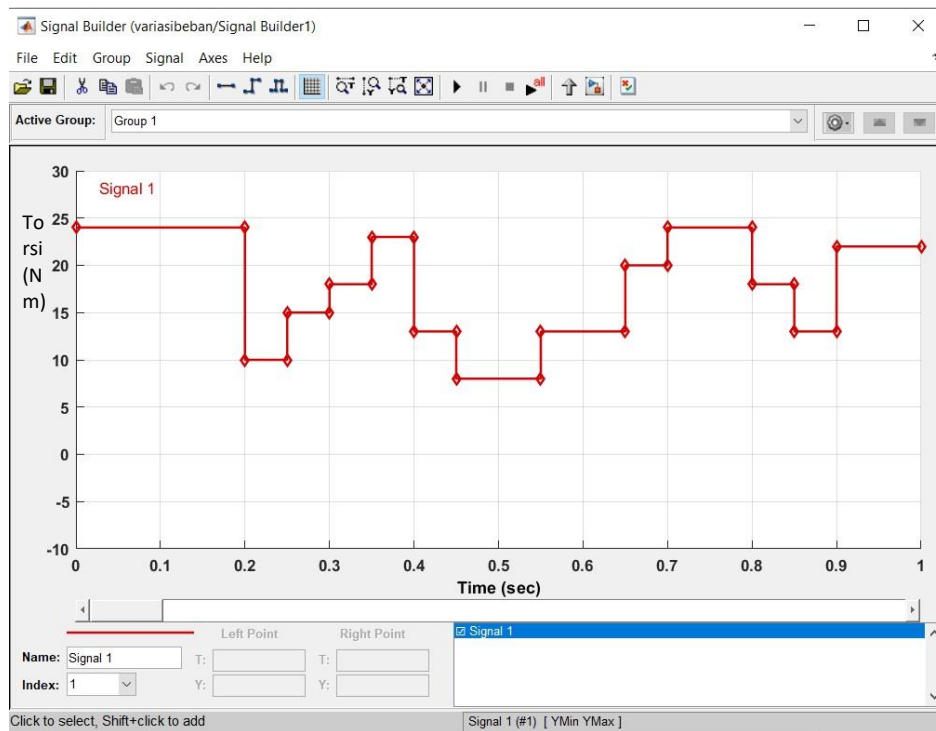
Adapun variasi beban yang diberikan pada simulasi ini sebesar 8 Nm, 10 Nm, 12 Nm, 13 Nm, 15 Nm, 18 Nm, 20 Nm, 22 Nm, 23 Nm dan 24 Nm dengan waktu pembebanan yang berbeda-beda pada masing-masing beban.



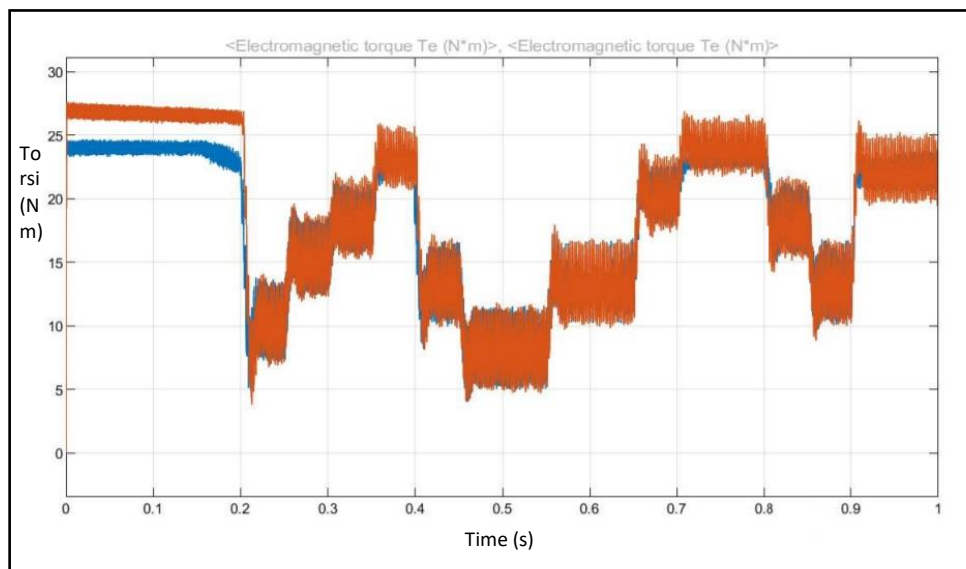
Gambar 4.33. Simulasi model *simulink Field Oriented Control* dengan variasi beban



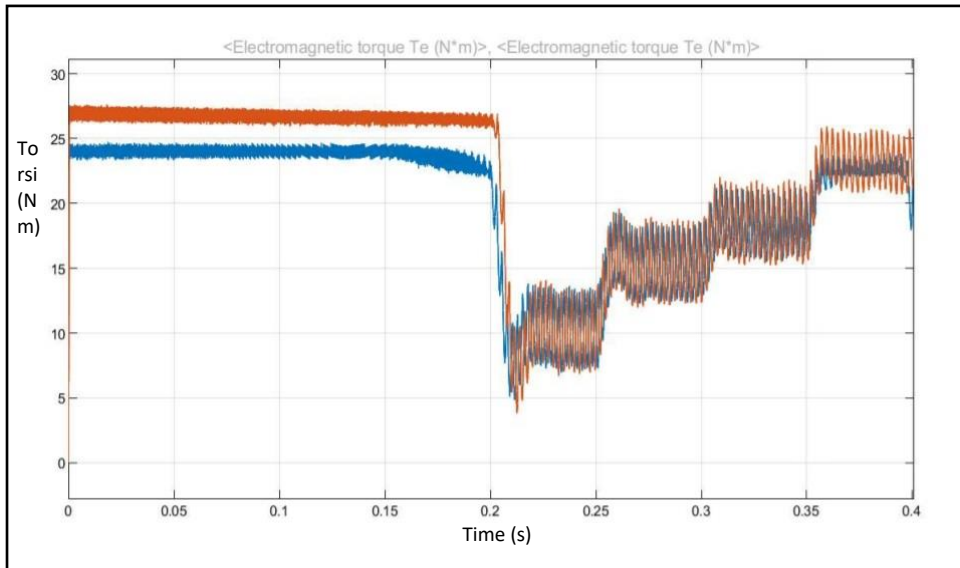
Gambar 4.34. Simulasi model *simulink Lookup Table Field Oriented Control* dengan variasi beban



Gambar 4.35. Variasi beban pada simulasi model *simulink Field Oriented Control* dan *Lookup Table Field Oriented Control*



Gambar 4.36. Hasil simulasi model *simulink Field Oriented Control* dan *Lookup Table Field Oriented Control* dengan variasi beban

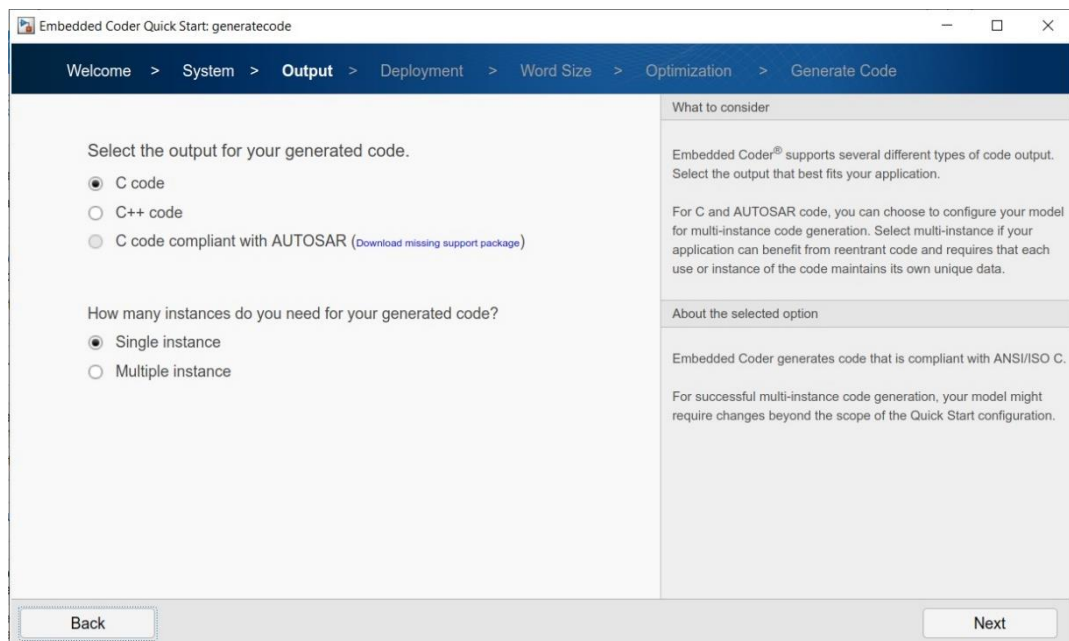


Gambar 4.37. Zoom hasil simulasi model *simulink Field Oriented Control* dan *Lookup Table Field Oriented Control* dengan variasi beban

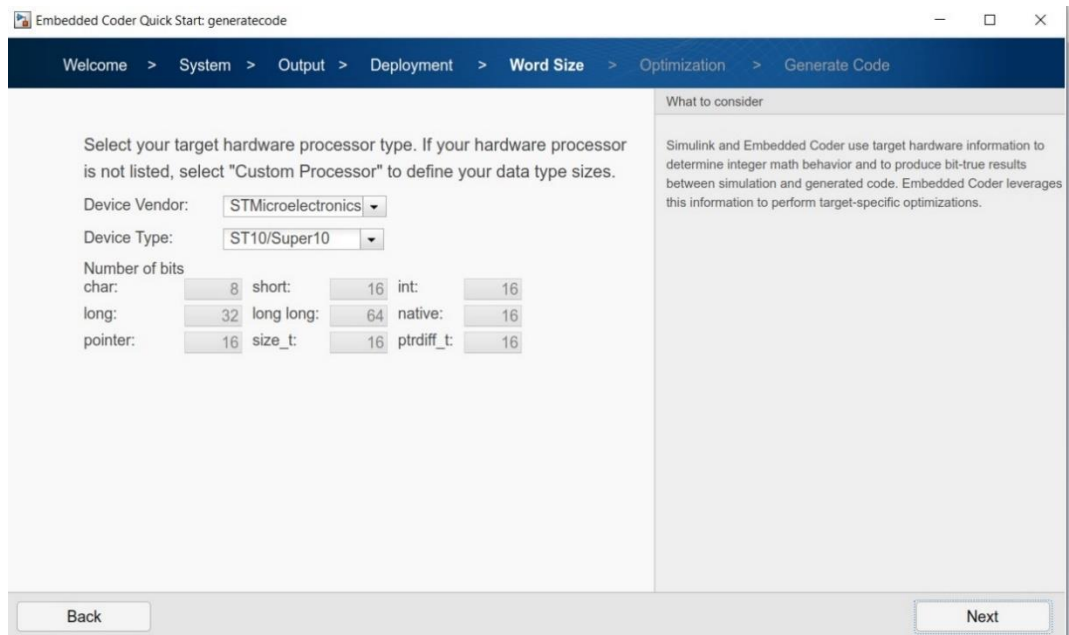
Pemodelan *field oriented control* dan *lookup table field oriented control* dengan variasi beban menghasilkan respon torsi seperti yang ditunjukkan pada gambar 4.36 dan 4.37. Respon torsi yang dihasilkan motor *BLDC* pada simulasi ini sesuai dengan variasi beban yang diberikan. Respon torsi yang dihasilkan model *lookup table field oriented control* memiliki nilai *percent overshoot* lebih rendah dibandingkan dengan respon torsi yang dihasilkan model *field oriented control*, hal ini menunjukkan pada model *simulink lookup table field oriented control* memiliki *delivery torque* yang lebih baik dibandingkan model *simulink field oriented control*.

4.5. Generate Simulasi *Lookup Table Field Oriented Control* ke C code

Tesis ini menggunakan model *look up table field oriented control* di *generate* menjadi bahasa C seperti yang ditunjukkan Gambar 4.38 untuk diimplementasikan ke bahasa pemrograman pada *controller*. Dengan melakukan *generate code* pada simulasi *look up table field oriented control* dapat membantu peneliti selanjutnya untuk melakukan implementasi dan validasi ke *BLDC motor controller*.

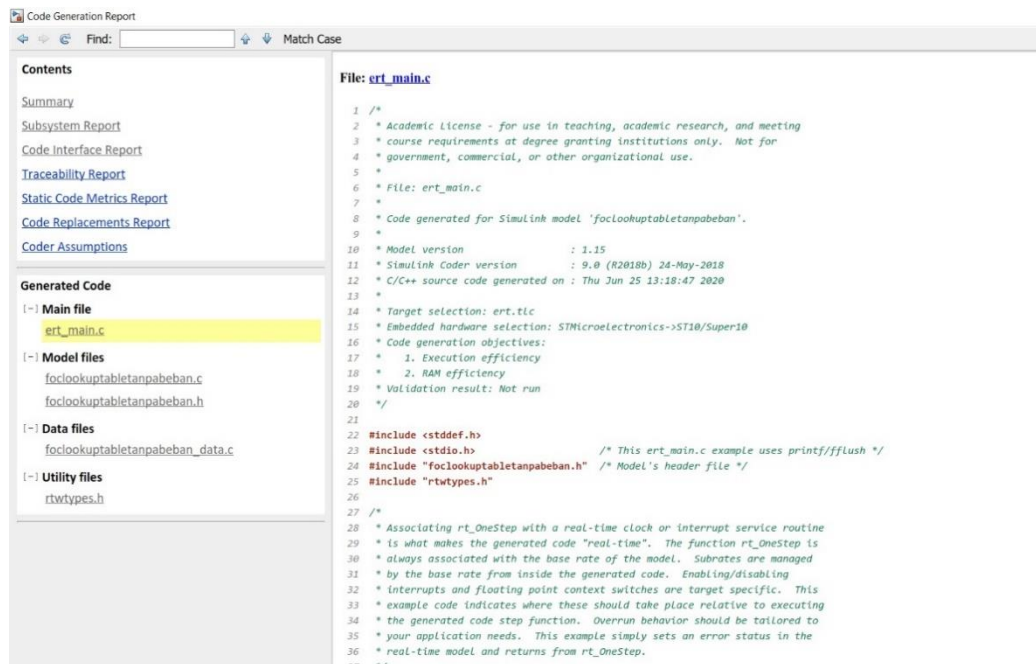


Gambar 4.38. Proses *Generate code* menjadi bahasa C



Gambar 4.39. Pemilihan target *hardware processor*

Model *lookup table field oriented control* berhasil di *generate code* menjadi bahasa C yang dapat dilihat pada gambar 4.40.



Gambar 4.40. Bahasa C dari *Generate Code* pemodelan *Lookup Table Field Oriented Control*

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Tesis ini bertujuan untuk mengatasi permasalahan yang terjadi pada *controller* motor *BLDC* yaitu kondisi *starting* yang kasar. Hal tersebut terjadi karena terdapat sistem proteksi berupa limit torsi atau arus yang mampu menonaktifkan *controller* ketika torsi atau arus mencapai limit. Guna mengatasi permasalahan tersebut peneliti melakukan pemodelan *vehicle dynamic* pada kendaraan listrik untuk mengetahui nilai *power*, torsi, dan rpm yang dibutuhkan serta digunakan sebagai spesifikasi motor *BLDC* selanjutnya. Adapun spesifikasi motor *BLDC* yang diperoleh dari hasil pemodelan *vehicle dynamic* yaitu *rated power* sebesar 5 kw, torsi sebesar 24 Nm, dan rpm sebesar 3000 rpm. Sedangkan untuk hasil pengukuran parameter motor pada kondisi aktual diperoleh nilai resistansi sebesar 0,043 Ω , induktansi sebesar 105,26 μH , konstanta BEMF sebesar 18,95 volt/krpm, konstanta torsi sebesar 0,18085 Nm/A, konstanta waktu mekanis sebesar 0,15 detik, koefisien gaya gesek sebesar 0,016158 Nms, dan momen inersia sebesar 0,059 kgm^2 . Kemudian nilai parameter yang telah diperoleh tersebut digunakan sebagai parameter *block diagram PMSM* pada *simulink*.

Pada pemodelan *lookup table field oriented control* dengan nilai parameter PI *default* ($k_p = 1$ dan $k_i = 1$) diperoleh hasil simulasi yang ditinjau dari respon rpm berupa nilai *rise time* sebesar 0,6445 detik, *settling time* sebesar 1,979 detik, dan nilai *percent overshoot* sebesar 0%. Sedangkan untuk hasil simulasi setelah melakukan proses *tuning* parameter ($k_p = 10$ dan $k_i = 0,5$) diperoleh nilai *rise time* sebesar 0,5522 detik, *settling time* sebesar 0,878 detik, dan *percent overshoot* sebesar 0%. Sementara pada pemodelan *field oriented control* diperoleh hasil simulasi yang ditinjau dari respon rpm berupa nilai *rise time* sebesar 0,6 detik, *settling time* sebesar 1 detik, dan *percent overshoot* sebesar 0%. Pada pemodelan *field oriented control vedder* diperoleh hasil simulasi yang ditinjau dari respon

rpm berupa *rise time* sebesar 0,65 detik, *settling time* sebesar 1,5 detik, dan *percent overshoot* sebesar 0%.

Pada pemodelan *lookup table field oriented control* diperoleh hasil simulasi yang ditinjau dari respon torsi berupa nilai *rise time* sebesar 0,001 detik, *settling time* sebesar 0,002detik, dan *percent overshoot* sebesar 4%. Sedangkan pemodelan *field oriented control* memperoleh hasil simulasi ditinjau dari respon torsi berupa nilai *rise time* sebesar 0,001 detik, *settling time* sebesar 0,003 detik, dan *percent overshoot* sebesar 12,5%. Sementara pada pemodelan pemodelan *field oriented control vedder* diperoleh hasil simulasi yang ditinjau dari respon torsi berupa nilai *rise time* sebesar 0,001 detik, *settling time* dan *percent overshoot* yang masing-masing tidak tercapai. Berdasarkan hasil yang telah dijabarkan diatas maka dapat ditarik kesimpulan bahwa pemodelan *look up table field oriented control* memiliki hasil respon yang lebih baik dibandingkan dengan pemodelan *field oriented control* dan *field oriented control vedder*. Kemudian pemodelan *look up table field oriented control* dirubah menjadi bahasa C untuk diimplementasikan pada *controller*. Hal ini dilakukan bertujuan untuk membantu dan mempermudah peneliti selanjutnya dalam melakukan implementasi serta validasi pada *controller* motor *BLDC*.

5.2. Saran

Adapun saran peneliti yang dapat digunakan untuk penelitian selanjutnya ialah melakukan implementasi dan validasi dari hasil pemodelan *lookup table field oriented* pada *controller* motor *BLDC*.

DAFTAR PUSTAKA

- Bose, B. K. (2000). *Energy, environment, and advances in power electronics*. IEEE Transactions on Power Electronics: pages 688–701.
- Chen, Ken. (2018). Why choose VESC? FLIPSKY. [Online] 24 December 2018. [Dikutip: 21 July 2020.] <https://flipsky.net/blogs/vesc-tool/why-choose-vesc>.
- Close, C., Frederick, D., & Newell, J. (2002). *Modelling and Analysis of Dynamic Systems*. New York: John Wiley & Sons.
- Hong-xing Wu, Shu-kang Cheng, & Shu-mei Cui. (2005). *A controller of brushless DC motor for electric vehicle*. IEEE Transactions on Magnetics: pages 509–513.
- Jacob, J., & Chitra, A. (2017). *Field Oriented Control of Space Vector Modulated Multilevel Inverter fed PMSM Drive*. Energy Procedia: pages 117 & 966–973.
- Kommula, B. N., & Kota, V. R. (2015). *Performance evaluation of Hybrid Fuzzy PI speed controller for Brushless DC motor for Electric vehicle application*. Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG).
- Lee, B. K., & Ehsani, M. (n.d.). (2001). *Advanced BLDC motor drive for low cost and high performance propulsion system in electric and hybrid vehicles*. IEMDC 2001. IEEE International Electric Machines and Drives Conference (Cat. No.01EX485).
- Shah, V., & N, K. P. (2017). *FPGA Implementation of Sensor less Field Oriented Current Control of Induction Machine*. IEEE International Conference on Computational Intelligence and Computing Research (ICCIC).
- Singh, M., & Garg, A. (2012). *Performance evaluation of BLDC motor with conventional PI and fuzzy speed controller*. IEEE 5th India International Conference on Power Electronics (IICPE).
- Somanatham, R., Prasad, P. V. N., & Rajkumar, A. D. (n.d.). (2014). *Modeling and simulation of sensorless control of PMBLDC motor using zero-crossing back E.M.F. detection*. International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM).
- Stefan Badurson. (2005). *BLDC Motor Modelling and Control – A Matlab/Simulink Implementation*. Chalmer Tekniska Hogskola: Electrical Power Engineering.
- Yildirim, M., Polat, M., & Kurum, H. (2014). *A survey on comparison of electric motor types and drives used for electric vehicles*. 16th International Power Electronics and Motion Control Conference and Exposition.

(halaman ini sengaja dikosongkan)

Lampiran

Hasil generate Code model Field Oriented Control

Main

```
1  /*
2  * Academic License - for use in teaching, academic research, and meeting
3  * course requirements at degree granting institutions only. Not for
4  * government, commercial, or other organizational use.
5  *
6  * File: ert_main.c
7  *
8  * Code generated for Simulink model 'foclookuptabletanpabeban'.
9  *
10 * Model version : 1.15
11 * Simulink Coder version : 9.0 (R2018b) 24-May-2018
12 * C/C++ source code generated on : Thu Jun 25 13:18:47 2020
13 *
14 * Target selection: ert.tlc
15 * Embedded hardware selection: STMicroelectronics->ST10/Super10
16 * Code generation objectives:
17 * 1. Execution efficiency
18 * 2. RAM efficiency
19 * Validation result: Not run
20 */
21
22 #include <stddef.h>
23 #include <stdio.h> /* This ert_main.c example uses printf/fflush */
```

```

24 #include "foclookuptabletanpabeban.h" /* Model's header file */
25 #include "rtwtypes.h"
26
27 /*
28  * Associating rt_OneStep with a real-time clock or interrupt service routine
29  * is what makes the generated code "real-time". The function rt_OneStep is
30  * always associated with the base rate of the model. Subrates are managed
31  * by the base rate from inside the generated code. Enabling/disabling
32  * interrupts and floating point context switches are target specific. This
33  * example code indicates where these should take place relative to executing
34  * the generated code step function. Overrun behavior should be tailored to
35  * your application needs. This example simply sets an error status in the
36  * real-time model and returns from rt_OneStep.
37 */
38 void rt_OneStep(void);
39 void rt_OneStep(void)
40 {
41 static boolean_T OverrunFlag = false;
42
43 /* Disable interrupts here */
44
45 /* Check for overrun */
46 if (OverrunFlag) {
47   rtmSetErrorStatus(rtm, "Overrun");
48 return;
49 }
50

```

```

51  OverrunFlag = true;
52
53  /* Save FPU context here (if necessary) */
54  /* Re-enable timer or interrupt here */
55  /* Set model inputs here */
56
57  /* Step the model for base rate */
58  foclookuptabletanpabeban_step();
59
60  /* Get model outputs here */
61
62  /* Indicate task complete */
63  OverrunFlag = false;
64
65  /* Disable interrupts here */
66  /* Restore FPU context here (if necessary) */
67  /* Enable interrupts here */
68  }
69
70  /*
71  * The example "main" function illustrates what is required by your
72  * application code to initialize, execute, and terminate the generated code.
73  * Attaching rt_OneStep to a real-time clock is target specific. This example
74  * illustrates how you do this relative to initializing the model.
75  */
76  int_T main(int_T argc, const char *argv[])
77  {

```

```

78  /* Unused arguments */
79  (void)(argc);
80  (void)(argv);
81
82  /* Initialize model */
83  foclookuptabletanpabeban_initialize();
84
85  /* Simulating the model step behavior (in non real-time) to
86  * simulate model behavior at stop time.
87  */
88  while ((rtmGetErrorStatus(rtM) == (NULL)) &&
      !rtmGetStopRequested(rtM)) {
89    rt_OneStep();
90  }
91
92  /* Disable rt_OneStep() here */
93  return 0;
94  }
95
96  /*
97  * File trailer for generated code.
98  *
99  * [EOF]
100 */
101

```


File: foclookuptabletanpabeban.c

```
1  /*
2  * Academic License - for use in teaching, academic research, and meeting
3  * course requirements at degree granting institutions only. Not for
4  * government, commercial, or other organizational use.
5  *
6  * File: foclookuptabletanpabeban.c
7  *
8  * Code generated for Simulink model 'foclookuptabletanpabeban'.
9  *
10 * Model version : 1.15
11 * Simulink Coder version : 9.0 (R2018b) 24-May-2018
12 * C/C++ source code generated on : Thu Jun 25 13:18:47 2020
13 *
14 * Target selection: ert.tlc
15 * Embedded hardware selection: STMicroelectronics->ST10/Super10
16 * Code generation objectives:
17 * 1. Execution efficiency
18 * 2. RAM efficiency
19 * Validation result: Not run
20 */
21
22 #include "foclookuptabletanpabeban.h"
23 #include <math.h>
24 #include <stdlib.h>
25 #define NumBitsPerChar 8U
26
27 /* Private macros used by the generated code to access rtModel */
28 #ifndef rtmIsMajorTimeStep
29 # define rtmIsMajorTimeStep(rtm) (((rtm)->Timing.simTimeStep) ==
    MAJOR_TIME_STEP)
30 #endif
```

```

31
32 #ifndef rtmIsMinorTimeStep
33 #define rtmIsMinorTimeStep(rtm) (((rtm)->Timing.simTimeStep) ==
    MINOR_TIME_STEP)
34 #endif
35
36 #ifndef rtmSetTPtr
37 #define rtmSetTPtr(rtm, val) ((rtm)->Timing.t = (val))
38 #endif
39
40 #ifndef CodeFormat
41 #define CodeFormat S-Function
42 #else
43 #undef CodeFormat
44 #define CodeFormat S-Function
45 #endif
46
47 #ifndef S_FUNCTION_NAME
48 #define S_FUNCTION_NAME simulink_only_sfcn
49 #else
50 #undef S_FUNCTION_NAME
51 #define S_FUNCTION_NAME simulink_only_sfcn
52 #endif
53
54 #ifndef S_FUNCTION_LEVEL
55 #define S_FUNCTION_LEVEL 2
56 #else
57 #undef S_FUNCTION_LEVEL
58 #define S_FUNCTION_LEVEL 2
59 #endif
60
61 #ifndef RTW_GENERATED_S_FUNCTION
62 #define RTW_GENERATED_S_FUNCTION
63 #endif
64
65 #ifndef rtmGetDataMapInfo
66 #define rtmGetDataMapInfo(rtm) NULL
67 #endif
68

```

```

69 #ifndef rtmSetDataMapInfo
70 # define rtmSetDataMapInfo(rtm, val)
71 #endif
72
73 #if !defined(RTW_SFUNCTION_DEFINES)
74 #define RTW_SFUNCTION_DEFINES
75 #ifndef _RTW_COMMON_DEFINES_
76 #define _RTW_COMMON_DEFINES_
77 #endif
78 #endif
79
80 /* Used by FromWorkspace Block: '<S9>/FromWs' */
81 #ifndef rtInterpolate
82 # define rtInterpolate(v1,v2,f1,f2) (((v1)==(v2))?((double)(v1)):
      (((f1)*((double)(v1)))+((f2)*((double)(v2)))))
83 #endif
84
85 #ifndef rtRound
86 # define rtRound(v) ( ((v) >= 0) ? floor((v) + 0.5) : ceil((v) - 0.5) )
87 #endif
88
89 const real_T foclookuptabletanpabeban_RGND = 0.0;/* real_T ground */
90
91 /* Continuous states */
92 X rtX;
93
94 /* Block signals and states (default storage) */
95 DW rtDW;
96
97 /* Real-time model */
98 RT_MODEL rtM_;
99 RT_MODEL *const rtM = &rtM_;
100 static real_T look2_binlx(real_T u0, real_T u1, const real_T bp0[], const
      real_T
101 bp1[], const real_T table[], const uint32_T maxIndex[], uint32_T stride);
102
103 /* private model entry point functions */
104 extern void foclookuptabletanpabeban_derivatives(void);
105 static void rate_scheduler(void);

```

```

106 extern real_T rtGetNaN(void);
107 extern real32_T rtGetNaNF(void);
108 extern real_T rtInf;
109 extern real_T rtMinusInf;
110 extern real_T rtNaN;
111 extern real32_T rtInfF;
112 extern real32_T rtMinusInfF;
113 extern real32_T rtNaNF;
114 extern void rt_InitInfAndNaN(size_t realSize);
115 extern boolean_T rtIsInf(real_T value);
116 extern boolean_T rtIsInfF(real32_T value);
117 extern boolean_T rtIsNaN(real_T value);
118 extern boolean_T rtIsNaNF(real32_T value);
119 typedef struct {
120 struct {
121     uint32_T wordH;
122     uint32_T wordL;
123 } words;
124 } BigEndianIEEEDouble;
125
126 typedef struct {
127 struct {
128     uint32_T wordL;
129     uint32_T wordH;
130 } words;
131 } LittleEndianIEEEDouble;
132
133 typedef struct {
134 union {
135     real32_T wordLreal;
136     uint32_T wordLuint;
137 } wordL;
138 } IEEESingle;
139
140 real_T rtInf;
141 real_T rtMinusInf;
142 real_T rtNaN;
143 real32_T rtInfF;
144 real32_T rtMinusInfF;

```

```

145 real32_T rtNaNF;
146 extern real_T rtGetInf(void);
147 extern real32_T rtGetInfF(void);
148 extern real_T rtGetMinusInf(void);
149 extern real32_T rtGetMinusInfF(void);
150
151 /*
152  * Initialize rtNaN needed by the generated code.
153  * NaN is initialized as non-signaling. Assumes IEEE.
154  */
155 real_T rtGetNaN(void)
156 {
157     size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
158     real_T nan = 0.0;
159     if (bitsPerReal == 32U) {
160         nan = rtGetNaNF();
161     } else {
162         union {
163             LittleEndianIEEEDouble bitVal;
164             real_T fltVal;
165         } tmpVal;
166
167         tmpVal.bitVal.words.wordH = 0xFFFF80000U;
168         tmpVal.bitVal.words.wordL = 0x000000000U;
169         nan = tmpVal.fltVal;
170     }
171
172     return nan;
173 }
174
175 /*
176  * Initialize rtNaNF needed by the generated code.
177  * NaN is initialized as non-signaling. Assumes IEEE.
178  */
179 real32_T rtGetNaNF(void)
180 {
181     IEEESingle nanF = { { 0 } };
182
183     nanF.wordL.wordLuint = 0xFFC00000U;

```

```

184 return nanF.wordL.wordLreal;
185 }
186
187 /*
188  * Initialize the rtInf, rtMinusInf, and rtNaN needed by the
189  * generated code. NaN is initialized as non-signaling. Assumes IEEE.
190  */
191 void rt_InitInfAndNaN(size_t realSize)
192 {
193     (void) (realSize);
194     rtNaN = rtGetNaN();
195     rtNaNF = rtGetNaNF();
196     rtInf = rtGetInf();
197     rtInfF = rtGetInfF();
198     rtMinusInf = rtGetMinusInf();
199     rtMinusInfF = rtGetMinusInfF();
200 }
201
202 /* Test if value is infinite */
203 boolean_T rtIsInf(real_T value)
204 {
205     return (boolean_T)((value==rtInf || value==rtMinusInf) ? 1U : 0U);
206 }
207
208 /* Test if single-precision value is infinite */
209 boolean_T rtIsInfF(real32_T value)
210 {
211     return (boolean_T)(((value)==rtInfF || (value)==rtMinusInfF) ? 1U : 0U);
212 }
213
214 /* Test if value is not a number */
215 boolean_T rtIsNaN(real_T value)
216 {
217     boolean_T result = (boolean_T) 0;
218     size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
219     if (bitsPerReal == 32U) {
220         result = rtIsNaNF((real32_T)value);
221     } else {
222         union {

```

```

223 LittleEndianIEEEDouble bitVal;
224 real_T fltVal;
225 } tmpVal;
226
227 tmpVal.fltVal = value;
228 result = (boolean_T)((tmpVal.bitVal.words.wordH & 0x7FF00000) ==
    0x7FF00000 &&
229 ( (tmpVal.bitVal.words.wordH & 0x000FFFFFF) != 0 ||
230 (tmpVal.bitVal.words.wordL != 0) ));
231 }
232
233 return result;
234 }
235
236 /* Test if single-precision value is not a number */
237 boolean_T rtIsNaNF(real32_T value)
238 {
239 IEEESingle tmp;
240 tmp.wordL.wordLreal = value;
241 return (boolean_T)( (tmp.wordL.wordLuint & 0x7F800000) ==
    0x7F800000 &&
242 (tmp.wordL.wordLuint & 0x007FFFFFF) != 0 );
243 }
244
245 /*
246 * Initialize rtInf needed by the generated code.
247 * Inf is initialized as non-signaling. Assumes IEEE.
248 */
249 real_T rtGetInf(void)
250 {
251 size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
252 real_T inf = 0.0;
253 if (bitsPerReal == 32U) {
254 inf = rtGetInfF();
255 } else {
256 union {
257 LittleEndianIEEEDouble bitVal;
258 real_T fltVal;
259 } tmpVal;

```

```

260
261 tmpVal.bitVal.words.wordH = 0x7FF00000U;
262 tmpVal.bitVal.words.wordL = 0x00000000U;
263 inf = tmpVal.fltVal;
264 }
265
266 return inf;
267 }
268
269 /*
270  * Initialize rtInfF needed by the generated code.
271  * Inf is initialized as non-signaling. Assumes IEEE.
272  */
273 real32_T rtGetInfF(void)
274 {
275     IEEESingle infF;
276     infF.wordL.wordLuint = 0x7F800000U;
277     return infF.wordL.wordLreal;
278 }
279
280 /*
281  * Initialize rtMinusInf needed by the generated code.
282  * Inf is initialized as non-signaling. Assumes IEEE.
283  */
284 real_T rtGetMinusInf(void)
285 {
286     size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
287     real_T minf = 0.0;
288     if (bitsPerReal == 32U) {
289         minf = rtGetMinusInfF();
290     } else {
291         union {
292             LittleEndianIEEEDouble bitVal;
293             real_T fltVal;
294         } tmpVal;
295
296         tmpVal.bitVal.words.wordH = 0xFFF00000U;
297         tmpVal.bitVal.words.wordL = 0x00000000U;
298         minf = tmpVal.fltVal;

```



```

299 }
300
301 return minf;
302 }
303
304 /*
305  * Initialize rtMinusInfF needed by the generated code.
306  * Inf is initialized as non-signaling. Assumes IEEE.
307  */
308 real32_T rtGetMinusInfF(void)
309 {
310     IEEESingle minfF;
311     minfF.wordL.wordLuint = 0xFF800000U;
312     return minfF.wordL.wordLreal;
313 }
314
315 static real_T look2_binlx(real_T u0, real_T u1, const real_T bp0[], const
    real_T
316     bp1[], const real_T table[], const uint32_T maxIndex[], uint32_T stride)
317 {
318     real_T frac;
319     uint32_T bpIndices[2];
320     real_T fractions[2];
321     real_T yL_1d;
322     uint32_T iRght;
323     uint32_T bpIdx;
324     uint32_T iLeft;
325
326     /* Column-major Lookup 2-D
327      Search method: 'binary'
328      Use previous index: 'off'
329      Interpolation method: 'Linear point-slope'
330      Extrapolation method: 'Linear'
331      Use last breakpoint for index at or above upper limit: 'off'
332      Remove protection against out-of-range input in generated code: 'off'
333      */
334     /* Prelookup - Index and Fraction
335      Index Search method: 'binary'
336      Extrapolation method: 'Linear'

```

```

337 Use previous index: 'off'
338 Use last breakpoint for index at or above upper limit: 'off'
339 Remove protection against out-of-range input in generated code: 'off'
340 */
341 if (u0 <= bp0[0UL]) {
342 iLeft = 0UL;
343 frac = (u0 - bp0[0UL]) / (bp0[1UL] - bp0[0UL]);
344 } else if (u0 < bp0[maxIndex[0UL]]) {
345 /* Binary Search */
346 bpIdx = maxIndex[0UL] >> 1UL;
347 iLeft = 0UL;
348 iRght = maxIndex[0UL];
349 while (iRght - iLeft > 1UL) {
350 if (u0 < bp0[bpIdx]) {
351 iRght = bpIdx;
352 } else {
353 iLeft = bpIdx;
354 }
355
356 bpIdx = (iRght + iLeft) >> 1UL;
357 }
358
359 frac = (u0 - bp0[iLeft]) / (bp0[iLeft + 1UL] - bp0[iLeft]);
360 } else {
361 iLeft = maxIndex[0UL] - 1UL;
362 frac = (u0 - bp0[maxIndex[0UL] - 1UL]) / (bp0[maxIndex[0UL]] -
    bp0[maxIndex
363 [0UL] - 1UL]);
364 }
365
366 fractions[0UL] = frac;
367 bpIndices[0UL] = iLeft;
368
369 /* Prelookup - Index and Fraction
370 Index Search method: 'binary'
371 Extrapolation method: 'Linear'
372 Use previous index: 'off'
373 Use last breakpoint for index at or above upper limit: 'off'
374 Remove protection against out-of-range input in generated code: 'off'

```

```

375  */
376  if (u1 <= bp1[0UL]) {
377    iLeft = 0UL;
378    frac = (u1 - bp1[0UL]) / (bp1[1UL] - bp1[0UL]);
379  } else if (u1 < bp1[maxIndex[1UL]]) {
380    /* Binary Search */
381    bpIdx = maxIndex[1UL] >> 1UL;
382    iLeft = 0UL;
383    iRght = maxIndex[1UL];
384    while (iRght - iLeft > 1UL) {
385      if (u1 < bp1[bpIdx]) {
386        iRght = bpIdx;
387      } else {
388        iLeft = bpIdx;
389      }
390
391      bpIdx = (iRght + iLeft) >> 1UL;
392    }
393
394    frac = (u1 - bp1[iLeft]) / (bp1[iLeft + 1UL] - bp1[iLeft]);
395  } else {
396    iLeft = maxIndex[1UL] - 1UL;
397    frac = (u1 - bp1[maxIndex[1UL] - 1UL]) / (bp1[maxIndex[1UL]] -
        bp1[maxIndex
398    [1UL] - 1UL]);
399  }
400
401  /* Column-major Interpolation 2-D
402  Interpolation method: 'Linear point-slope'
403  Use last breakpoint for index at or above upper limit: 'off'
404  Overflow mode: 'wrapping'
405  */
406  bpIdx = iLeft * stride + bpIndices[0UL];
407  yL_1d = (table[bpIdx + 1UL] - table[bpIdx]) * fractions[0UL] +
    table[bpIdx];
408  bpIdx += stride;
409  return (((table[bpIdx + 1UL] - table[bpIdx]) * fractions[0UL] +
    table[bpIdx])
410  - yL_1d) * frac + yL_1d;

```

```

411 }
412
413 /*
414  * This function updates active task flag for each substrate.
415  * The function is called at model base rate, hence the
416  * generated code self-manages all its substrates.
417  */
418 static void rate_scheduler(void)
419 {
420  /* Compute which substrates run during the next base time step. Subrates
421  * are an integer multiple of the base rate counter. Therefore, the subtask
422  * counter is reset when it reaches its limit (zero means run).
423  */
424  (rtM->Timing.TaskCounters.TID[2])++;
425  if ((rtM->Timing.TaskCounters.TID[2]) > 19) { /* Sample time: [0.0001s,
426  0.0s] */
427  }
428 }
429
430 /*
431  * This function updates continuous states using the ODE3 fixed-step
432  * solver algorithm
433  */
434 static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
435 {
436  /* Solver Matrices */
437  static const real_T rt_ODE3_A[3] = {
438  1.0/2.0, 3.0/4.0, 1.0
439  };
440
441  static const real_T rt_ODE3_B[3][3] = {
442  { 1.0/2.0, 0.0, 0.0 },
443
444  { 0.0, 3.0/4.0, 0.0 },
445
446  { 2.0/9.0, 1.0/3.0, 4.0/9.0 }
447  };
448

```

```

449 time_T t = rtsiGetT(si);
450 time_T tnew = rtsiGetSolverStopTime(si);
451 time_T h = rtsiGetStepSize(si);
452 real_T *x = rtsiGetContStates(si);
453 ODE3_IntgData *id = (ODE3_IntgData *)rtsiGetSolverData(si);
454 real_T *y = id->y;
455 real_T *f0 = id->f[0];
456 real_T *f1 = id->f[1];
457 real_T *f2 = id->f[2];
458 real_T hB[3];
459 int_T i;
460 int_T nXc = 7;
461 rtsiSetSimTimeStep(si, MINOR_TIME_STEP);
462
463 /* Save the state values at time t in y, we'll use x as ynew. */
464 (void) memcpy(y, x,
465 (uint_T)nXc*sizeof(real_T));
466
467 /* Assumes that rtsiSetT and ModelOutputs are up-to-date */
468 /* f0 = f(t,y) */
469 rtsiSetdX(si, f0);
470 foclookuptabletanpabeban_derivatives();
471
472 /* f(:,2) = feval(odefile, t + hA(1), y + f*hB(:,1), args:)(*)); */
473 hB[0] = h * rt_ODE3_B[0][0];
474 for (i = 0; i < nXc; i++) {
475 x[i] = y[i] + (f0[i]*hB[0]);
476 }
477
478 rtsiSetT(si, t + h*rt_ODE3_A[0]);
479 rtsiSetdX(si, f1);
480 foclookuptabletanpabeban_step();
481 foclookuptabletanpabeban_derivatives();
482
483 /* f(:,3) = feval(odefile, t + hA(2), y + f*hB(:,2), args:)(*)); */
484 for (i = 0; i <= 1; i++) {
485 hB[i] = h * rt_ODE3_B[1][i];
486 }
487

```

```

488 for (i = 0; i < nXc; i++) {
489   x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1]);
490 }
491
492 rtsiSetT(si, t + h*rt_ODE3_A[1]);
493 rtsiSetdX(si, f2);
494 foclookuptabletanpabeban_step();
495 foclookuptabletanpabeban_derivatives();
496
497 /* tnew = t + hA(3);
498 ynew = y + f*hB(:,3); */
499 for (i = 0; i <= 2; i++) {
500   hB[i] = h * rt_ODE3_B[2][i];
501 }
502
503 for (i = 0; i < nXc; i++) {
504   x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1] + f2[i]*hB[2]);
505 }
506
507 rtsiSetT(si, tnew);
508 rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
509 }
510
511 /* Model step function */
512 void foclookuptabletanpabeban_step(void)
513 {
514   /* local block i/o variables */
515   real_T rtb_b;
516   real_T rtb_Gain2_n;
517   real_T rtb_Gain_o;
518   real_T rtb_Sum_n;
519   real_T rtb_Sum1;
520   real_T rtb_DiscreteTimeIntegrator;
521   real_T rtb_DiscreteTimeIntegrator_o;
522   real_T rtb_Gain1_i[3];
523   real_T rtb_DiscreteTimeIntegrator_lz;
524   real_T rtb_Fcn2;
525   real_T rtb_c;
526   real_T rtb_Sum4;

```

```

527 boolean_T rtb_RelationalOperator;
528 boolean_T rtb_RelationalOperator1;
529 boolean_T rtb_RelationalOperator2;
530 real_T rtb_ElementaryMath_o1;
531 real_T rtb_ElementaryMath_o2;
532 int16_T i;
533 real_T rtb_Sum1_d_idx_0;
534 real_T rtb_Sum1_d_idx_1;
535 real_T rtb_Sum_k_idx_0;
536 real_T rtb_Sum_k_idx_1;
537 real_T rtb_c_tmp;
538 if (rtmIsMajorTimeStep(rtM)) {
539     /* set solver stop time */
540     if (!(rtM->Timing.clockTick0+1)) {
541         rtsiSetSolverStopTime(&rtM->solverInfo, ((rtM->Timing.clockTickH0 +
542             1) *
543             rtM->Timing.stepSize0 * 4294967296.0));
544     } else {
545         rtsiSetSolverStopTime(&rtM->solverInfo, ((rtM->Timing.clockTick0 + 1)
546             *
547             rtM->Timing.stepSize0 + rtM->Timing.clockTickH0 * rtM-
548             >Timing.stepSize0 *
549             4294967296.0));
550     }
551 } /* end MajorTimeStep */
552
553 /* Update absolute time of base rate at minor time step */
554 if (rtmIsMinorTimeStep(rtM)) {
555     rtM->Timing.t[0] = rtsiGetT(&rtM->solverInfo);
556 }
557
558 if (rtmIsMajorTimeStep(rtM) &&
559     rtM->Timing.TaskCounters.TID[1] == 0) {
560     /* Trigonometry: '<S225>/Elementary Math' incorporates:
561     * DiscreteIntegrator: '<S223>/Discrete-Time Integrator1'
562     */
563     rtb_ElementaryMath_o1 = sin(rtDW.DiscreteTimeIntegrator1_DSTATE);
564     rtb_ElementaryMath_o2 = cos(rtDW.DiscreteTimeIntegrator1_DSTATE);
565 }

```

```

563  /* DiscreteIntegrator: '<S230>/Discrete-Time Integrator' */
564  rtb_DiscreteTimeIntegrator = rtDW.DiscreteTimeIntegrator_DSTATE;
565
566  /* DiscreteIntegrator: '<S229>/Discrete-Time Integrator' */
567  rtb_DiscreteTimeIntegrator_o =
    rtDW.DiscreteTimeIntegrator_DSTATE_k;
568
569  /* Fcn: '<S227>/Fcn' incorporates:
570   * DiscreteIntegrator: '<S229>/Discrete-Time Integrator'
571   * DiscreteIntegrator: '<S230>/Discrete-Time Integrator'
572   */
573  rtDW.Fcn = rtDW.DiscreteTimeIntegrator_DSTATE *
    rtb_ElementaryMath_o2 +
574  rtDW.DiscreteTimeIntegrator_DSTATE_k * rtb_ElementaryMath_o1;
575
576  /* Fcn: '<S227>/Fcn1' incorporates:
577   * DiscreteIntegrator: '<S229>/Discrete-Time Integrator'
578   * DiscreteIntegrator: '<S230>/Discrete-Time Integrator'
579   */
580  rtDW.Fcn1 = ((-rtDW.DiscreteTimeIntegrator_DSTATE -
    1.7320508075688772 *
581  rtDW.DiscreteTimeIntegrator_DSTATE_k) * rtb_ElementaryMath_o2
582  + (1.7320508075688772 * rtDW.DiscreteTimeIntegrator_DSTATE -
583  rtDW.DiscreteTimeIntegrator_DSTATE_k) *
584  rtb_ElementaryMath_o1) * 0.5;
585
586  /* S-Function (sfun_spssw_discc): '<S232>/State-Space' incorporates:
587   * Constant: '<S208>/DC'
588   */
589
590  /* S-Function block: '<S232>/State-Space' */
591  {
592  real_T accum;
593
594  /* Circuit has switches */
595  int_T *switch_status = (int_T*)
    rtDW.StateSpace_PWORK.SWITCH_STATUS;
596  int_T *switch_status_init = (int_T*)
597  rtDW.StateSpace_PWORK.SWITCH_STATUS_INIT;

```



```

598 int_T *SwitchChange = (int_T*) rtDW.StateSpace_PWORK.SW_CHG;
599 int_T *gState = (int_T*) rtDW.StateSpace_PWORK.G_STATE;
600 real_T *yswitch = (real_T*)rtDW.StateSpace_PWORK.Y_SWITCH;
601 int_T *switchTypes = (int_T*)
    rtDW.StateSpace_PWORK.SWITCH_TYPES;
602 int_T *idxOutSw = (int_T*) rtDW.StateSpace_PWORK.IDX_OUT_SW;
603 real_T *DxCol = (real_T*)rtDW.StateSpace_PWORK.DX_COL;
604 real_T *tmp2 = (real_T*)rtDW.StateSpace_PWORK.TMP2;
605 real_T *uswlast = (real_T*)rtDW.StateSpace_PWORK.USWLAST;
606 int_T newState;
607 int_T swChanged = 0;
608 int loopsToDo = 20;
609 real_T temp;
610
611 /* keep an initial copy of switch_status*/
612 memcpy(switch_status_init, switch_status, 6 * sizeof(int_T));
613 memcpy(uswlast, &rtDW.StateSpace_o1[0], 6*sizeof(real_T));
614 do {
615     if (loopsToDo == 1) { /* Need to reset some variables: */
616         swChanged = 0;
617
618         /* return to the original switch status*/
619         {
620             int_T i1;
621             for (i1=0; i1 < 6; i1++) {
622                 swChanged = ((SwitchChange[i1] = switch_status_init[i1] -
623                     switch_status[i1]) != 0) ? 1 : swChanged;
624                 switch_status[i1] = switch_status_init[i1];
625             }
626         }
627     } else {
628         /*
629         * Compute outputs:
630         * -----
631         */
632         real_T *Ds = (real_T*)rtDW.StateSpace_PWORK.DS;
633
634         {
635             int_T i1;

```

```

636 real_T *y0 = &rtDW.StateSpace_o1[0];
637 for (i1=0; i1 < 11; i1++) {
638 accum = 0.0;
639
640 {
641 int_T i2;
642 const real_T *u0;
643 for (i2=0; i2 < 6; i2++) {
644 accum += *(Ds++) * 0.0;
645 }
646
647 accum += *(Ds++) * rtDW.Fcn;
648 accum += *(Ds++) * rtDW.Fcn1;
649 accum += *(Ds++) * 400.0;
650 }
651
652 y0[i1] = accum;
653 }
654 }
655
656 swChanged = 0;
657
658 {
659 int_T i1;
660 real_T *y0 = &rtDW.StateSpace_o1[0];
661 for (i1=0; i1 < 6; i1++) {
662 newState = ((y0[i1] > 0.0) && (gState[i1] > 0)) || (y0[i1] < 0.0) ?
663 1 : (((y0[i1] > 0.0) && gState[i1] == 0) ? 0 : switch_status[i1]);
664 swChanged = ((SwitchChange[i1] = newState - switch_status[i1]) !=
665 0) ? 1 : swChanged;
666 switch_status[i1] = newState; /* Keep new state */
667 }
668 }
669 }
670
671 /*
672  * Compute new As, Bs, Cs and Ds matrixes:
673  * -----
674  */

```

```

675  if (swChanged) {
676  real_T *Ds = (real_T*)rtDW.StateSpace_PWORK.DS;
677  real_T a1;
678
679  {
680  int_T i1;
681  for (i1=0; i1 < 6; i1++) {
682  if (SwitchChange[i1] != 0) {
683  a1 = 1000.0*SwitchChange[i1];
684  temp = 1/(1-Ds[i1*10]*a1);
685
686  {
687  int_T i2;
688  for (i2=0; i2 < 11; i2++) {
689  DxCol[i2]= Ds[i2 * 9 + i1]*temp*a1;
690  }
691  }
692
693  DxCol[i1] = temp;
694
695  /* Copy row nSw of Ds into tmp2 and zero it out in Ds */
696  memcpy(tmp2, &Ds[i1 * 9], 9 * sizeof(real_T));
697  memset(&Ds[i1 * 9], '\0', 9 * sizeof(real_T));
698
699  /* Cs = Cs + DxCol * tmp1, Ds = Ds + DxCol * tmp2
        *****/
700  {
701  int_T i2;
702  for (i2=0; i2 < 11; i2++) {
703  a1 = DxCol[i2];
704
705  {
706  int_T i3;
707  for (i3=0; i3 < 9; i3++) {
708  Ds[i2 * 9 + i3] += a1 * tmp2[i3];
709  }
710  }
711  }
712  }

```

```

713 }
714 }
715 }
716 /* if (swChanged) */
717 } while (swChanged > 0 && --loopsToDo > 0);
718
719 if (loopsToDo == 0) {
720   real_T *Ds = (real_T*)rtDW.StateSpace_PWORK.DS;
721
722   {
723     int_T i1;
724     real_T *y0 = &rtDW.StateSpace_o1[0];
725     for (i1=0; i1 < 11; i1++) {
726       accum = 0.0;
727
728       {
729         int_T i2;
730         const real_T *u0;
731         for (i2=0; i2 < 6; i2++) {
732           accum += *(Ds++) * 0.0;
733         }
734
735         accum += *(Ds++) * rtDW.Fcn;
736         accum += *(Ds++) * rtDW.Fcn1;
737         accum += *(Ds++) * 400.0;
738       }
739
740       y0[i1] = accum;
741     }
742   }
743 }
744
745 /* Output new switches states */
746 {
747   int_T i1;
748   real_T *y1 = &rtDW.StateSpace_o2[0];
749   for (i1=0; i1 < 6; i1++) {
750     y1[i1] = (real_T)switch_status[i1];
751   }

```

```

752 }
753 }
754
755 for (i = 0; i < 3; i++) {
756 /* Gain: '<S11>/Gain1' incorporates:
757 * Gain: '<S11>/Gain3'
758 */
759 rtb_Gain1_i[i] = 0.66666666666666663 * (rtConstP.Gain3_Gain[i + 6] *
760 rtDW.StateSpace_o1[10] + (rtConstP.Gain3_Gain[i + 3] *
761 rtDW.StateSpace_o1[9] + rtConstP.Gain3_Gain[i] *
762 rtDW.StateSpace_o1[8]));
763 }
764
764 /* Gain: '<Root>/Gain' incorporates:
765 * DiscreteIntegrator: '<S223>/Discrete-Time Integrator1'
766 * Fcn: '<S223>/Fcn'
767 */
768 rtDW.Gain = rtDW.DiscreteTimeIntegrator1_DSTATE / 4.0 * 4.0;
769
770 /* Fcn: '<S1>/d' incorporates:
771 * Fcn: '<S1>/q'
772 */
773 rtb_Sum1_d_idx_0 = sin(rtDW.Gain - 1.5707963267948966);
774 rtb_Sum1_d_idx_1 = cos(rtDW.Gain - 1.5707963267948966);
775 rtDW.d = rtb_Sum1_d_idx_1 * rtb_Gain1_i[0] + rtb_Sum1_d_idx_0 *
776 rtb_Gain1_i
777 [1];
778
778 /* Fcn: '<S1>/q' */
779 rtDW.q = -rtb_Sum1_d_idx_0 * rtb_Gain1_i[0] + rtb_Sum1_d_idx_1 *
780 rtb_Gain1_i[1];
781 }
782
783 /* FromWorkspace: '<S9>/FromWs' */
784 {
785 real_T *pDataValues = (real_T *) rtDW.FromWs_PWORK.DataPtr;
786 real_T *pTimeValues = (real_T *) rtDW.FromWs_PWORK.TimePtr;
787 int_T currTimeIndex = rtDW.FromWs_IWORK.PrevIndex;
788 real_T t = rtM->Timing.t[0];

```

```

789
790  /* Get index */
791  if (t <= pTimeValues[0]) {
792    currTimeIndex = 0;
793  } else if (t >= pTimeValues[6]) {
794    currTimeIndex = 5;
795  } else {
796    if (t < pTimeValues[currTimeIndex]) {
797      while (t < pTimeValues[currTimeIndex]) {
798        currTimeIndex--;
799      }
800    } else {
801      while (t >= pTimeValues[currTimeIndex + 1]) {
802        currTimeIndex++;
803      }
804    }
805  }
806
807  rtDW.FromWs_IWORK.PrevIndex = currTimeIndex;
808
809  /* Post output */
810  {
811    real_T t1 = pTimeValues[currTimeIndex];
812    real_T t2 = pTimeValues[currTimeIndex + 1];
813    if (t1 == t2) {
814      if (t < t1) {
815        rtb_b = pDataValues[currTimeIndex];
816      } else {
817        rtb_b = pDataValues[currTimeIndex + 1];
818      }
819    } else {
820      real_T f1 = (t2 - t) / (t2 - t1);
821      real_T f2 = 1.0 - f1;
822      real_T d1;
823      real_T d2;
824      int_T TimeIndex = currTimeIndex;
825      d1 = pDataValues[TimeIndex];
826      d2 = pDataValues[TimeIndex + 1];
827      rtb_b = (real_T) rtInterpolate(d1, d2, f1, f2);

```

```

828 pDataValues += 7;
829 }
830 }
831 }
832
833 if (rtmIsMajorTimeStep(rtM) &&
834 rtM->Timing.TaskCounters.TID[1] == 0) {
835  /* DiscreteIntegrator: '<S223>/Discrete-Time Integrator' */
836  rtb_DiscreteTimeIntegrator_lz =
      rtDW.DiscreteTimeIntegrator_DSTATE_p;
837
838  /* Gain: '<Root>/rad2rpm' incorporates:
839  * DiscreteIntegrator: '<S223>/Discrete-Time Integrator'
840  */
841  rtDW.rad2rpm = 9.5492965855137211 *
      rtDW.DiscreteTimeIntegrator_DSTATE_p;
842  }
843
844  /* Lookup_n-D: '<Root>/2-D Lookup Table' */
845  rtb_b = look2_binlx(rtb_b, rtDW.rad2rpm,
      rtConstP.uDLookupTable_bp01Data,
846  rtConstP.uDLookupTable_bp02Data,
847  rtConstP.uDLookupTable_tableData,
848  rtConstP.uDLookupTable_maxIndex, 10UL);
849
850  /* Gain: '<S10>/Gain1' */
851  rtb_b *= 5.5309734513274336;
852
853  /* Saturate: '<S10>/Saturation' */
854  if (rtb_b > 3.0) {
855    rtb_b = 3.0;
856  } else {
857    if (rtb_b < -3.0) {
858      rtb_b = -3.0;
859    }
860  }
861
862  /* End of Saturate: '<S10>/Saturation' */
863

```

```

864 /* Sum: '<Root>/Sum1' */
865 rtb_Sum1_d_idx_0 = rtb_b - rtDW.d;
866 rtb_Sum1_d_idx_1 = rtb_b - rtDW.q;
867
868 /* Gain: '<S77>/Filter Coefficient' incorporates:
869 * Gain: '<S44>/Derivative Gain'
870 * Integrator: '<S45>/Filter'
871 * Sum: '<S45>/SumD'
872 */
873 rtDW.FilterCoefficient[0] = (0.0 * rtb_Sum1_d_idx_0 -
rtX.Filter_CSTATE[0]) *
874 100.0;
875
876 /* Sum: '<S97>/Sum' incorporates:
877 * Gain: '<S44>/Derivative Gain'
878 * Gain: '<S84>/Proportional Gain'
879 * Integrator: '<S65>/Integrator'
880 */
881 rtb_Sum_k_idx_0 = (30.0 * rtb_Sum1_d_idx_0 +
rtX.Integrator_CSTATE[0]) +
882 rtDW.FilterCoefficient[0];
883
884 /* Gain: '<S77>/Filter Coefficient' incorporates:
885 * Gain: '<S44>/Derivative Gain'
886 * Integrator: '<S45>/Filter'
887 * Sum: '<S45>/SumD'
888 */
889 rtDW.FilterCoefficient[1] = (0.0 * rtb_Sum1_d_idx_1 -
rtX.Filter_CSTATE[1]) *
890 100.0;
891
892 /* Sum: '<S97>/Sum' incorporates:
893 * Gain: '<S44>/Derivative Gain'
894 * Gain: '<S84>/Proportional Gain'
895 * Integrator: '<S65>/Integrator'
896 */
897 rtb_Sum_k_idx_1 = (30.0 * rtb_Sum1_d_idx_1 +
rtX.Integrator_CSTATE[1]) +
898 rtDW.FilterCoefficient[1];

```



```

899 if (rtmIsMajorTimeStep(rtM) &&
900 rtM->Timing.TaskCounters.TID[1] == 0) {
901 /* Sum: '<Root>/Sum2' incorporates:
902 * Constant: '<Root>/Constant'
903 * Fcn: '<S1>/0'
904 */
905 rtb_Fcn2 = 0.0 - rtb_Gain1_i[2];
906
907 /* Gain: '<S180>/Proportional Gain' incorporates:
908 * Constant: '<Root>/Constant'
909 * Fcn: '<S1>/0'
910 * Sum: '<Root>/Sum2'
911 */
912 rtDW.ProportionalGain = (0.0 - rtb_Gain1_i[2]) * 2.0;
913
914 /* Gain: '<S140>/Derivative Gain' */
915 rtDW.DerivativeGain = 0.0 * rtb_Fcn2;
916 }
917
918 /* Integrator: '<S161>/Integrator' */
919 rtb_b = rtX.Integrator_CSTATE_j;
920
921 /* Gain: '<S173>/Filter Coefficient' incorporates:
922 * Integrator: '<S141>/Filter'
923 * Sum: '<S141>/SumD'
924 */
925 rtDW.FilterCoefficient_k = (rtDW.DerivativeGain -
rtX.Filter_CSTATE_n) * 100.0;
926
927 /* Sum: '<S193>/Sum' */
928 rtb_b = (rtDW.ProportionalGain + rtb_b) + rtDW.FilterCoefficient_k;
929
930 /* Fcn: '<S7>/alpha' incorporates:
931 * Fcn: '<S7>/beta'
932 */
933 rtb_Sum4 = sin(1.5707963267948966 - rtDW.Gain);
934 rtb_c_tmp = cos(1.5707963267948966 - rtDW.Gain);
935 rtb_c = rtb_c_tmp * rtb_Sum_k_idx_0 + rtb_Sum4 * rtb_Sum_k_idx_1;
936

```

```

937 /* Fcn: '<S7>/beta' */
938 rtb_Sum_k_idx_0 = -rtb_Sum4 * rtb_Sum_k_idx_0 + rtb_c_tmp *
    rtb_Sum_k_idx_1;
939
940 /* Fcn: '<S7>/0' */
941 rtb_Sum_k_idx_1 = rtb_b;
942
943 /* Fcn: '<S2>/a' incorporates:
944 * Fcn: '<S7>/0'
945 */
946 rtb_b += rtb_c;
947
948 /* Sum: '<S220>/Sum4' incorporates:
949 * Constant: '<S220>/Constant2'
950 * Gain: '<S220>/Gain1'
951 * Gain: '<S220>/Gain2'
952 * Integrator: '<S220>/Integrator'
953 */
954 rtb_Sum4 = 10000.0 * rtX.Integrator_CSTATE_a * 2.0 - 1.0;
955
956 /* RelationalOperator: '<S6>/Relational Operator' */
957 rtb_RelationalOperator = (rtb_b >= rtb_Sum4);
958
959 /* Fcn: '<S2>/b' */
960 rtb_b = (-0.5 * rtb_c + 0.8660254037844386 * rtb_Sum_k_idx_0) +
961 rtb_Sum_k_idx_1;
962
963 /* RelationalOperator: '<S6>/Relational Operator1' */
964 rtb_RelationalOperator1 = (rtb_b >= rtb_Sum4);
965
966 /* Fcn: '<S2>/c' */
967 rtb_c = (-0.5 * rtb_c - 0.8660254037844386 * rtb_Sum_k_idx_0) +
968 rtb_Sum_k_idx_1;
969
970 /* RelationalOperator: '<S6>/Relational Operator2' */
971 rtb_RelationalOperator2 = (rtb_c >= rtb_Sum4);
972
973 /* DataTypeConversion: '<S218>/Data Type Conversion' incorporates:
974 * Logic: '<S6>/Logical Operator'

```

```

975 * Logic: '<S6>/Logical Operator1'
976 * Logic: '<S6>/Logical Operator2'
977 */
978 rtDW.DataTypeConversion[0] = rtb_RelationalOperator;
979 rtDW.DataTypeConversion[1] = !rtb_RelationalOperator;
980 rtDW.DataTypeConversion[2] = rtb_RelationalOperator1;
981 rtDW.DataTypeConversion[3] = !rtb_RelationalOperator1;
982 rtDW.DataTypeConversion[4] = rtb_RelationalOperator2;
983 rtDW.DataTypeConversion[5] = !rtb_RelationalOperator2;
984
985 /* Gain: '<S57>/Integral Gain' */
986 rtDW.IntegralGain[0] = 0.5 * rtb_Sum1_d_idx_0;
987 rtDW.IntegralGain[1] = 0.5 * rtb_Sum1_d_idx_1;
988 if (rtmIsMajorTimeStep(rtM) &&
989 rtM->Timing.TaskCounters.TID[1] == 0) {
990 /* Gain: '<S153>/Integral Gain' */
991 rtDW.IntegralGain_a = 10.0 * rtb_Fcn2;
992
993 /* Signum: '<S231>/Sign' */
994 if (rtb_DiscreteTimeIntegrator_lz < 0.0) {
995 rtb_Fcn2 = -1.0;
996 } else if (rtb_DiscreteTimeIntegrator_lz > 0.0) {
997 rtb_Fcn2 = 1.0;
998 } else if (rtb_DiscreteTimeIntegrator_lz == 0.0) {
999 rtb_Fcn2 = 0.0;
1000 } else {
1001 rtb_Fcn2 = (rtNaN);
1002 }
1003
1004 /* End of Signum: '<S231>/Sign' */
1005
1006 /* Gain: '<S223>/Gain2' incorporates:
1007 * Fcn: '<S221>/Te '
1008 * Gain: '<S231>/Gain'
1009 * Gain: '<S231>/Gain1'
1010 * Sum: '<S223>/Sum'
1011 * Sum: '<S231>/Sum'
1012 */
1013 rtb_Gain2_n = ((0.0 * rtb_DiscreteTimeIntegrator *

```

```

1014 rtb_DiscreteTimeIntegrator_o + 0.175 *
1015 rtb_DiscreteTimeIntegrator) * 6.0 - (0.0 * rtb_Fcn2 +
1016 0.00016158 * rtb_DiscreteTimeIntegrator_lz)) * 169.49152542372883;
1017
1018 /* Gain: '<S223>/Gain' */
1019 rtb_Gain_o = 4.0 * rtb_DiscreteTimeIntegrator_lz;
1020
1021 /* Fcn: '<S225>/Fcn3' incorporates:
1022 * Fcn: '<S225>/Fcn2'
1023 */
1024 rtb_DiscreteTimeIntegrator_lz = 2.0 * rtDW.StateSpace_o1[6] +
1025 rtDW.StateSpace_o1[7];
1026
1027 /* Sum: '<S229>/Sum' incorporates:
1028 * Fcn: '<S225>/Fcn3'
1029 * Gain: '<S229>/1//Ld'
1030 * Gain: '<S229>/R//Ld'
1031 * Product: '<S229>/Product'
1032 */
1033 rtb_Sum_n = ((rtb_DiscreteTimeIntegrator_lz * rtb_ElementaryMath_o1 +
1034 -1.7320508075688772 * rtDW.StateSpace_o1[7] *
1035 rtb_ElementaryMath_o2) * 0.33333333333333331 *
1036 95.238095238095227 - 38.095238095238095 *
1037 rtb_DiscreteTimeIntegrator_o) + rtb_Gain_o *
1038 rtb_DiscreteTimeIntegrator;
1039
1040 /* Sum: '<S230>/Sum1' incorporates:
1041 * Fcn: '<S225>/Fcn2'
1042 * Gain: '<S230>/1//Lq'
1043 * Gain: '<S230>/R//Lq'
1044 * Gain: '<S230>/lam//Lq'
1045 * Product: '<S230>/Product1'
1046 */
1047 rtb_Sum1 = (((rtb_DiscreteTimeIntegrator_lz * rtb_ElementaryMath_o2 +
1048 1.7320508075688772 * rtDW.StateSpace_o1[7] *
1049 rtb_ElementaryMath_o1) * 0.33333333333333331 *
1050 95.238095238095227 - 38.095238095238095 *
1051 rtb_DiscreteTimeIntegrator) - rtb_DiscreteTimeIntegrator_o *
1052 rtb_Gain_o) - 16.666666666666664 * rtb_Gain_o;

```

```

1053 }
1054
1055 if (rtmIsMajorTimeStep(rtM) &&
1056 rtM->Timing.TaskCounters.TID[2] == 0) {
1057 /* DiscretePulseGenerator: '<S220>/Pulse Generator' */
1058 rtb_ElementaryMath_o1 = (rtDW.clockTickCounter < 1L) &&
1059 (rtDW.clockTickCounter >= 0L) ? 2.0 : 0.0;
1060 if (rtDW.clockTickCounter >= 1L) {
1061 rtDW.clockTickCounter = 0L;
1062 } else {
1063 rtDW.clockTickCounter++;
1064 }
1065
1066 /* End of DiscretePulseGenerator: '<S220>/Pulse Generator' */
1067
1068 /* Sum: '<S220>/Sum3' incorporates:
1069 * Constant: '<S220>/Constant1'
1070 */
1071 rtDW.Sum3 = rtb_ElementaryMath_o1 - 1.0;
1072 }
1073
1074 if (rtmIsMajorTimeStep(rtM)) {
1075 if (rtmIsMajorTimeStep(rtM) &&
1076 rtM->Timing.TaskCounters.TID[1] == 0) {
1077 /* Update for DiscreteIntegrator: '<S223>/Discrete-Time Integrator1' */
1078 rtDW.DiscreteTimeIntegrator1_DSTATE += 5.0E-6 * rtb_Gain_o;
1079
1080 /* Update for DiscreteIntegrator: '<S230>/Discrete-Time Integrator' */
1081 rtDW.DiscreteTimeIntegrator_DSTATE += 5.0E-6 * rtb_Sum1;
1082
1083 /* Update for DiscreteIntegrator: '<S229>/Discrete-Time Integrator' */
1084 rtDW.DiscreteTimeIntegrator_DSTATE_k += 5.0E-6 * rtb_Sum_n;
1085
1086 /* Update for S-Function (sfun_spssw_disce): '<S232>/State-Space'
    incorporates:
1087 * Constant: '<S208>/DC'
1088 */
1089 {
1090 int_T *gState = (int_T*)rtDW.StateSpace_PWORK.G_STATE;

```

```

1091
1092  /* Store switch gates values for next step */
1093  {
1094    int_T i1;
1095    const real_T *u1 = &rtDW.DataTypeConversion[0];
1096    for (i1=0; i1 < 6; i1++) {
1097      *(gState++) = (int_T) u1[i1];
1098    }
1099  }
1100 }
1101
1102 /* Update for DiscreteIntegrator: '<S223>/Discrete-Time Integrator' */
1103 rtDW.DiscreteTimeIntegrator_DSTATE_p += 5.0E-6 * rtb_Gain2_n;
1104 }
1105 } /* end MajorTimeStep */
1106
1107 if (rtmIsMajorTimeStep(rtm)) {
1108   rt_ertODEUpdateContinuousStates(&rtM->solverInfo);
1109
1110   /* Update absolute time for base rate */
1111   /* The "clockTick0" counts the number of times the code of this task has
1112   * been executed. The absolute time is the multiplication of "clockTick0"
1113   * and "Timing.stepSize0". Size of "clockTick0" ensures timer will not
1114   * overflow during the application lifespan selected.
1115   * Timer of this task consists of two 32 bit unsigned integers.
1116   * The two integers represent the low bits Timing.clockTick0 and the high
1117   * bits Timing.clockTickH0. When the low bit overflows to 0, the high bits
1118   * increment.
1119   */
1120   if (!(++rtM->Timing.clockTick0)) {
1121     ++rtM->Timing.clockTickH0;
1122   }
1123   rtM->Timing.t[0] = rtsiGetSolverStopTime(&rtM->solverInfo);
1124
1125   {
1126     /* Update absolute timer for sample time: [5.0E-6s, 0.0s] */
1127     /* The "clockTick1" counts the number of times the code of this task has

```

```

1128  * been executed. The resolution of this integer timer is 5.0E-6, which is the
1129  * of the task. Size of "clockTick1" ensures timer will not overflow during
1130  * application lifespan selected.
1131  * Timer of this task consists of two 32 bit unsigned integers.
1132  * The two integers represent the low bits Timing.clockTick1 and the high
1133  * bits Timing.clockTickH1. When the low bit overflows to 0, the high bits
1134  * increment.
1135  */
1136  rtM->Timing.clockTick1++;
1137  if (!rtM->Timing.clockTick1) {
1138      rtM->Timing.clockTickH1++;
1139  }
1140  }
1141  rate_scheduler();
1142  } /* end MajorTimeStep */
1143  }
1144
1145  /* Derivatives for root system: '<Root>' */
1146  void foclookuptabletanpabeban_derivatives(void)
1147  {
1148      XDot * _rtXdot;
1149      _rtXdot = ((XDot *) rtM->derivs);
1150
1151  /* Derivatives for Integrator: '<S65>/Integrator' */
1152  _rtXdot->Integrator_CSTATE[0] = rtDW.IntegralGain[0];
1153
1154  /* Derivatives for Integrator: '<S45>/Filter' */
1155  _rtXdot->Filter_CSTATE[0] = rtDW.FilterCoefficient[0];
1156
1157  /* Derivatives for Integrator: '<S65>/Integrator' */
1158  _rtXdot->Integrator_CSTATE[1] = rtDW.IntegralGain[1];
1159
1160  /* Derivatives for Integrator: '<S45>/Filter' */
1161  _rtXdot->Filter_CSTATE[1] = rtDW.FilterCoefficient[1];
1162

```

```

1163  /* Derivatives for Integrator: '<S161>/Integrator' */
1164  _rtXdot->Integrator_CSTATE_j = rtDW.IntegralGain_a;
1165
1166  /* Derivatives for Integrator: '<S141>/Filter' */
1167  _rtXdot->Filter_CSTATE_n = rtDW.FilterCoefficient_k;
1168
1169  /* Derivatives for Integrator: '<S220>/Integrator' */
1170  _rtXdot->Integrator_CSTATE_a = rtDW.Sum3;
1171  }
1172
1173  /* Model initialize function */
1174  void foclookuptabletanpabeban_initialize(void)
1175  {
1176  /* Registration code */
1177
1178  /* initialize non-finites */
1179  rt_InitInfAndNaN(sizeof(real_T));
1180
1181  {
1182  /* Setup solver object */
1183  rtsiSetSimTimeStepPtr(&rtM->solverInfo, &rtM->Timing.simTimeStep);
1184  rtsiSetTPtr(&rtM->solverInfo, &rtmGetTPtr(rtM));
1185  rtsiSetStepSizePtr(&rtM->solverInfo, &rtM->Timing.stepSize0);
1186  rtsiSetdXPtr(&rtM->solverInfo, &rtM->derivs);
1187  rtsiSetContStatesPtr(&rtM->solverInfo, (real_T **) &rtM->contStates);
1188  rtsiSetNumContStatesPtr(&rtM->solverInfo, &rtM->
    >Sizes.numContStates);
1189  rtsiSetNumPeriodicContStatesPtr(&rtM->solverInfo,
1190  &rtM->Sizes.numPeriodicContStates);
1191  rtsiSetPeriodicContStateIndicesPtr(&rtM->solverInfo,
1192  &rtM->periodicContStateIndices);
1193  rtsiSetPeriodicContStateRangesPtr(&rtM->solverInfo,
1194  &rtM->periodicContStateRanges);
1195  rtsiSetErrorStatusPtr(&rtM->solverInfo, (&rtmGetErrorStatus(rtM)));
1196  rtsiSetRTModelPtr(&rtM->solverInfo, rtM);
1197  }
1198
1199  rtsiSetSimTimeStep(&rtM->solverInfo, MAJOR_TIME_STEP);
1200  rtM->intgData.y = rtM->odeY;

```



```

1201 rtM->intgData.f[0] = rtM->odeF[0];
1202 rtM->intgData.f[1] = rtM->odeF[1];
1203 rtM->intgData.f[2] = rtM->odeF[2];
1204 rtM->contStates = ((X *) &rtX);
1205 rtsiSetSolverData(&rtM->solverInfo, (void *)&rtM->intgData);
1206 rtsiSetSolverName(&rtM->solverInfo,"ode3");
1207 rtmSetTPtr(rtm, &rtM->Timing.tArray[0]);
1208 rtM->Timing.stepSize0 = 5.0E-6;
1209
1210 /* Start for S-Function (sfun_spssw_discc): '<S232>/State-Space'
    incorporates:
1211 * Constant: '<S208>/DC'
1212 */
1213
1214 /* S-Function block: <S232>/State-Space */
1215 {
1216 rtDW.StateSpace_PWORK.DS = (real_T*)calloc(11 * 9, sizeof(real_T));
1217 rtDW.StateSpace_PWORK.DX_COL = (real_T*)calloc(11,
    sizeof(real_T));
1218 rtDW.StateSpace_PWORK.TMP2 = (real_T*)calloc(9, sizeof(real_T));
1219 rtDW.StateSpace_PWORK.SWITCH_STATUS = (int_T*)calloc(6,
    sizeof(int_T));
1220 rtDW.StateSpace_PWORK.SW_CHG = (int_T*)calloc(6, sizeof(int_T));
1221 rtDW.StateSpace_PWORK.G_STATE = (int_T*)calloc(6, sizeof(int_T));
1222 rtDW.StateSpace_PWORK.Y_SWITCH = (real_T*)calloc(6,
    sizeof(real_T));
1223 rtDW.StateSpace_PWORK.SWITCH_TYPES = (int_T*)calloc(6,
    sizeof(int_T));
1224 rtDW.StateSpace_PWORK.IDX_OUT_SW = (int_T*)calloc(6,
    sizeof(int_T));
1225 rtDW.StateSpace_PWORK.SWITCH_STATUS_INIT = (int_T*)calloc(6,
    sizeof(int_T));
1226 rtDW.StateSpace_PWORK.USWLAST = (real_T*)calloc(6,
    sizeof(real_T));
1227 }
1228
1229 /* Start for FromWorkspace: '<S9>/FromWs' */
1230 {
1231 static real_T pTimeValues0[] = { 0.0, 0.3, 0.3, 1.0, 1.3, 1.5, 10.0 } ;
1232

```

```

1233 static real_T pDataValues0[] = { 0.0, 100.0, 100.0, 100.0, 100.0, 100.0,
1234 100.0 } ;
1235
1236 rtDW.FromWs_PWORK.TimePtr = (void *) pTimeValues0;
1237 rtDW.FromWs_PWORK.DataPtr = (void *) pDataValues0;
1238 rtDW.FromWs_IWORK.PrevIndex = 0;
1239 }
1240
1241 /* InitializeConditions for S-Function (sfun_spssw_discc): '<S232>/State-
1242   Space' incorporates:
1243   * Constant: '<S208>/DC'
1244   */
1244 {
1245   int32_T i, j;
1246   real_T *Ds = (real_T*)rtDW.StateSpace_PWORK.DS;
1247
1248   /* Copy and transpose D */
1249   for (i=0; i<11; i++) {
1250     for (j=0; j<9; j++)
1251       Ds[i*9 + j] = (rtConstP.StateSpace_DS_param[i + j*11]);
1252   }
1253
1254   {
1255     /* Switches work vectors */
1256     int_T *switch_status = (int_T*)
1257       rtDW.StateSpace_PWORK.SWITCH_STATUS;
1257     int_T *gState = (int_T*)rtDW.StateSpace_PWORK.G_STATE;
1258     real_T *yswitch = (real_T*)rtDW.StateSpace_PWORK.Y_SWITCH;
1259     int_T *switchTypes =
1260       (int_T*)rtDW.StateSpace_PWORK.SWITCH_TYPES;
1260     int_T *idxOutSw = (int_T*)rtDW.StateSpace_PWORK.IDX_OUT_SW;
1261     int_T *switch_status_init = (int_T*)
1262       rtDW.StateSpace_PWORK.SWITCH_STATUS_INIT;
1263
1264     /* Initialize work vectors */
1265     switch_status[0] = 0;
1266     switch_status_init[0] = 0;
1267     gState[0] = (int_T) 0.0;
1268     yswitch[0] = 1/0.001;

```

```

1269 switchTypes[0] = (int_T)7.0;
1270 idxOutSw[0] = ((int_T)0.0) - 1;
1271 switch_status[1] = 0;
1272 switch_status_init[1] = 0;
1273 gState[1] = (int_T) 0.0;
1274 yswitch[1] = 1/0.001;
1275 switchTypes[1] = (int_T)7.0;
1276 idxOutSw[1] = ((int_T)0.0) - 1;
1277 switch_status[2] = 0;
1278 switch_status_init[2] = 0;
1279 gState[2] = (int_T) 0.0;
1280 yswitch[2] = 1/0.001;
1281 switchTypes[2] = (int_T)7.0;
1282 idxOutSw[2] = ((int_T)0.0) - 1;
1283 switch_status[3] = 0;
1284 switch_status_init[3] = 0;
1285 gState[3] = (int_T) 0.0;
1286 yswitch[3] = 1/0.001;
1287 switchTypes[3] = (int_T)7.0;
1288 idxOutSw[3] = ((int_T)0.0) - 1;
1289 switch_status[4] = 0;
1290 switch_status_init[4] = 0;
1291 gState[4] = (int_T) 0.0;
1292 yswitch[4] = 1/0.001;
1293 switchTypes[4] = (int_T)7.0;
1294 idxOutSw[4] = ((int_T)0.0) - 1;
1295 switch_status[5] = 0;
1296 switch_status_init[5] = 0;
1297 gState[5] = (int_T) 0.0;
1298 yswitch[5] = 1/0.001;
1299 switchTypes[5] = (int_T)7.0;
1300 idxOutSw[5] = ((int_T)0.0) - 1;
1301 }
1302 }
1303
1304 /* InitializeConditions for Integrator: '<S65>/Integrator' */
1305 rtX.Integrator_CSTATE[0] = 0.0;
1306
1307 /* InitializeConditions for Integrator: '<S45>/Filter' */

```

```

1308 rtX.Filter_CSTATE[0] = 0.0;
1309
1310 /* InitializeConditions for Integrator: '<S65>/Integrator' */
1311 rtX.Integrator_CSTATE[1] = 0.0;
1312
1313 /* InitializeConditions for Integrator: '<S45>/Filter' */
1314 rtX.Filter_CSTATE[1] = 0.0;
1315
1316 /* InitializeConditions for Integrator: '<S161>/Integrator' */
1317 rtX.Integrator_CSTATE_j = 0.0;
1318
1319 /* InitializeConditions for Integrator: '<S141>/Filter' */
1320 rtX.Filter_CSTATE_n = 0.0;
1321
1322 /* InitializeConditions for Integrator: '<S220>/Integrator' */
1323 rtX.Integrator_CSTATE_a = 0.0;
1324 }
1325
1326 /*
1327  * File trailer for generated code.
1328  *
1329  * [EOF]
1330  */
1331

```

BIODATA PENULIS



Agus Nurtriartono dilahirkan di Balikpapan, 14 Agustus 1991. Anak ketiga dari Djamil dan Sutini. Penulis menyelesaikan masa studi sekolah dasar di SDN 008 Balikpapan pada tahun 2003, dilanjutkan ke SMP YASPORBI 2 Jakarta pada tahun 2006 dan SMAN 28 Jakarta lulus pada tahun 2009.

Selepas SMA penulis melanjutkan studinya di Institut Teknologi Sepuluh Nopember Jurusan Teknik Mesin pada tahun ajaran 2009/2010. Selama kuliah di ITS penulis aktif mengikuti organisasi Lembaga Bengkel Mahasiswa Mesin. Tidak hanya itu penulis juga telah bergabung dalam tim mobil surya ITS yang telah berkompetisi di Australia pada *World Solar Challenge 2013* serta turut aktif dalam tim Molina (Mobil Listrik Nasional) ITS. Di teknik mesin penulis memilih untuk masuk Laboratorium Otomasi dan mengerjakan tugas akhir dengan topic Rancang Bangun dan Uji Performa motor listrik dibawah bimbingan Dr. Muhammad Nur Yuniarto. Pada tahun 2015 penulis menyelesaikan studi S1-nya. Pada tahun 2018 penulis melanjutkan study S2-nya di Institut Teknologi Sepuluh Nopember Jurusan Teknik Mesin dibawah bimbingan Dr. Muhammad Nur Yuniarto. Pada 2020 penulis menyelesaikan study S2-nya.