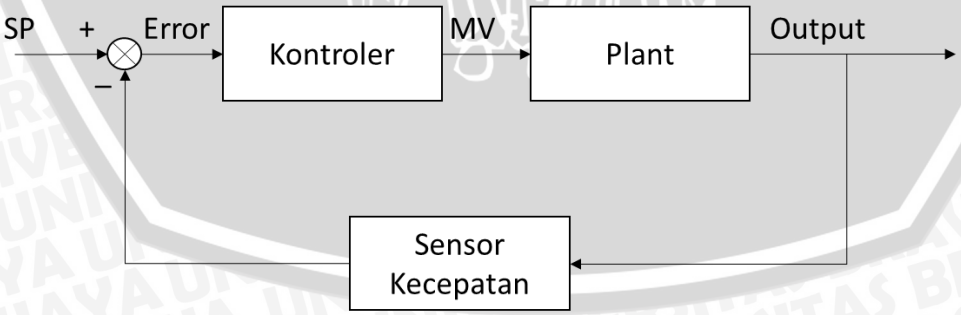
BAB III
METODE PENELITIAN

Perancangan sistem kontrol kecepatan sepeda listrik menggunakan metode *self-tuning* parameter PI dengan metode logika *fuzzy* dilakukan secara bertahap sehingga akan mempermudah dalam menganalisis setiap blok atau keseluruhan sistem. Pada dasarnya rancangan sistem tersebut meliputi rancangan perangkat keras atau *hardware* dan algoritma perangkat lunak atau *software*.

3.1. Diagram Blok Sistem

Dalam perancangan alat diperlukan perancanaan blok diagram sistem yang dapat menjelaskan sistem secara garis besar dan diharapkan alat dapat bekerja sesuai dengan rencana.

Setpoint (SP) adalah *input* berupa nilai kecepatan yang diberikan oleh pengguna. Nilai *setpoint* akan disimpan ke dalam mikrokontroler. *Error* adalah deviasi atau simpangan antara pembacaan aktual kecepatan motor dari *rotary encoder* dan nilai *setpoint*. Kedua nilai tersebut akan dikalkulasi dan diolah oleh mikrokontroler. Nilai yang dikalkulasi adalah *manipulated variable* (MV) atau sinyal kontrol yang akan digunakan sebagai *input* pada *plant*, sehingga *plant* yang berupa sepeda listrik akan bekerja sesuai dengan *setpoint* yang telah ditentukan.



Gambar 3.1 Diagram Blok Sistem *Loop* Tertutup

3.2. Spesifikasi Desain

Desain yang diinginkan pada perancangan sistem kontrol kecepatan sepeda listrik menggunakan metode *self-tuning* parameter PI dengan metode logika *fuzzy* mempunyai spesifikasi yaitu:

1. *Error steady-state* < 5%

   Error steady-state < 5%, karena sistem yang baik memiliki *output* dengan batas nilai akhir 5% dari *setpoint*.

2. Rasio redaman $0 < \xi < 1$

   Rasio redaman ditentukan $0 < \xi < 1$, agar *output* melesat mencapai nilai *setpoint* kemudian turun dan stabil pada nilai *setpoint*.

3. *Settling time* < 15 detik

   Settling time < 15 detik, karena diharapkan sistem kontrol kecepatan sepeda listrik menggunakan metode *self-tuning* parameter PI dengan metode logika *fuzzy* mampu mempercepat *settling time* sistem kurang dari 15 detik.

4. *Maximum (Percent) Overshoot* < 10%

   Sistem memiliki *maximum (percent) overshoot* kurang dari 10%.

5. Mampu mengatasi beban tanjakan sampai dengan 30º

   Sistem diharapkan mampu mengatasi beban berupa jalan tanjakan dengan kemiringan bidang sebesar 30º, seperti kendaraan ekivalen dari sepeda listrik, yaitu sepeda dengan penggerak motor bakar.

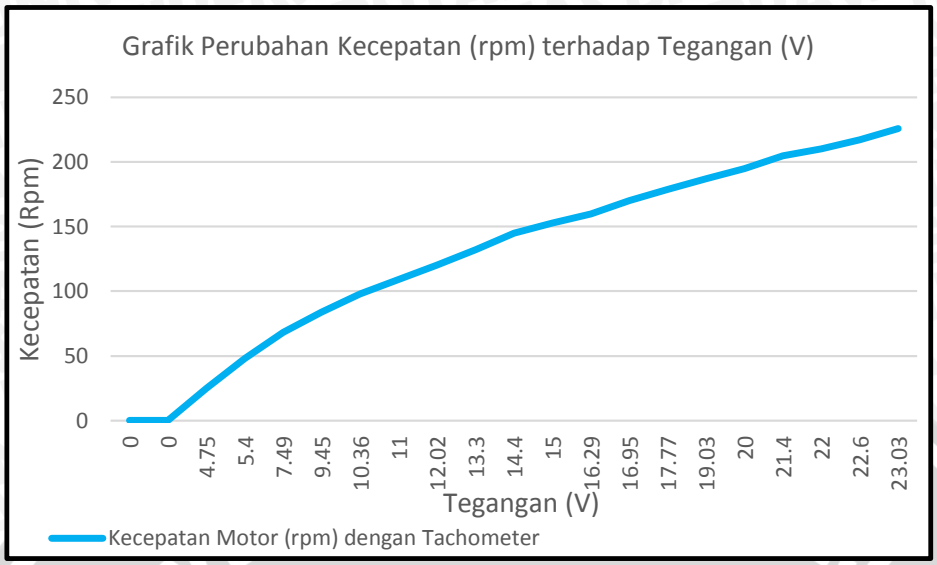
3.3. Karakterisasi Motor *Brushless* DC (BLDC)

Untuk mengetahui karakteristik kecepatan motor BLDC pada setiap kenaikan tegangan *input* motor, dilakukan pengujian karakterisasi kecepatan motor BLDC. Pengujian dilakukan dengan cara membandingkan tegangan *input* motor terhadap kecepatan motor, menggunakan tachometer digital sebagai pembaca kecepatan motor, dan voltmeter yang dipasang pada *input* motor BLDC sebagai pembaca tegangan *input* motor. Langkah – langkah dalam pengujian karakterisasi motor BLDC yaitu: menghubungkan *output* tegangan baterai dengan *input driver* motor tiga fasa, lalu mengatur *duty cycle* dari 0% sampai dengan 100% dengan kenaikan 5% setiap pembacaan tegangan *input* dan kecepatan motor, kemudian hasil dari pengujian tersebut dicatat dalam Tabel 3.1

Tabel 3.1 Data pengujian kecepatan motor BLDC (rpm) terhadap tegangan (V)

Duty Cycle (%)	*Tegangan (V)*	*Kecepatan Motor (RPM) dengan Tachometer*
0	0	0
5	0	0
10	4,75	25
15	6,3	48
20	7,49	68
25	9,03	84
30	10,36	98
35	11,5	109
40	12,5	120
45	13,3	132
50	14,4	145
55	15	153
60	16,29	160
65	16,95	170
70	17,77	179
75	19,03	187
80	20	195
85	21	205
90	21,79	210
95	22,6	217
100	23,03	226

Berdasarkan Tabel 3.1 dapat diketahui bahwa pada *duty cycle* 0% dan 5%, motor BLDC tidak berputar dan tegangan yang terukur sama dengan 0 V. Dengan mengambil data tegangan *input* motor terhadap kecepatan motor yang terukur, didapatkan kurva kecepatan motor (rpm) terhadap *input* tegangan (V) seperti ditunjukan pada Gambar 3.2.



Gambar 3.2 Grafik perubahan kecepatan motor BLDC(rpm) terhadap tegangan(v)

3.4. Karakterisasi *Driver* Motor Tiga Fasa

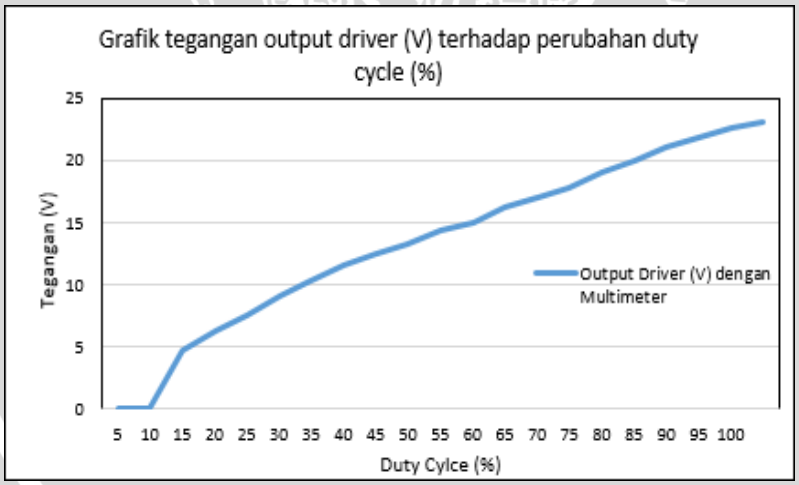
Untuk mengetahui karakteristik, kinerja, dan *output* rangkaian *driver* motor tiga fasa, dilakukan pengujian karakterisasi *driver* motor tiga fasa. Pengujian dilakukan dengan cara membandingan *input duty cycle* sinyal PWM yang diberikan oleh kontroler Arduino Mega 2560 terhadap tegangan efektif *output driver*, menggunakan voltmeter yang dipasang pada *output* rangkaian *driver* sebagai pembaca tegangan efektif *output driver* motor tiga fasa. Langkah – langkah dalam pengujian karakterisasi *driver* motor tiga fasa yaitu: menghubungkan *output* tegangan baterai dengan *input driver* motor tiga fasa, lalu menghubungkan *input* sinyal kontrol pada *driver* motor tiga fasa dengan pin *output* PWM di Arduino Mega 2560, kemudian mengatur *duty cycle* dari 0% sampai dengan 100% dengan kenaikan 5% setiap pembacaan tegangan efektif *output driver*. Hasil pembacaan yang didapatkan dari pengujian tersebut dicatat dalam Tabel 3.2

Tabel 3.2 Data pengujian *driver* motor tiga fasa.

Duty Cycle %	*Output Driver* (V) dengan Multimeter
0	0
5	0
10	4,75
15	6,3
20	7,49
25	9,03
30	10,36

Duty Cycle %	*Output Driver* (V) dengan Multimeter
35	11,5
40	12,5
45	13,3
50	14,4
55	15
60	16,29
65	16,95
70	17,77
75	19,03
80	20
85	21
90	21,79
95	22,6
100	23,03

Berdasarkan Tabel 3.2 dapat diketahui bahwa pada *duty cycle* 0% dan 5% tegangan efektif *output driver* sama dengan 0 V. Dengan mengambil data *duty cycle* terhadap tegangan efektif *output driver* motor tiga fasa yang terukur, didapatkan kurva tegangan efektif *output driver* (V) terhadap *input duty cycle* sinyal PWM (%) seperti ditunjukan pada Gambar 3.3.



Gambar 3.3 Grafik perubahan tegangan *output driver* terhadap *duty cycle*

3.5. Karakterisasi *Rotary encoder*

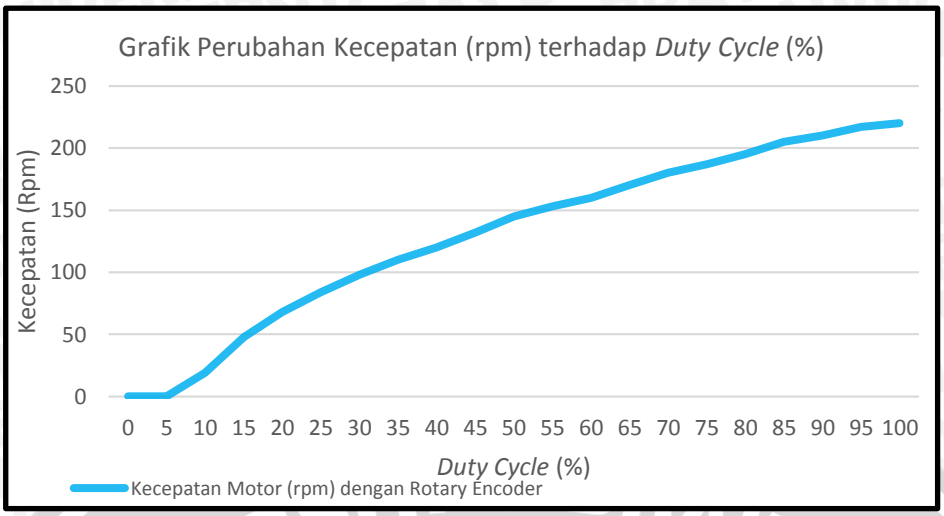
Untuk mengetahui tingkat kelinieran dari *rotary encoder* dalam pembacaan kecepatan motor BLDC, dilakukan pengujian karakterisasi *rotary encoder*. Pengujian dilakukan dengan cara membandingkan *input duty cycle* sinyal PWM yang diberikan oleh kontroler Arduino Mega 2560 terhadap kecepatan motor BLDC yang dibaca oleh *rotary encoder*. Langkah – langkah dalam pengujian karakterisasi *rotary encoder* yaitu: menghubungkan

output tegangan baterai dengan *input driver* motor tiga fasa, lalu menghubungkan pin ouput *rotary encoder* dengan pin interrupt eksternal, dan *input* sinyal kontrol pada *driver* motor tiga fasa dengan pin *output* PWM pada Arduino Mega 2560, kemudian mengatur *duty cycle* dari 0% sampai dengan 100% dengan kenaikan 5% setiap pembacaan kecepatan motor. Hasil pembacaan yang didapatkan dari pengujian tersebut dicatat dalam Tabel 3.3

Tabel 3.3 Data pengujian *Rotary encoder*

***Duty Cycle* %**	**Kecepatan Motor (rpm) dengan *Rotary encoder***
0	0
5	0
10	19
15	48
20	68
25	84
30	98
35	110
40	120
45	132
50	145
55	153
60	160
65	170
70	180
75	187
80	195
85	205
90	210
95	217
100	220

Berdasarkan Tabel 3.3 dapat diketahui bahwa pada *duty cycle* 0% dan 5% kecepatan motor sama dengan 0 rpm. Dengan mengambil data *duty cycle* terhadap kecepatan motor BLDC yang dibaca oleh *rotary encoder*, akan didapatkan kurva kecepatan motor yang dibaca oleh *rotary encoder* terhadap *input duty cycle* sinyal PWM (%) seperti ditunjukkan pada Gambar 3.4.

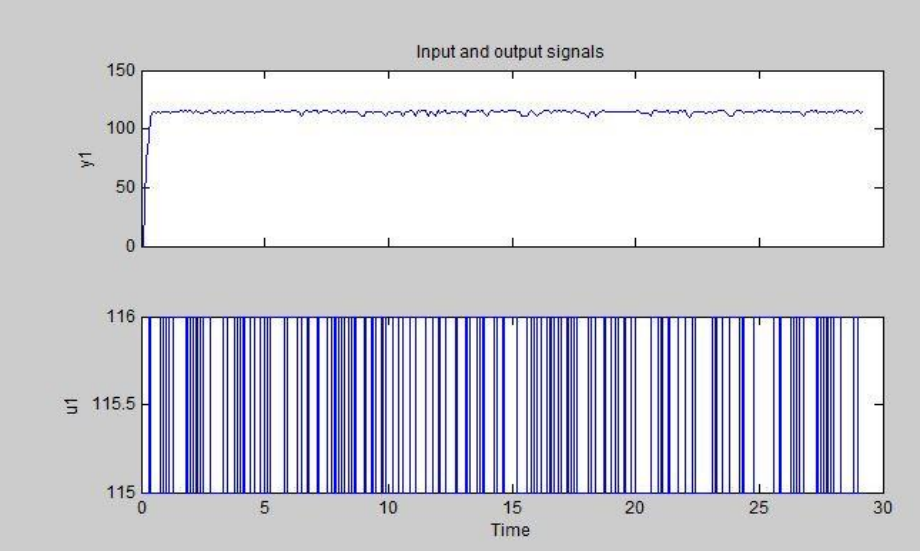


Gambar 3.4 Grafik perubahan *output* kecepatan motor BLDC terhadap *duty cycle*

3.6. Penentuan Fungsi Alih Motor *Brushless* DC (BLDC)

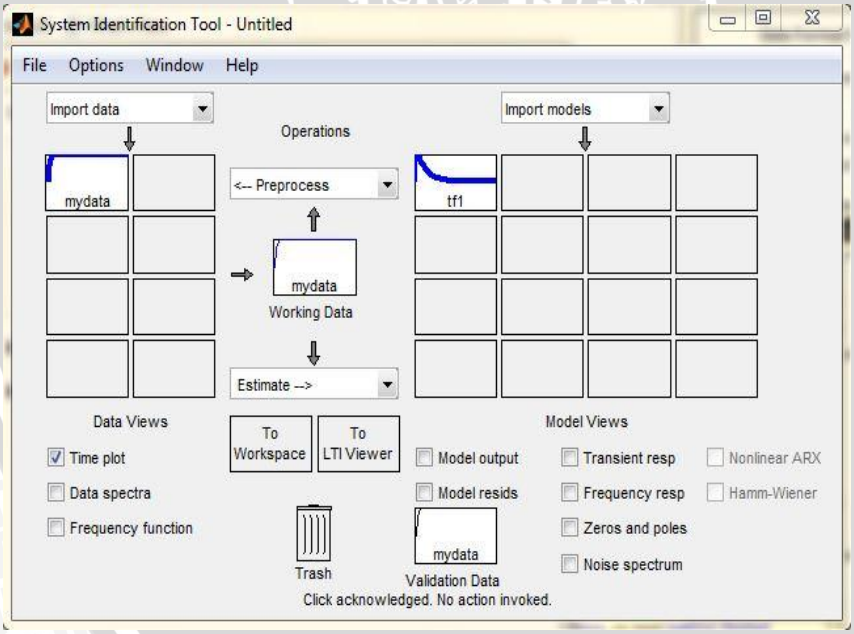
Untuk mengontrol kecepatan motor BLDC digunakan Arduino Mega 2560 yang berfungsi sebagai kontroler untuk mengolah sinyal *error*, dan memberikan sinyal kontrol berupa *Pulse Width Modulation* (PWM) agar motor dapat bekerja sesuai spesifikasi yang telah ditentukan. Agar kontroler PI dapat bekerja secara optimal, diperlukan fungsi alih untuk mendapatkan parameter PI. Motor BLDC yang digunakan pada perancangan ini tidak diketahui spesifikasi teknisnya, sehingga tidak memungkinkan untuk mendapatkan fungsi alih dari perhitungan. Oleh karena itu, fungsi alih didapatkan dengan melakukan pengujian menggunakan *rotary encoder* untuk membaca kecepatan motor dan membangkitkan sinyal *Pseudo Random Binary Sequence* (PRBS) sebagai *input*. Langkah yang dilakukan untuk menentukan fungsi alih dengan cara membangkitkan sinyal PRBS adalah sebagai berikut:

1. Mencari nilai yang linier dari hasil kecepatan motor terhadap *duty cycle* PWM.
2. Memasukkan nilai batas atas dan bawah berdasarkan nilai yang linier untuk membangkitkan sinyal PRBS.
3. Sinyal PRBS yang telah dibangkitkan kemudian digunakan sebagai *input* motor BLDC.
4. Setelah didapatkan data sinyal PRBS dan data kecepatan motor BLDC dalam Gambar 3.5, selanjutnya adalah melakukan identifikasi dengan menggunakan *software* MATLAB



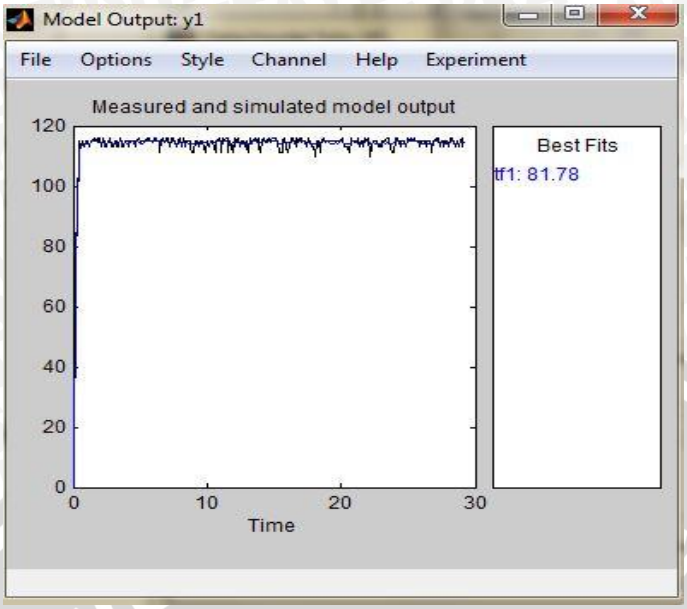
Gambar 3.5 *Output* sinyal PRBS

5. Dengan menggunakan sintaks ident pada command window pada MATLAB, data sinyal PRBS dan data kecepatan motor yang telah disimpan kemudian di-*import* pada blok *System Identification Toolbox*. Menu *System Identification Toolbox dapat* dilihat dalam Gambar 3.6.



Gambar 3.6 Tampilan aplikasi ident di software MATLAB

6. Setelah melakukan beberapa identifikasi berdasarkan data yang telah di-*import* didapatkan fungsi alih motor dengan *best fits* sebesar 81,78. Hasil simulasi *output* dapat dilihat dalam Gambar 3.7.



Gambar 3.7 Hasil simulasi *output*

7. Dari hasil identifikasi, fungsi alih yang didapat adalah

$$\frac{Y(s)}{U(s)} = G(S) = \frac{1182}{s^2+125.3s+1985} \quad (3\text{-}1)$$

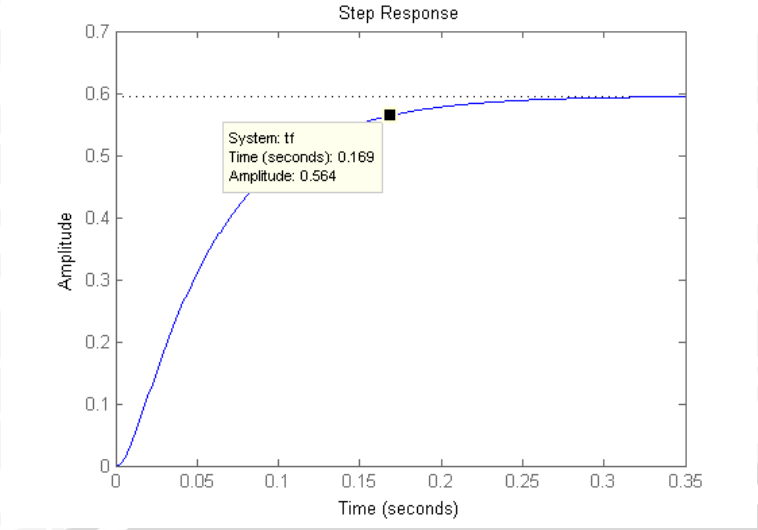
$$G(s) = \frac{{\omega_n}^2}{s^2+2\xi\omega_n s+{\omega_n}^2} \quad (3\text{-}2)$$

$$2\xi\omega_n = 125.3 \quad (3\text{-}3)$$

$$\omega_n = \sqrt{1985} = 44.553 \quad (3\text{-}4)$$

$$\xi = \frac{125.3}{2 \text{ x } 44.553} = 1.4061 \quad (3\text{-}5)$$

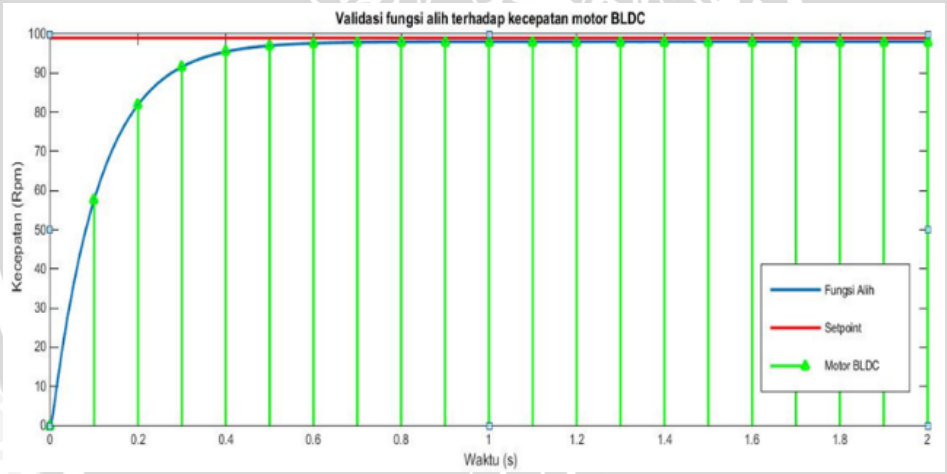
8. Dengan memberikan *input unit step* pada program MATLAB didapatkan *output* dalam Gambar 3.8, dan didapatkan nilai *settling time* 0,169 s dengan *output* maksimal berada di 0,594



Gambar 3.8 *output* fungsi alih motor BLDC dengan *input* unit step

3.7. Validasi Fungsi Alih Motor *Brushless* DC (BLDC)

Validasi fungsi alih motor dilakukan dengan cara membandingkan *output* motor BLDC yang didapatkan dari identifikasi dan *output* kecepatan motor BLDC yang didapatkan dari pembacaan *rotary encoder* dengan memberikan *input* pulsa *unit step*. Perbandingan kedua *output* yang didapat dengan menggunakan MATLAB dapat dilihat dalam Gambar 3.9.



Gambar 3.9 Validasi *output* sistem yang didapatkan dari identifikasi dengan *output* kecepatan motor BLDC

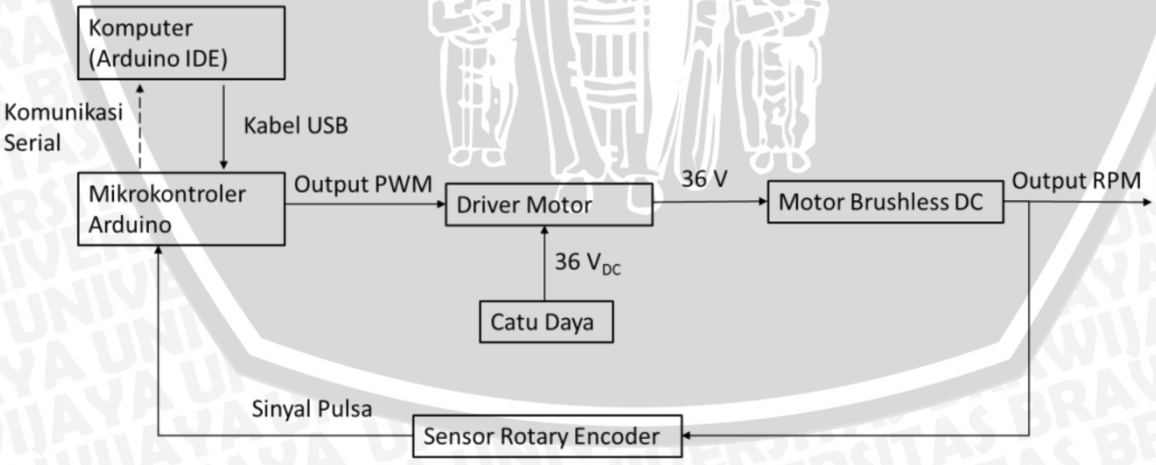
Dari Gambar 3.9 dapat dilihat bahwa *output* sistem yang telah didapat dari proses identifikasi hampir menyerupai *output* kecepatan motor BLDC. fungsi alih yang telah didapatkan dianggap dapat mewakili sistem.

3.8. Perancangan Perangkat Keras

Sebelum merancang algoritma kontrol, perangkat keras terelebih dahulu dirancang dengan mempertimbangkan kajian pustaka yang telah ada. Hal tersebut bertujuan agar sistem yang telah direncanakan dapat bekerja sebagaimana mestinya. Pembuatan perangkat keras meliputi:

1. Skema pembuatan perangkat keras

Baterai 36 Volt sebagai supply motor BLDC dan rangkaian *driver* motor tiga fasa. Kontroler berupa mikrokontroler Arduino Mega 2560 yang di-*program* melalui *port* USB yang dihubungkan pada komputer. Kontroler menerima sinyal pulsa dari *rotary encoder* untuk diproses menjadi *output* yang terbaca, dan selanjutnya mengurangkan *output* terhadap *setpoint* untuk mendapatkan *error* yang nantinya akan diproses menjadi sinyal kontrol atau *Manipulated Variable* (*MV*). Mikrokontroler juga mengirimkan data parameter sistem melalui komunikasi serial pada komputer. Sinyal kontrol dari Mikrokontroler Arduino Mega 2560 masuk ke *driver* motor tiga fasa. Tegangan *output driver* motor tiga fasa masuk ke dalam Motor BLDC dan memutar motor tersebut. Motor BLDC yang berputar telah dikopel dengan *rotary encoder* sebagai pembaca kecepatan motor BLDC. *Output rotary encoder* berupa pulsa dengan tegangan 0 V sampai 5 V sebagai *input* digital pada pin *external interrupt* Arduino Mega 2560.



Gambar 3.10 Skema perangkat keras

2. Penentuan modul elektronik yang digunakan

- Komputer atau PC yang didalamnya telah ter-*install software* MATLAB dan Arduino IDE, dan memiliki *port* USB untuk pemrograman mikrokontroler. Contoh komputer atau PC dapat dilihat dalam Gambar 3.11

Gambar 3.11 Komputer atau PC

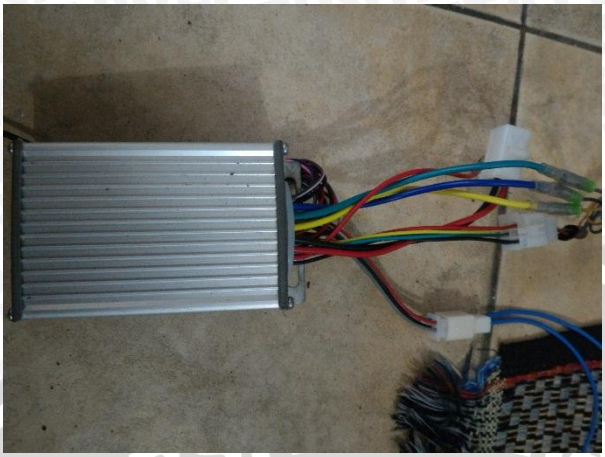
- Baterai 12 V 7.2 Ah sebanyak 3 buah yang disusun secara seri sebagai catu daya 36 V, dapat dilihat dalam Gambar 3.12

Gambar 3.12 Baterai

- Mikrokontroler Arduino Mega 2560 sebagai kontroler seperti dalam Gambar 3.13

Gambar 3.13 Mikrokontroler Arduino Mega 2560

- *Driver* motor tiga fasa seperti dalam Gambar 3.14



Gambar 3.14 *Driver* Motor Tiga Fasa

- Motor *Brushless* DC (BLDC) 350 watt, seperti dalam Gambar 3.15

Gambar 3.15 Motor Bruhsless DC

- *Rotary encoder*, dengan pin VCC terhubung dengan catu +5 V mikrokontroler Arduino Mega 2560, GND terhubung dengan *ground* mikrokontroler, dan OUT terhubung dengan *external interrupt* 0 mikrokontroler. Contoh modul sensor *rotary encoder* dapat dilihat dalam Gambar 3.16.

Gambar 3.16 *Rotary encoder*

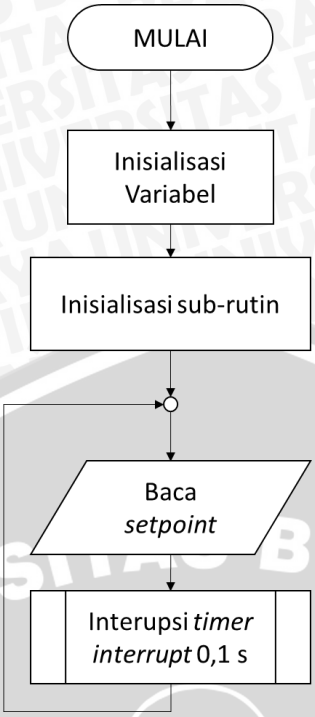
3. Perancangan algoritma sistem kontrol kecepatan sepeda listrik menggunakan metode *self-tuning* parameter PI dengan metode logika *fuzzy* meliputi tahap – tahap berikut:
    1. Identifikasi fungsi alih Motor BLDC.
    2. Pembuatan diagram alir program.
    3. Pembuatan algoritma kontrol logika *fuzzy* penala parameter PI.
    4. Pembuatan algoritma kontroler PI dan penalaan parameter PI.
    5. Pembobotan parameter PI terhadap kontrol logika *fuzzy* dan kontrol PI
    6. Verifikasi sistem meliputi *output*, *error steady-state*, *settling time*, dan perubahan parameter PI.

Sistem yang telah dirangkai dapat dilihat dalam Gambar 3.17

Gambar 3.17 Sistem yang telah dirangkai

3.9. Diagram Alir Program Utama

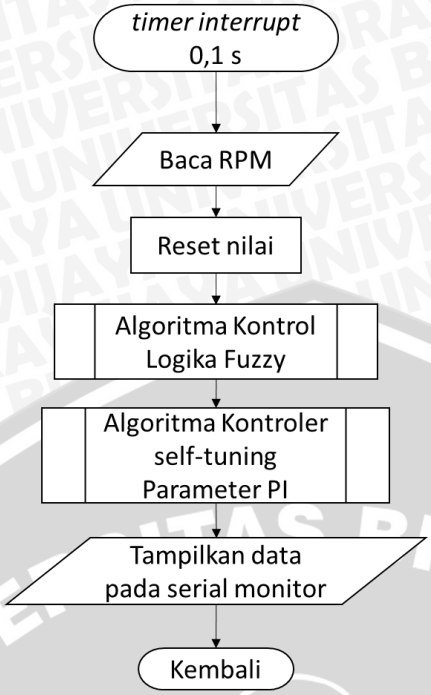
Program diawali dengan inisialisasi masing – masing variabel dan sub-rutin untuk proses kontrol, dan dilanjutkan dengan membaca variabel *input* yang berguna sebagai penentu *setpoint*. Pada program utama terdapat sub-rutin *timer interrupt* yang dieksekusi setiap 0,1 detik. Diagram alir dari program utama dapat dilihat dalam Gambar 3.18.



Gambar 3.18 Diagram alir program utama

3.9.1. Diagram Alir Sub-Rutin *Timer Interrupt*

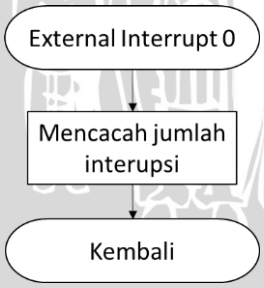
Program dieksekusi setiap 0,1 detik sekali, dan diawali dengan pembacaan *output* kecepatan yang dikurangkan terhadap setpoint yang telah ada. Hasil dari perhitungan tersebut berupa *error*. Kemudian, dilakukan proses reset nilai perhitungan *fuzzy*, agar tidak ada sisa angka dari siklus sebelumnya yang ikut dalam perhitungan siklus sekarang. *Error* yang telah didapat digunakan sebagai *input* untuk sub-rutin algoritma kontrol logika *Fuzzy* dan kontroler PI, dengan prioritas kontrol logika *Fuzzy* terlebih dahulu. Hasil perhitungan dari algoritma kontrol logika *Fuzzy* digunakan sebagai penala parameter PI pada kontroler PI. Nilai kontroler PI dikeluarkan sebagai *manipulated variable* yang digunakan untuk mengatur kecepatan motor. Diagram alir dari sub-rutin *timer interrupt* dapat dilihat dalam gambar 3.19.



Gambar 3.19 Diagram alir sub-rutin *timer interrupt* 0,1 detik

3.9.2. Diagram Alir Sub-Rutin *External interrupt*

Program dieksekusi ketika pin *external interrupt* 0 membaca tepi turun dari sinyal pulsa yang dikirim oleh *rotary encoder*, dan berfungsi untuk mencacah jumlah interupsi yang terjadi untuk menentukan kecepatan motor BLDC. Diagram alir dari sub-rutin *external interrupt* dapat dilihat dalam gambar 3.20.



Gambar 3.20 Diagram alir sub-rutin *external interrupt*

3.10. Perancangan Kontrol Logika *Fuzzy*

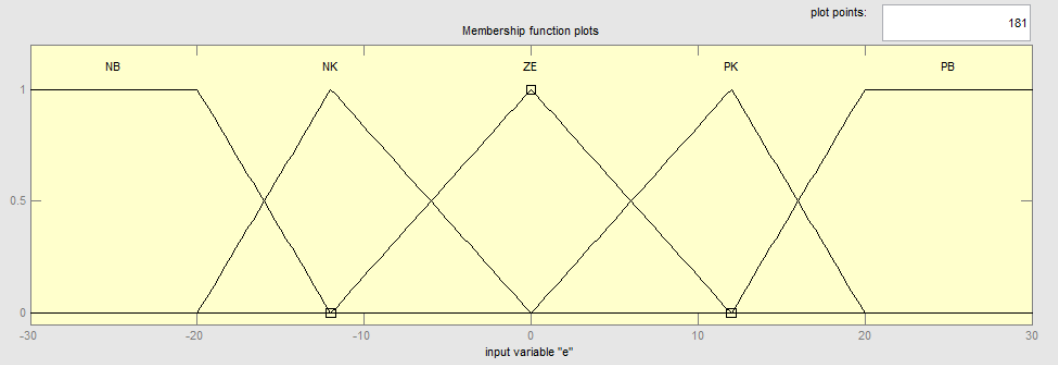
3.10.1. Fungsi Keanggotaan

Algoritma fuzzy dirancang dengan menggunakan 2 buah *input*, yaitu *error* dan *delta error* dari kecepatan. *Error* adalah selisih antara *setpoint* dan *output* sistem yang dibaca oleh sensor *rotary encoder*. *Error* digunakan untuk mengetahui seberapa besar deviasi yang

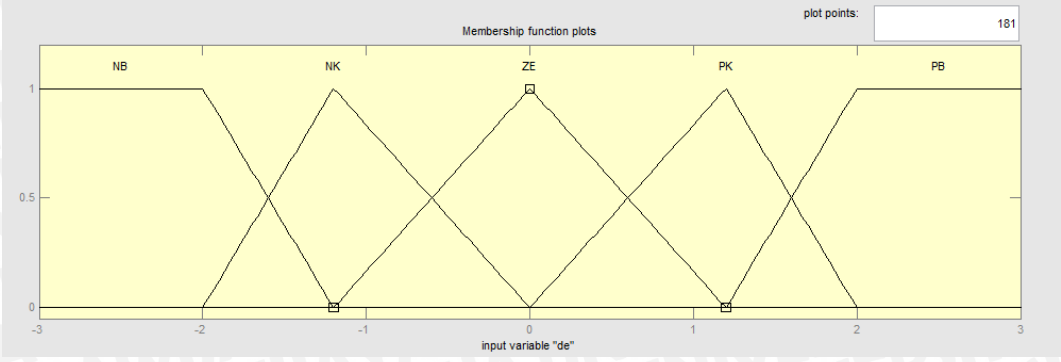
terjadi pada sistem sehingga nantinya akan menjadi *input* bagi logika *fuzzy* untuk memberi seberapa besar nilai pengaturan parameter PI, yaitu *Kp* dan *Ki*.

Selain *error* terdapat *delta error* yang berfungsi untuk mengetahui tingkat kesalahan *error* tersebut. Maksudnya adalah, seberapa besar perubahan *error* sekarang terhadap *error* sebelumnya. *Delta error* juga perlu dipertimbangkan dalam memutuskan nilai *Kp* dan *Ki*.

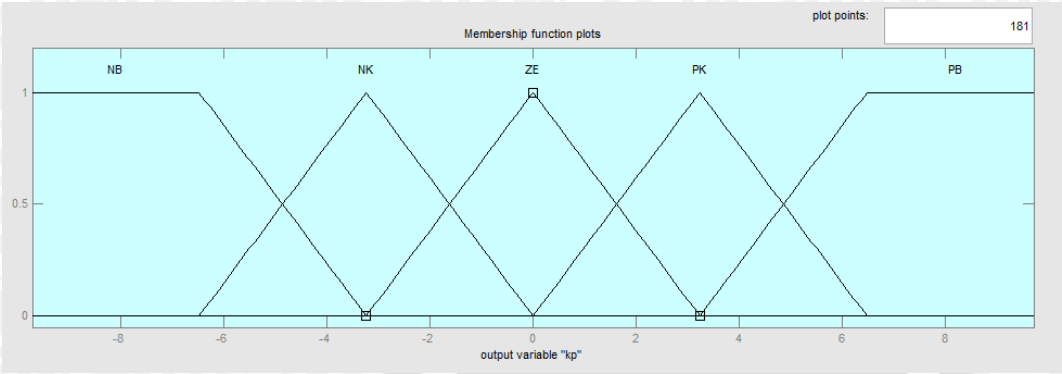
Dalam perancangan logika *fuzzy* dibuatlah sebuah fungsi keanggotaan untuk masing-masing *input* dan *output*. Setiap keanggotaan dideskripsikan dalam bahasa linguistik seperti NB (Negatif Besar), NK (Negatif Kecil), ZE (*Zero*/ Nol), PK (Positif Kecil), dan PB (Positif Besar). Digunakan *range* negatif hingga positif agar *error* dan *delta error* dapat tercakup secara keseluruhan mulai positif dan negatif. *Range* pada *ouput* juga dibuat sedemikian rupa agar *Kp* dan *Ki* mampu berfluktuasi mengikuti *setpoint*. Fungsi keanggotaan *error* ditunjukkan dalam gambar 3.21, dan fungsi keanggotaan delta *error* ditunjukkan dalam gambar 3.22. Fungsi keanggotaan *Kp* ditunjukkan dalam gambar 3.23, dan fungsi keanggotaan *Ki* ditunjukkan dalam gambar 3.24.



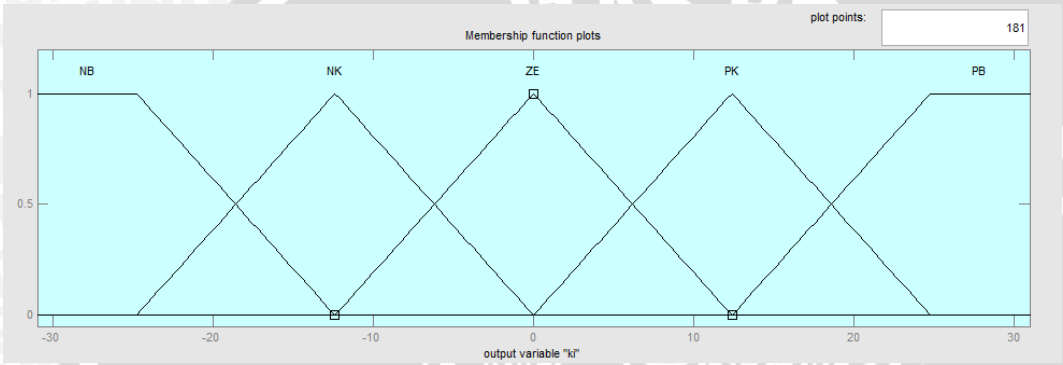
Gambar 3.21 Fungsi keanggotaan untuk *error*



Gambar 3.22 Fungsi keanggotaan untuk *delta error*



Gambar 3.23 Fungsi keanggotaan untuk *Kp*



Gambar 3.24 Fungsi keanggotaan untuk *Ki*

3.10.2. Aturan *Fuzzy*

Secara umum, hubungan antara *error* (*e*) dan *delta error* (*de*) terhadap nilai *Kp* dan *Ki* dijabarkan sebagai berikut:

1. Saat *e* relatif besar, maka kontroler akan memperbesar nilai *Kp* dan membuat nilai *Ki* sama dengan 0 (nol).
2. Saat *e* dan *de* relatif sesuai maka kontroler akan memperkecil nilai *Kp* agar mengurangi *overshoot*. Nilai *Ki* tidak berubah.
3. Saat *e* dan *de* nilainya sangat kecil maka kontroler akan memperbesar nilai *Kp* dan *Ki*.

Dengan acuan yang telah dijabarkan di atas, diperoleh tabel aturan *fuzzy* untuk *Kp* pada tabel 3.4 dan aturan *fuzzy* untuk *Ki* pada tabel 3.5

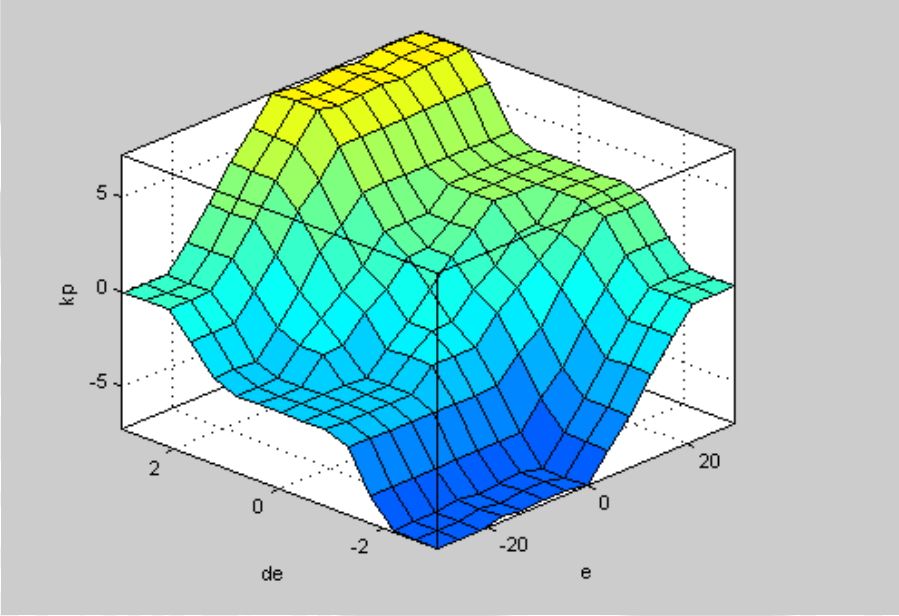
Tabel 3.4 Aturan *fuzzy* untuk *Kp*

dE / E	NB	NK	Z	PK	PB
NB	NB	NK	NK	NK	Z
NK	NB	NK	NK	Z	PK
Z	NB	NK	Z	PK	PB
PK	NK	Z	PK	PK	PB
PB	Z	PK	PK	PK	PB

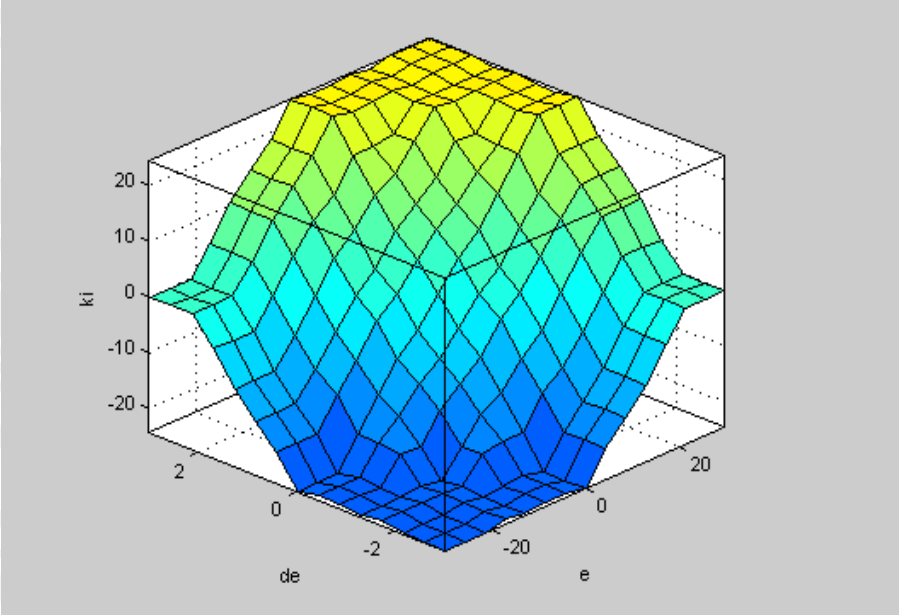
Tabel 3.5 Aturan *fuzzy* untuk *Ki*

dE / E	NB	NK	Z	PK	PB
NB	NB	NB	NB	NK	Z
NK	NB	NB	NK	Z	PK
Z	NB	NK	Z	PK	PB
PK	NK	Z	PK	PB	PB
PB	Z	PK	PB	PB	PB

Dari Tabel 3.4 dan Tabel 3.5 dapat digambarkan ruang solusi *fuzzy* untuk aturan *Kp* dalam Gambar 3.25 dan ruang solusi *fuzzy* untuk aturan *Ki* dalam Gambar 3.26



Gambar 3.25 Ruang solusi *fuzzy* untuk *Kp*



Gambar 3.26 Ruang solusi *fuzzy* untuk *Ki*

dari Gambar 3.25 dan 3.26, dapat diketahui bahwa perubahan parameter *Kp* dan *Ki* tidak lagi linier, namun mengikuti aturan *fuzzy* yang telah dibangun.

3.10.3. Metode Inferensi dan Defuzzifikasi

Setelah didapatkan fungsi keanggotaan dan aturan *fuzzy*, maka dapat ditentukan metode untuk inferensi dan defuzzifikasi dari sistem kontrol fuzzy tersebut. Pada penelitian kali ini, metoode inferensi yang digunakan adalah metode Min – Max (Mamdani), sedangkan untuk defuzzifikasi digunakan metode *Weighted Average*.

3.11. Perancangan Kontroler Proporsional Integral

3.11.1. Diskritisasi Persamaan Kontroler Proporsional Integral

Dalam kawasan waktu kontroler PI dapat dinotasikan dengan persamaan

$$c(t) = Kp(e(t) + \frac{1}{T_i}\int_0^t e(t)dt \quad (3\text{-}6)$$

Dimana c(t) adalah *output* kontroler, *Kp* adalah gain proporsional, *Ti* adalah waktu konstanta Integal atau *reset time* dan e(t) adalah *error* yang terjadi. Dari persamaan di atas dapat diubah menjadi kawasan frekuensi dengan Transformasi Laplace sehingga menjadi persamaan berikut:

$$C(s) = Kp\left(1 + \frac{1}{Ti\,s}\right)E(s) \quad (3\text{-}7)$$

Persamaan (3-7) belum bisa dimasukkan kedalam mikrokontroler karena persamaan tersebut masih berupa persamaan kontinyu. Maka persamaan kontinyu dalam persamaan (3-7) harus diubah kedalam bentuk diskrit melalui Transformasi Z. Dalam Transformasi Z dibutuhkan waktu cuplik (*Ts*). Digunakan metode *Billinear Transform* sehingga nilai notasi s pada Laplace setara dengan

$$s = \frac{2}{Ts}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \quad (3\text{-}8)$$

persamaan (3-8) disubstitusikan ke dalam persamaan (3-7) menjadi:

$$C(z) = \left[Kp + \frac{KpTs(1+z^{-1})}{2Ti(1-z^{-1})}\right] E(z) \quad (3\text{-}9)$$

Berikutnya adalah memodifikasi persamaan agar dapat disederhanakan. Kedua ruas pada persamaan (3-9) dikalikan dengan $(1\text{-}z^{-1})$:

$$C(z)(1-z^{-1}) = Kp\,E(z)(1-z^{-1}) + \frac{Kp\,Ts}{2Ti}E(z)(1+z^{-1}) \quad (3\text{-}10)$$

Persamaan (3-10) disusun kembali dengan *output* kontroler berada di ruas kiri persamaan:

$$C(z) = C(z)(z^{-1}) + Kp\,(E(z) - E(z)(z^{-1})) + \frac{Kp\,Ts}{2Ti}(E(z) + E(z)(z^{-1}) \quad (3\text{-}11)$$

Persamaan (3-11) diubah menjadi bentuk persamaan beda:

$$C(k) = C(k-1) + Kp\,\big(E(k) - E(k-1)\big) + \frac{Kp\,Ts}{2Ti}(E(k) + E(k-1)) \quad (3\text{-}12)$$

Dimana k-1 adalah kondisi sebelumnya. Persamaan (3-12) lalu dimasukkan ke dalam program pada mikrokontroler.

3.11.2. Penentuan Parameter PI dengan Metode *Symmetrical Optimum*

Pada kontroler PI, dibutuhkan sebuah parameter yang berfungsi untuk memperbaiki *settling time* dan *error steady-state output*. Untuk menentukan parameter tersebut digunakan metode *Symmetrical Optimum.* Penalaan parameter dengan metode *Symmetrical Optimum* pertama kali dikemukakan oleh Kessler pada 1958. Metode ini memaksimalkan *phase margin* dari sistem kontrol dan mengarahkan ke fasa yang simetris dan karakteristik amplitudo. Metode tersebut dipilih karena sistem memiliki keunggulan berupa kemantapan pada parameter *phase margin, gain margin,* dan dinamika sistem itu sendiri. (Barbossa, 2014).

Untuk mententukan konstanta dengan metode *Symmetrical Optimum*, pertama – tama persamaan fungsi alih dimodifikasi menjadi *open loop system* seperti berikut:

$$F_{ol}(s) = \underbrace{Gc\left(\frac{T_i s+1}{T_i s}\right)}_{Kontroler\ PI} \underbrace{\frac{G_m}{T_m s+1}}_{Plant} \frac{1}{T_{mn} s} \tag{3-13}$$

Untuk membentuk persamaan sistem diatas maka fungsi alih motor perlu diubah, di mana *Gm* adalah nilai *gain* dari motor yang telah dimodifikasi dan *Tm* adalah *Time Constant* motor. Oleh Karena itu, persamaan (3-1) harus diubah terlebih dahulu ke bentuk

$$F(s) = \frac{G_{cw}}{(T_1 s+1)(T_2 s+1)} \tag{3-14}$$

Mulanya bagian penyebut diakarkan menjadi

$$G(s) = \frac{1182}{s^2+125{,}3s+1985}$$

$$G(s) = \frac{1182}{(s+18{,}6)(s+106{,}7)}$$

$$G(s) = \frac{1182}{(18{,}6)(106{,}7)(\frac{1}{18{,}6}s+1)(\frac{1}{106{,}7}s+1)}$$

$$G(s) = \frac{0.596}{(\frac{1}{18{,}6}s+1)(\frac{1}{106{,}7}s+1)} \tag{3-15}$$

$$T_1 = \frac{1}{18{,}6} = 0{,}054 \tag{3-16}$$

$$T_2 = \frac{1}{106{,}7} = 0{,}00937 \tag{3-17}$$

$$T_2 < T_1 \tag{3-18}$$

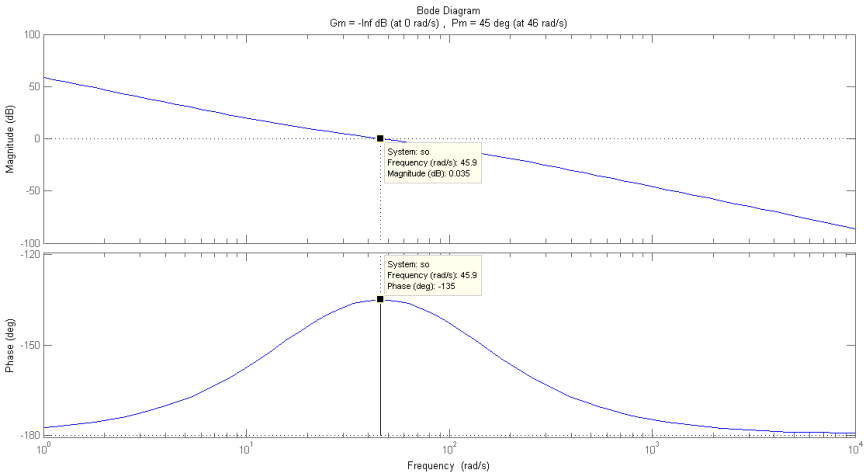
Pada persamaan (3-15) didapatkan nilai *Gcw* dan pada persamaan (3-16), (3-17), dan (3-18) didapatkan nilai *Tcw*. Nilai *Gcw* adalah 0,596 dan *Tcw* = T_1 = 0,054. Nilai *Tmn* ditetapkan dengan *Tmn* = 1. Dengan menentukan nilai faktor redaman (*D*) sebesar *D* = 0,707, maka dapat ditentukan nilai *Kp* dan *Ti* melalui persamaan berikut:

$$K_p = \frac{1}{a\ Gcw} \frac{Tmn}{Tcw} \tag{3-19}$$

$$a = 2D + 1 \tag{3-20}$$

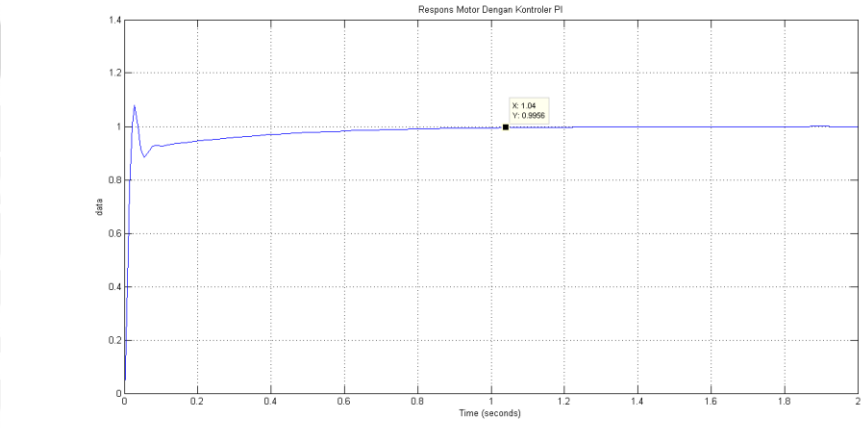
$$T_i = a^2\ Tcw \qquad a > 1 \tag{3-21}$$

Nilai-nilai dari variabel *Tcw*, *Gcw*, *Tmn* dan *D* dimasukkan ke dalam persamaan (3-19), (3-20), dan (3-21) untuk diolah menggunakan MATLAB dan hasilnya ditampilkan dalam bodeplot untuk mengetahui kestabilan sistem. Pada gambar 3.27, dapat diketahui dari bodeplot tersebut bahwa sistem stabil, dengan *frequency crossover* pada margin terletak pada fasa maksimumnya.



Gambar 3.27 Hasil Bodeplot dengan metode *Symmetrical Optimum*

Berdasarkan perhitungan menggunakan MATLAB, dengan nilai *Tmn* = 1 dan *D* = 0,707 didapatkan nilai *Kp* = 12,938 dan *Ti* = 0,3133. Sehingga nilai *Kp* = 12,938 dan *Ki* = 41,298. Nilai *Kp* dan *Ki* tersebut disimulasikan pada *toolbox* SIMULINK untuk mengetahui *output* sistem dengan kontroler PI dengan diberi *input* berupa sinyal *unit step*. Hasil simulasi *output* sistem dapat dilihat dalam Gambar 3.28.



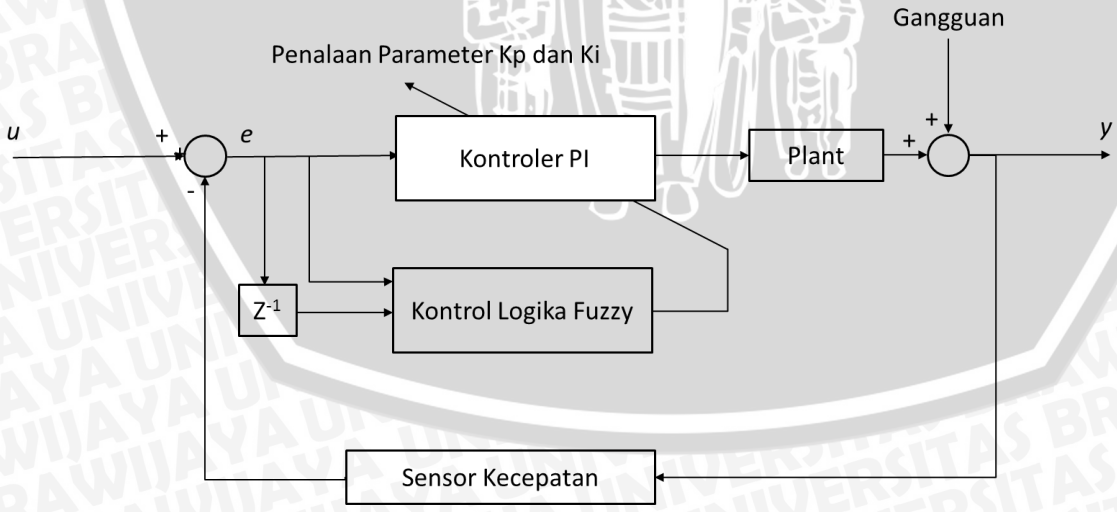
Gambar 3.28 Simulasi *output* sistem dengan kontroler PI

Besarnya nilai *settling time* setelah diberi kontroler adalah 1,04 s dan mampu mencapai *setpoint*.

3.12. Perancangan Kontroler *Self-Tuning* Parameter PI

Setelah melakukan perancangan kontrol logika *fuzzy* dan kontroler PI, maka kedua sistem tersebut digabungkan. Secara umum, sistem kontrol akan bekerja dengan kontrol logika *fuzzy* berfungsi untuk menala parameter PI secara *real time*, dan kontroler PI berfungsi untuk mengolah sinyal *error*, dan memberikan sinyal kontrol.

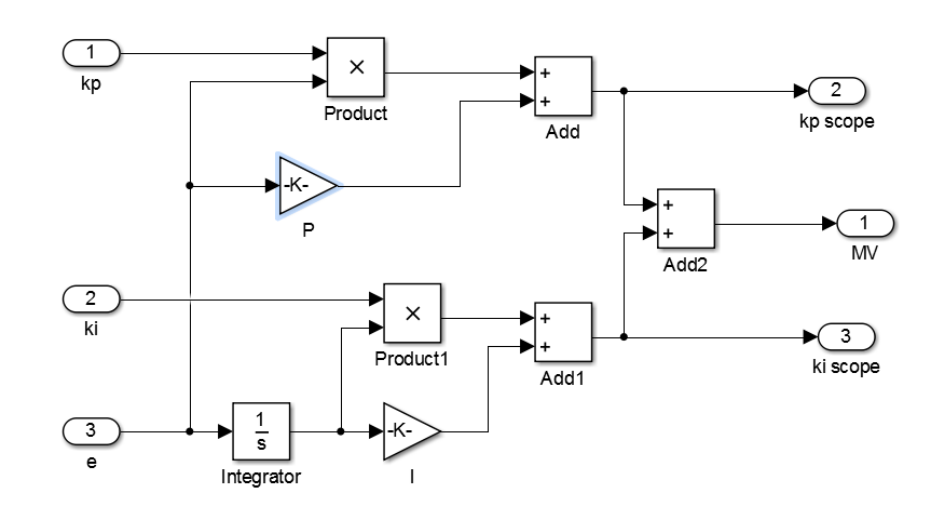
Input atau *setpoint* (*u*) adalah nilai kecepatan yang diberikan oleh pengguna, berupa kecepatan putar per menit atau *rotation per minute* (*rpm*). Nilai *input* tersebut dimasukkan ke dalam *summing point* untuk dikurangi oleh nilai *output* (*y*). Hasil dari operasi pengurangan tersebut adalah *error*, yaitu deviasi atau simpangan antara pembacaan aktual kecepatan motor dari *rotary encoder* (*output*) dan nilai *setpoint* (*input*). *Error* menjadi *input* bagi kontrol logika *fuzzy* dan kontroler PI, di mana kontrol logika *fuzzy* mendapatkan satu *input* lagi yaitu *delta error*. *Delta error* adalah besar perubahan *error* sekarang terhadap *error* sebelumnya. Dari kedua *input* tersebut dihasilkan nilai parameter PI hasil penalaan kontrol logika *fuzzy*. Parameter tersebut menjadi *input* parameter kontroler PI dengan nilai bobot tertentu. *Input error* kontroler PI dikalkulasi oleh kontroler PI yang telah memiliki nilai parameter PI tetap dan parameter PI hasil penalaan kontrol logika *fuzzy*. Hasil dari kalkulasi tersebut berupa sinyal kontrol yang digunakan sebagai *input* pada plant, sehingga *plant* dapat bergerak sesuai dengan *setpoint* yang telah ditentukan, meskipun dalam implementasinya terdapat gangguan. Diagram blok sistem secara keseluruhan dapat dilihat dalam Gambar 3.29.



Gambar 3.29 Diagram blok perancangan kontroler *self-tuning* parameter PI

Agar kontroler PI dapat bekerja secara optimal, maka diperlukan perancangan kontroler PI yang bobot parameter PI-nya dapat diubah, dan dapat bersinkronisasi dengan *output* parameter PI hasil penalaan kontrol logika *fuzzy*.

Input error kontroler PI dimasukkan ke dalam blok parameter PI tetap. *Input* parameter PI hasil penalaan kontrol logika *fuzzy* dimasukkan ke dalam blok perkalian untuk dikalikan dengan *input error*. *Output* dari blok perkalian tersebut dimasukkan kedalam blok penjumlahan untuk dijumlahkan dengan hasil perhitungan dari parameter PI tetap. Hasil dari penjumlahan tersebut adalah *Manipulated Variable* (*MV*), dan berfungsi sebagai sinyal kontrol. Dengan begitu, perbandingan bobot penalaan dapat diubah dengan cara mengatur parameter PI tetap dan semesta fungsi keanggotaan *fuzzy*. Diagram blok rancangan subsistem kontroler PI tersebut dapat dilihat dalam Gambar 3.30.



Gambar 3.30 Rancangan blok subsistem kontroler PI