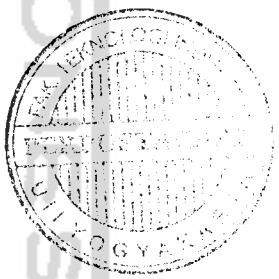
# PENGEREMAN MOTOR DC
# DENGAN KONTROL PID

**TUGAS AKHIR**

Diajukan Sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana
Teknik Elektro

Oleh :
**SIR BRAM NOVALDI SARIPUTRA**
01 524 112

**JURUSAN TEKNIK ELEKTRO**
**FAKULTAS TEKNOLOGI INDUSTRI**
**UNIVERSITAS ISLAM INDONESIA**
**JOGJAKARTA**
**2007**

**LEMBAR PENGESAHAN PEMBIMBING**

# PENGEREMAN MOTOR DC
# DENGAN KONTROL PID

## TUGAS AKHIR

Oleh :

Nama : SIR BRAM NOVALDI SARIPUTRA

No.Mhs : 01 524 112

Jogjakarta, Januari 2007

Disahkan oleh :

| Pembimbing I | Pembimbing II |
|---|---|
| (Wahyudi Budi Pramono,ST) | (Dwi Ana Ratna Wati,ST) |

UNIVERSITAS ISLAM INDONESIA

# MOTTO

HIDUP DI DUNIA ADALAH PERJUANGAN MENUNGGU KEMATIAN.

DAN KEMATIAN ADALAH SANG PEMUTUS KELEZATAN.

APAPUN YANG KAU KERJAKAN, KEMANAPUN KAU BERJALAN,

SUNGGUH KAU AKAN BERTEMU DENGAN KEMATIAN.

MAKA HIDUPLAH DIDUNIA INI SEPERTI SEORANG PENGEMBARA,

ATAU ORANG YANG NUMPANG LEWAT SAJA,

YANG TIDAK BERAMBISI UNTUK MEMBANGUN RUMAH

MEGAHNYA DALAM PENGEMBARAANNYA.

DAN SIAPKAN DIRIMU MENJADI PENGHUNI KUBUR,

DAN HIDUP DALAM KEHIDUPAN YANG KEKAL DAN ABADI

DI KAMPUNG KITA...KAMPUNG AKHIRAT NANTI.....

# KATA PENGANTAR

بسم الله الرحمن الرحيم

Assalamu'alaikum Wr. Wb.

Puji syukur kehadirat Allah SWT, atas nikmat iman, rahmat, hidayah dan pikiran yang diberikan. Sehingga penulis dapat menyelesaikan tugas akhir dengan judul "**Pengereman Motor DC dengan Kontrol PID**". Tidak lupa shalawat serta salam selalu tercurah kepada Nabi Muhammad. SAW beserta keluarga dan para sahabatnya.

Adapun maksud dari penyusunan tugas akhir ini adalah untuk memenuhi kurikulum S-1 Jurusan Teknik Elektro, Fakultas Teknologi Industri, Universitas Islam Indonesia. Disamping itu untuk menambah pengetahuan terhadap ilmu yang telah dipelajari di bangku perkuliahan untuk diterapkan ke masyarakat.

Dalam penyusunan ini, penulis banyak mendapat bantuan dari berbagai pihak, sehingga penulis ingin menyampaikan ucapan terima kasih kepada :

1. Kedua orang tuaku yang senantiasa memberikan dukungan moril, materi dan doa setiap saat.
2. Bpk Fathul Wahid,ST,MSc, selaku Dekan Fakultas Teknologi Industri (FTI) Universitas Islam Indonesia (UII)
3. Bapak Tito Yuwono,ST,MSc, selaku Kajur Teknik Elektro.
4. Bpk Wahyudi Budi Pramono, ST. selaku dosen pembimbing I atas bimbingannya.

5. Ibu Dwi Ana Ratna Wati, ST. selaku dosen pembimbing II atas waktu, kesabaran dan ilmunya.
6. Mas Romi Wiryadinata,ST atas waktu, kesabarannya serta bimbingannya.
7. Dosen dan karyawan Fakultas Teknologi Industri UII.
8. Istriku yang saya cintai, Peni Purwandari,SE atas dorongan semangatnya.
9. Adikku, Welly Ernando Putra serta keluarga besar yang telah memberikan dukungan, perhatian dan doanya.
10. Seluruh mahasiswa jurusan Teknik Elektro UII,
11. sahabat-sahabatku di Asrama tercinta Al Zain dan Al Mahfudz atas doa dan dukungannya serta keceriaannya.
12. Asrama Takmir Masjid Ulil Albab Universitas Islam Indonesia atas tempatnya selama proses menimba ilmu.
13. Seluruh pihak yang tidak dapat di sebutkan satu-persatu, yang telah memberikan *support* dan doa.

Penulis menyadari bahwa Tugas Akhir ini masih terdapat kesalahan dan kekurangan. Oleh karena itu, kritik dan saran yang membangun akan senantiasa penulis terima dengan senang hati. Akhirnya, harapan penulis semoga Tugas Akhir ini dapat bermanfaat bagi kita semua. Amiin...

Wassalamu'alaikum Wr.Wb

Yogyakarta, 29 Januari 2007

Sir Bram Novaldi Sariputra

# BRAM THANK`S

1. Segala puji dan syukur saya panjatkan kepada Allah *Ta`ala.*, atas segala nikmatNya yang tak terhitung jumlahnya khususnya nikamat istiqomah diatas islam hingga akhir hayat. Shalawat dan salam bagi RasulAllah Muhammad *Shalallahu `Alayhi Wasallam.* dan segenap para sahabat beliau *ridlwanullah alayhim ajma`ain*, dan para pengikut beliau yang istiqomah hingga hari Kiamat nanti.
2. Ibu dan Ayah yang sangat kucintai, atas segala dorongan dan pengorbanannya biaya wakti dan perhatiannya dalam penyelesaian tugas akhir ini.
3. Istriku sang "*penyejuk mataku*" Peni Purwandari,SE atas dorongan semangat perhatiannya dalam kelancaran penyelesaian tugas akhir ini.
4. Adikku, Welly Ernando Putra serta keluarga besar yang telah memberikan dukungan, perhatian dan doanya.
5. Semua Guru-guru dan teman-temanku dari mulai TK, , SD, SMP, SMA dan kulaih.
6. Temen-temen main dan belajar dirumah dan di jogja, temen kost dan kontrakan, asrama Takmir Masjid Ulil Albab dan semua yang ada di jurususan elektro..
7. Semua yang merasa mengenal saya dan pernah membantu dalam penyelesaian tugas akhir ini.

# ABSTRAK

Motor DC dan komputer banyak digunakan dalam kehidupan sehari-sehari, baik di rumah tangga, industri maupun lingkungan pendidikan yang sangat membutuhkan ketelitian dan penggunan yang serba otomatis. Kontrol *PID* merupakan salah satu kendali motor DC yang dapat disimulasikan menggunakan *Simulink* pada *software* Matlab 6.5. Pengendali *PID* adalah pengendali berumpan balik yang paling populer di dunia industri. Dalam mendesain pengendali *PID*, harus ditentukan parameter dari P, I, D agar didapatkan respon sistem yang memenuhi kriteria yang diinginkan. Keseluruhan dari pengendali tersebut bertujuan mempercepat reaksi sebuah sistem, menmghilangkan offset dan menghasilkan perubahan awal yang besar. Pengendali *PID* dapat mengendalikan sisa putaran motor dc. Untuk menunjukkan pengaruh pengendali *PID* dalam mengatur sisa putaran (pengereman) motor dc, maka masing-msing parameter dapat diisi dengan nilai-nilai yang dapat mewakilinya berikut ini: $P$ =.-0.004, $I$ = -0.0004, $D$ = -0.0005. Hasil yang diperoleh bahwa pengendali *P&I* saja sudah cukup untuk mengendalikan (mengerem) sisa putaran motor tanpa disertakan pengendali derivatif, walaupun dengan pengendali *PID* juga mampu untuk mengerem sisa putara motor. Namun pengendali *PID* membutuhkan waktu pengereman lebih lama dibandingkan dengan pengendali *P&I*. Sesuai dengan karakteristik masing-masing pengendali, yaitu: proporsional berfungsi menambah waktu naik, integral berfungsi menghilangkan kesalahan keadaan tunak, Derivatif berfungsi mengurangi overshoot.

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

LEMBAR PENGESAHAN DOSEN PENGUJI

# PENGEREMAN MOTOR DC DENGAN KONTROL PID

TUGAS AKHIR

Disusun Oleh:


NAMA : SIR BRAM NOVALDI SARIPUTRA

NO.MHS : 01524112


**TELAH DIPERTAHANKAN DIDEPAN SIDANG PENGUJI SEBAGAI SALAH SATU SYARAT UNTUK MEMPEROLEH GELAR SARJANA TEKNIK ELEKTRO FAKULTAS TEKNOLOGI INDUSTRI**

Jogjakarta, Januari 2007

**Tim Penguji:**

**Ketua**

**Wahyudi Budi Pramono,ST**

**Anggota I**

**Dwi Ana Ratnawati,ST**

**Anggota II**

**Yusuf Aziz Amrulloh,ST**

Mengetahui
Ketua Jurusan Teknik Elektro
Universitas Islam Indonesia

Tito Yuwono,ST,MSc

# Halaman Persembahan

Karya ini kupersembahkan untuk IBU & AYAHku yang kucintai atas segala pengorbanan yang IBU & AYAH lakukan untuk keberhasilan putramu ini. Semoga Allah Ta`ala memberikan ganjaran dan kasih sayangNya yang tak terhingga kepada kalian berdua. Amin

Juga untuk siapa saja yang ingin mengambil manfaat dari karya ini. Semoga karya yang sedikit ini dapat memberi manfaat yang banyak bagi pembacanya, dan dihitung sebagai amal sholih bagi kami di sisi Allah Ta`ala. Amin

# ABSTRAK

Motor DC dan komputer banyak digunakan dalam kehidupan sehari-sehari, baik di rumah tangga, industri maupun lingkungan pendidikan yang sangat membutuhkan ketelitian dan penggunan yang serba otomatis. Kontrol *PID* merupakan salah satu kendali motor DC yang dapat disimulasikan menggunakan *Simulink* pada *software* Matlab 6.5. Pengendali *PID* adalah pengendali berumpan balik yang paling populer di dunia industri. Dalam mendesain pengendali *PID*, harus ditentukan parameter dari P, I, D agar didapatkan respon sistem yang memenuhi kriteria yang diinginkan. Keseluruhan dari pengendali tersebut bertujuan mempercepat reaksi sebuah sistem, menmghilangkan offset dan menghasilkan perubahan awal yang besar. Pengendali *PID* dapat mengendalikan sisa putaran motor dc. Untuk menunjukkan pengaruh pengendali *PID* dalam mengatur sisa putaran (pengereman) motor dc, maka masing-msing parameter dapat diisi dengan nilai-nilai yang dapat mewakilinya berikut ini: $P$ =.-0.004, $I$ = -0.0004, $D$ = -0.0005. Hasil yang diperoleh bahwa pengendali *P&I* saja sudah cukup untuk mengendalikan (mengerem) sisa putaran motor tanpa disertakan pengendali derivatif, walaupun dengan pengendali *PID* juga mampu untuk mengerem sisa putara motor. Namun pengendali *PID* membutuhkan waktu pengereman lebih lama dibandingkan dengan pengendali *P&I*. Sesuai dengan karakteristik masing-masing pengendali, yaitu: proporsional berfungsi menambah waktu naik, integral berfungsi menghilangkan kesalahan keadaan tunak, Derivatif berfungsi mengurangi overshoot.

# BAB I

# PENDAHULUAN

## 1.1. Latar Belakang Masalah

Mesin dc banyak digunakan dalam industri modern. Sebab pengaturan kecepatan kerja mesin dc dalam rentang yang lebar relatif lebih mudah, disamping banyaknya metoda yang dapat digunakan. Metoda yang paling banyak dipergunakan dalam pengaturan kecepatan adalah pengendalian fluks medan, pengaturan tahanan rangkaian jangkar kumparan dan pengendalian tegangan terminal jangkar. Pesatnya kemajuan teknologi terutama pada akhir-akhir ini semakin memanjakan konsumen. Dengan itu masyarakat dapat melakukan segala aktivitasnya dengan mudah. Di era globalisasi, sektor industri memegang peranan penting khususya di Indonesia. Banyak industri yang berkembang dengan pesat. Dalam dunia industri kemajuan teknologi dapat dirasakan sekali terutama dalam hal pengendalian. Karena sistem pengendalian yang baik dapat menunjang proses di industri tersebut dan meningkatkan efisiensi dalam proses poduksi.

Kontrol otomatik telah memegang peranan yang sangat penting dalam perkembangan ilmu dan teknologi. Kontrol otomatik telah menjadi bagian yang penting dan terpadu dari proses-proses dalam pabrik dan industri modern. Sebagai contoh, kontrol automatik sangat diperlukan dalam operasi-operasi di industri untuk mengontrol tekanan, temperatur, kecepatan dan sebagainya.

Karena kemajuan dalam teori dan praktek kontrol otomatik memberikan kemudahan dalam mempertinggi kualitas dan menurunkan biaya produksi,

# BAB II

# DASAR TEORI

## 2.1 MOTOR ARUS SEARAH (DC)

Motor adalah mesin yang mengubah energi listrik menjadi energi mekanis. Konstruksi motor sama dengan genertor. Mesin yang bekerja baik sebagai generator akan bekerja pula sebagai motor.

### 2.1.1 Prinsip kerja motor arus searah (DC)

Prinsip kerja motor searah berdasarkan pada penghantar yang membawa arus ditempatkan dalam suatu medan magnet maka penghantar tersebut akan mengalami gaya. Gaya menimbulkan torsi yang akan menghasilkan rotasi mekanik, sehingga motor akan berputar. Jadi motor arus searah ini menerima sumber arus searah dari jala-jala kemudian diubah menjadi energi mekanik berupa perputaran, yang nantinya dipakai oleh peralatan lain. Secara matematis putaran motor dapat dirumuskan seperti persamaan

$$n = \frac{V_t - I_a \cdot R_a}{C \cdot \phi} \qquad (2.1)$$

$V_t$ : Tegangan Sumber

$I_a$ : Arus jangkar (armature)

$R_a$ : Tahanan kumparan jangkar

n : Jumlah putaran motor

$\phi$ : Fluks magnet

C : Konstanta

mempertinggi laju produksi, meniadakan pekerjaan-pekerjaan rutin dan membosankan yang harus dilakukan oleh manusia, dan sebagainya.

## 1.2 Perumusan Masalah

Berdasarkan uraian dari latar belakang diatas maka dapat dirumuskan permasalahan yaitu bagaimana merancang pengendali otomatik *PID* yang dapat mengendalikan kecepatan pengereman motor DC sesuai dengan yang diinginkan.

## 1.3 Batasan Masalah

Dengan adanya batasan masalah, penulis dapat lebih menyederhanakan dan mengarahkan penelitian dan pembuatan sistem agar tidak menyimpang dari apa yang diteliti dan dikembangkan. Batasan-batasannya adalah sebagai berikut :

1. Motor dc dimodelkan dengan model motor dc yang ada pada *simulink Matlab*.
2. Pembuatan sistem disimulasikan menggunakan perangkat lunak *Matlab 6.5*.
3. Menjalankan simulasi simulink pengereman Motor dc dengan menggunakan *GUI (Graphic User Interface)* Matlab.
4. Tidak membahas kondisi starting motor.

## 1.4 Tujuan Penelitian

Merancang pengendali *PID* yang dapat mengendalikan pengereman motor dc sesuai dengan yang diinginkan.

## 1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat menjadi salah satu kontribusi positif dalam perkembangan sistem pengendalian pengereman motor dc. Sehingga pengereman motor dc dengan pengendali *PID* dapat menjadi pilihan bagi masyarakat industri khususnya berhubung jenis kontrol ini telah populer dan dikenal luas.

## 1.6 Metodelogi Penelitian

1. Pengumpulan Data

   Data diperoleh dari studi pustaka berupa buku, artikel, makalah dan tutorial yang tersedia pada *website* di internet.

2. Studi Pustaka

   Pengumpulan data ini digunakan untuk mendapatkan informasi-informasi yang berkaitan dengan proses penyusunan tugas akhir, sehingga dapat digunakan sebagai acuan dalam proses pembuatan simulasi.

3. Pemecahan Masalah

   Setelah semua data terkumpul, maka dilakukan perancangan sistem, pembutan simulasi sistem dan pengujian sistem.

## 1.7 Sistematika Penulisan

Sistematika penulisan tugas akhir ini terdiri dari 5 bab bagian isi laporan, dengan penjelasan bab sebagai berikut :

**BAB I : PENDAHULUAN**

Berisi tentang latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian dan sistematika penulisan.

**BAB II : LANDASAN TEORI**

Memuat dasar-dasar teori mengenai pengendali *PID* dan pemodelan motor *DC* yang akan digunakan. Juga teori-teori tentang *GUI (Graphic User Interface)* pada *Matlab*.

**BAB III : PERANCANGAN SISTEM**

Menjelaskan tentang rancangan pengendali *PID*, pembagian fungsi kerja dalam diagram blok dan diagram alir serta berisi lebih terperinci tentang apa yang telah disampaikan pada proposal Tugas Akhir.

**BAB IV : PENGUJIAN, ANALISIS DAN PEMBAHASAN**

Membahas tentang hasil pengujian dan analisis dari sistem yang dibuat dibandingkan dengan dasar teori sistem atau uraian alasan ilmiah yang lain.

**BAB V : PENUTUP**

Berisi kesimpulan dan saran-saran dari proses perancangan, simulasi sistem, serta keterbatasan-keterbatasan yang ditemukan dan juga asumsi-asumsi yang dibuat selama melakukan penelitian.

# BAB II

# DASAR TEORI

## 2.1 MOTOR ARUS SEARAH (DC)

Motor adalah mesin yang mengubah energi listrik menjadi energi mekanis. Konstruksi motor sama dengan genertor. Mesin yang bekerja baik sebagai generator akan bekerja pula sebagai motor.

### 2.1.1 Prinsip kerja motor arus searah (DC)

Prinsip kerja motor searah berdasarkan pada penghantar yang membawa arus ditempatkan dalam suatu medan magnet maka penghantar tersebut akan mengalami gaya. Gaya menimbulkan torsi yang akan menghasilkan rotasi mekanik, sehingga motor akan berputar. Jadi motor arus searah ini menerima sumber arus searah dari jala-jala kemudian diubah menjadi energi mekanik berupa perputaran, yang nantinya dipakai oleh peralatan lain. Secara matematis putaran motor dapat dirumuskan seperti persamaan

$$n = \frac{V_t - I_a \cdot R_a}{C \cdot \phi} \qquad (2.1)$$

$V_t$ : Tegangan Sumber

$I_a$ : Arus jangkar (armature)

$R_a$ : Tahanan kumparan jangkar

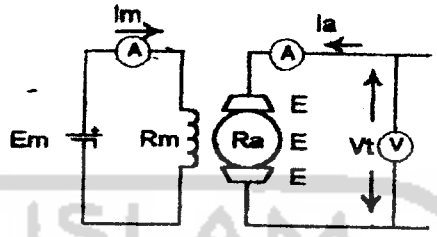
n : Jumlah putaran motor

$\phi$ : Fluks magnet

C : Konstanta

### 2.1.2 Karakteristik Motor DC

Di bawah ini adalah gambar rangkaian ekivalen motor DC *shunt* dengan eksitasi terpisah beserta persamaan yang berlaku :




Gambar 2.1 Rangkaian ekivalen motor DC

$$E_a = V_t - I_a R_a , \tag{2.2}$$

$$Vt = C\, n\, \phi , \tag{2.3}$$

$$n = {}^{Vt - Ia\,Ra}/_{C\,\phi} , \tag{2.4}$$

Ea = Tegangan armatur

Vt = Tegangan yagn dibangkitkan oleh rangkaian armatur

Ia = Induktansi armatur

Ra = Resistansi armatur

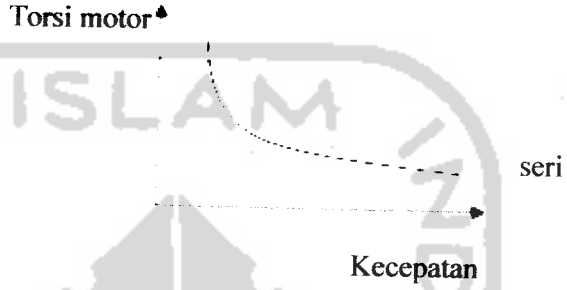
C = Konstanta

n = kecepatan

Φ = fluks medan

Dari persamaan 2.4 diketahui bahwa pada motor *shunt*, bertambahnya beban (arus jangkar bertambah) mengakibatkan kecepatan (n) menurun. Pada motor seri, bertambahnya beban (arus) akan menyebabkan bertambahnya harga fluks (ϕ),

karena fluks pada motor seri merupakan merupakan fungsi arus jangkar. Untuk harga arus jangkar sama dengan nol, harga fluks juga mendekati nol sehingga dari persamaan 2.4, diperoleh harga n menuju tak terhingga. Sedangkan untuk harga $I_a$ yang cukup besar, harga n akan mendekati nol. Dengan demikian karakteristik kecepatan-torsi untuk motor seri dapat digambarkan sebagai berikut:




Gambar 2.2 karakteristik kecepatan-torsi motor seri

Pada motor dengan medan magnet permanen, medan magnetnya dihasilkan oleh satu atau beberapa magnet permanen. Magnet-magnet ini digenggam oleh besi atau baja, atau terkadang oleh rangka motor itu sendiri. Magnet ini merupakan bagian motor yang diam di tempatnya (stator). Kawat yang mengalirkan arus listrik digulung pada bagian motor yang berputar (rotor). Rotor yang terdapat pada motor sederhana, dibuat menjadi tiga buah kutub kumparan yang dibuat dari logam berlapis.

Fluks magnet menempuh suatu lintasan tertutup melalui rangka motor. Mengalirnya arus di dalam suatu kumparan kawat akan menimbulkan gaya. Gaya ini akan menggerakan rotor sampai arah gayanya sejajar dengan arah medan magnet. Pada keadaan seperti ini, rotor akan tetap diam dan tidak akan berputar

lagi. Akan tetapi, jika arusnya dihubungkan ke salah satu kumparan lainnya, maka motor akan bergerak kembali sampai berada pada posisi sejajar yang baru.

Arus dialirkan ke kumparan rotor melalui dua buah sikat yang sekaligus merupakan kontak dengan cincin penghantar pada rotor (komutator). Komutator dibagi menjadi tiga bagian yang disusun berdekatan. Kedua sikat akan memindahkan arus dari satu kumparan ke kumparan lainnya sesuai dengan putaran motor. Salah satu kekurangan dari motor sederhana ini adalah adanya perubahan torsi pada setiap putaran motor. Pada kecepatan tinggi, perubahan torsi tidak masalah dan masih ada beban dan kelembaman atau *inertia* yang memperhalus gerakan motor. Pada kecepatan rendah, perubahan torsi membuat putaran motor cenderung meloncat dari satu posisi ke posisi lainnya.

### 2.1.3 Pengaturan Kecepatan Motor

Tiga metode dasar pengendalian kecepatan yaitu:

1. Pengendalian fluksi medan

   Arus medan dan juga fluksi medan dalam motor *shunt* atau kompon dapat diubah dengan mengatur tahanan geser medan yang dihubungkan secara seri dengan medan *shunt*. Dengan menaikkan tahanan dalam rangkaian medan akan menyebabkan penurunan dalam fluksi medan dan oleh sebab itu menaikkan kepesatan. Dan sebaliknya, menurunkan tahanan rangkaian medan akan menyebabkan berkurangnya kepesatan.

2. Pengendalian tahanan rangkaian jangkar

Tahanan rangkain jangkar motor dapat diubah dengan menambahkan tahanan variabel yang dihubungkan seri dengan jangkar. Bila tahanan seri dinaikkan, tegangan pada jangkar motor berkurang dan kepesatan motor turun. Dan sebaliknya, kepesatan motor akan bertambah jika tahanan seri dikurangi. Ini adalah metode pengendalian kepesatan yang biasanya digunakan untuk motor seri.

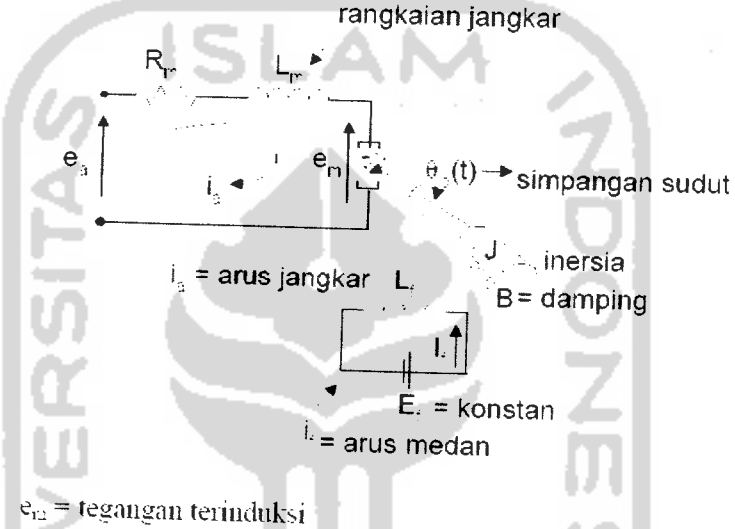
3. Pengendalian tegangan terminal

Kepesatan motor dapat dikendalikan dengan mengubah tegangan yang dikenakan pada rangkaian jangkar. Sebelum adanya komponen zat padat yang relatif tidak mahal, perlengkapan utnuk catu tegangan dc yang dapat diatur terasa sangat mahal. Oleh sebab itu untuk mengendalikan motor-motor yang sangat besar, pengendalian kecepatan pada metode ini sebagian besar terbatas pada sistem *Ward-Leonard* atau variasinya.

Sistem *Ward-Leonard* atau sistem tegangan yang dapat diatur adalah sistem pengereman dinamis yang biasanya digunakan untuk motor dc yang memerlukan daerah pengendalian pengendalian kecepatan yang luas dan tahapan yang halus. Sistem ini melibatkan generator lain yang menggerakkan motor yang kepesatannya dapat diatur. Dengan mengubah eksitasi medan generator, tegangan yang dikenakan pada motor dapat diubah secara luas dan menghasilkan daerah kepesatan (kecepatan) yang luas pada motor. Dengan kata lain sistem Ward-Leonard adalah sistem yang sangat efektif, karena dengan *pengereman dinamis*, respons terhadap perubahan kepesatannya cepat

dan daerah penyetelan kepesatannya luas. Hal-hal tersebut dapat menutupi kekurangan yang ada pada sistem ini yaitu mahalnya biaya awal dan relatif tidak efisien.

### 2.1.4 Pemodelan motor dc




Gambar 2.3 Model motor dc

$$e_m = k_1 . n \tag{2.5}$$

$$= k_2 . i_f \tag{2.6}$$

Dengan : n= kecepatan rotasi (putaran) motor

$\Phi$ = konstan

$I_f$ = konstan

Sehingga :

$$e_m = k_e . n = k_e \frac{d\theta}{dt} \tag{2.7}$$

$k_e$ = konstanta tegangan motor

*Transformasi Laplace* untuk persamaan rangkaian:

$$e_\alpha = R_m\, i_a + L_m \frac{d_{ia}}{\mathrm{dt}} + e_m$$

$$e_a = R_m\, i_a + L_m \frac{d_{ia}}{dt} + k_e \frac{\mathrm{d}\,_o}{dt}$$

$$E_a\,(\mathrm{s}) = (R_m + sL_m)\, I_a(s) + k_e s\ {}_o(s) \qquad (2.8)$$

Keterangan:

| | | |
|---|---|---|
| $e_\alpha$ | = | tegangan armatur |
| $R_m$ | = | Resistansi rangkaian armatur |
| $i_a$ | = | arus armatur |
| $L_m$ | = | induktansi rangkaian armatur |
| $e_m$ | = | tegangan kumparan armature |
| $k_e$ | = | parameter motor |
| $\frac{d_0}{dt}$ | = | kecepatan sudut poros |

Persamaan beban:

Torsi yang dihasilkan motor sebanding dengan fluksi (yang dalam hal ini konstan) dan sebanding dengan arus jangkar $i_a$. *Transformasi Laplacenya*:

$$T = k_T \,.\, i_a \qquad (2.9)$$

$K_T$ = konstansta torsi motor

Persamaan akhir diturunkan dari penjumlahan torsipada armature motor. Pada Gambar 2.3, momen inersia *J* meliputi semua inersia yang dihubungkan ke poros

motor dan $B$ meliputi gesekan udara dan gesekan bantalan poros. Oleh karena itu persamaan torsi adalah:

$$T = J \frac{d^2\rho}{dt^2} + B\frac{d}{dt}$$

Atau

$$k_T I_a(s) = (Js^2 + Bs)\,\Theta_o\,(s) \qquad (2.10)$$

## 2.2 KONTROL PID

Kontrol *Proposional Integral Derivatif* (*PID*) adalah kontroler yang menggabungkan kontroler *proporsional*, *integral* dan *derivatif* . Kontroler ini dipresentasikan dengan persamaan:

$$m(t) = Kp.e(t) + \frac{Kp}{Ti}\int_0^1 e(t)dt + Kp.Td\ \frac{De(t)}{dt} \qquad (2.18)$$

Keterangan:

$m(t)$ = keluaran

$e(t)$ = masukan

Kp = *Proporsional gain*

Ki = *Integral gain*

Kd = *Derivatif gain*

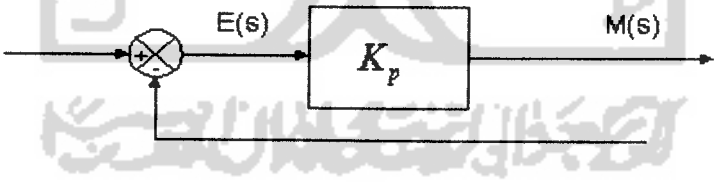
Keberadaan kontroler dalam sebuah sistem kontrol mempunyai kontribusi yang besar terhadap sistem. Pada prinsipnya hal itu disebabkan oleh tidak dapat diubahnya komponen penyusun sistem tersebut. Artinya, karakteristik *plant* harus diterima sebagaimana adanya, sehingga perubahan perilaku sistem hanya dapat dilakukan melalui penambahan suatu sub sistem, yaitu kontroler.

Salah satu tugas komponen kontroler adalah mereduksi sinyal kesalahan, yaitu perbedaan antara sinyal setting dan sinyal aktual. Hal ini sesuai dengan tujuan sistem kontrol adalah mendapatkan sinyal aktual senantiasa (diinginkan) sama dengan sinyal setting. Semakin cepat reaksi sistem mengikuti sinyal aktual dan semakin kecil kesalahan yang terjadi, semakin baiklah kinerja sistem kontrol yang diterapkan.

### 2.2.1 Kontroler *Proporsional*

Kontroler *proporsional* memiliki keluaran yang sebanding/proporsional dengan besarnya *error* sinyal (selisih antara besaran yang diinginkan dengan harga aktualnya)

Secara lebih sederhana dapat dikatakan, bahwa keluaran kontroler proporsional merupakan perkalian antara konstanta *proporsional* dengan masukannya. Perubahan pada sinyal masukan akan segera menyebabkan sistem secara langsung mengubah keluarannya sebesar konstanta pengalinya.

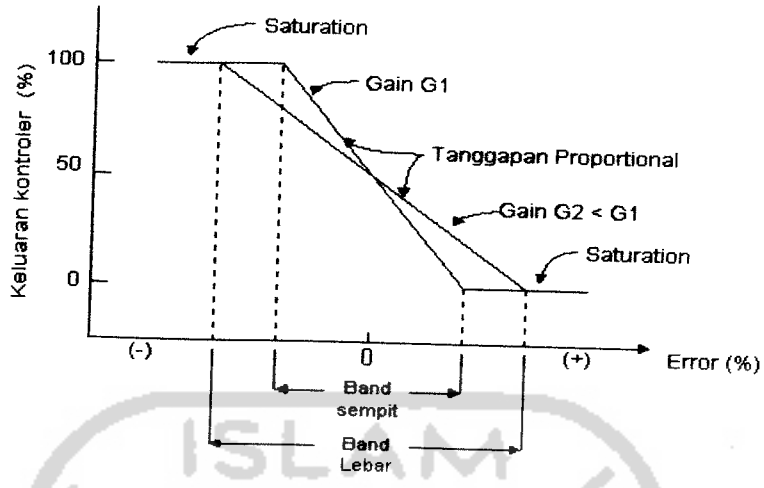



Gambar 2.4 Diagram blok kontroler *proporsional*

E(s) = Masukan

M(s) = Keluaran

Kp = Kontrol *Proporsional*



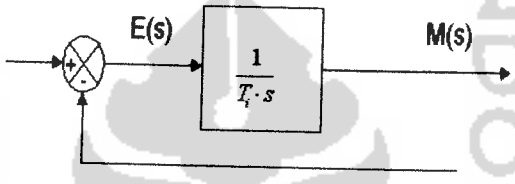

Gambar 2.5 Tanggapan *Proporsional*

Ciri-ciri kontroler proporsional harus diperhatikan ketika kontroler tersebut diterapkan pada suatu sistem. Secara eksperimen, pengguna kontroller *proporsional* harus memperhatikan ketentuan-ketentuan berikut ini:

1. Kalau nilai $K_p$ kecil, kontroler *proporsional* hanya mampu melakukan koreksi kesalahan yang kecil, sehingga akan menghasilkan respon sistem yang lambat.
2. Kalau nilai $K_p$ dinaikkan, respon sistem menunjukkan semakin cepat mencapai keadaan mantabnya.
3. Namun jika nilai $K_p$ diperbesar sehingga mencapai harga yang berlebihan, akan mengakibatkan sistem bekerja tidak stabil, atau respon sistem akan berosilasi

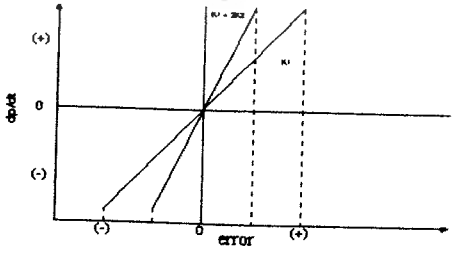
### 2.2.2 Kontroler *Integral*

Kontroler *integral* memiliki karakteristik seperti halnya sebuah integral. Keluaran kontroler sangat dipengaruhi oleh perubahan yang sebanding dengan nilai sinyal kesalahan. Keluaran kontroler ini merupakan jumlahan yang terus menerus dari perubahan masukannya. Kalau sinyal kesalahan tidak mengalami perubahan, keluaran akan menjaga keadaan seperti sebelum terjadinya perubahan masukan.




Gambar 2.6 Blok diagram hubungan antara besaran kesalahan dengan kontroler *integral*

Pengaruh perubahan konstanta integral terhadap keluaran integral ditunjukkan oleh Gambar 2.7. Ketika sinyal kesalahan berlipat ganda, maka nilai laju perubahan keluaran kontroler berubah menjadi dua kali dari semula. Jika nilai konstanta integrator berubah menjadi lebih besar, sinyal kesalahan yang relatif kecil dapat mengakibatkan laju keluaran menjadi besar.



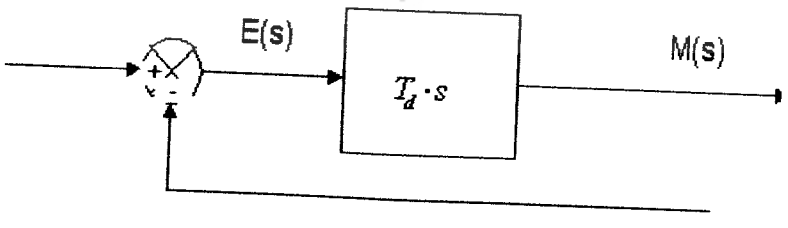

Gambar 2.7 Perubahan keluaran sebagai akibat penguatan dan kesalahan

Ketika digunakan, kontroler integral mempunyai beberapa karakteristik berikut ini:

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

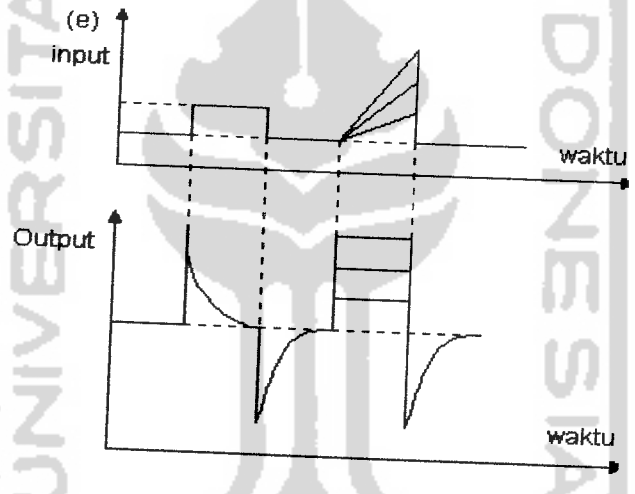
### 2.2.3 Kontroler *Derivatif*

Perubahan yang mendadak pada masukan kontroler, akan mengakibatkan perubahan yang sangat besar dan cepat. Gambar 2.8 menunjukkan blok diagram yang menggambarkan hubungan antara sinyal kesalahan dengan keluaran kontroller.




Gambar 2.8 Blok Diagram kontroler *Derivatif*

Gambar 2.9 menyatakan hubungan antara sinyal masukan dengan sinyal keluaran kontroler *Derivatif*. Ketika masukannya tidak mengalami perubahan, keluaran kontroler juga tidak mengalami perubahan, sedangkan apabila sinyal masukan berubah mendadak dan menaik (berbentuk fungsi *step*), keluaran menghasilkan sinyal berbentuk *impuls*. Jika sinyal masukan berubah naik secara perlahan (fungsi *ramp*), keluarannya justru merupakan fungsi *step* yang besar magnitudnya sangat dipengaruhi oleh kecepatan naik dari fungsi *ramp* dan faktor konstanta *diferensialnya* $T_d$ (Guterus, 1994, 8-4).




Gambar 2.9 Kurva waktu hubungan input-output kontroler *derivatif*

Karakteristik kontroler *derivatif* adalah sebagai berikut:

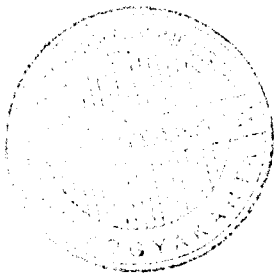
1. Kontroler ini tidak dapat menghasilkan keluaran bila tidak ada perubahan pada masukannya (berupa sinyal kesalahan). Jika sinyal kesalahan berubah terhadap waktu, maka keluaran yang dihasilkan kontroler tergantung pada nilai Td dan laju perubahan sinyal kesalahan. (Powel, 1994, 184).

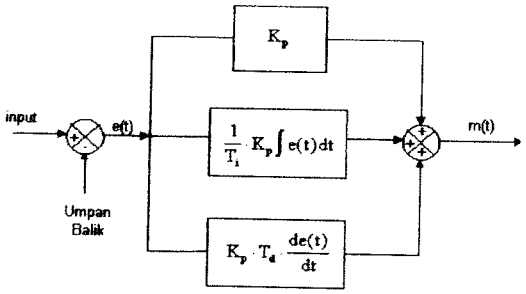
2. Kontroler diferensial mempunyai suatu karakter untuk mendahului, sehingga kontroler ini dapat menghasilkan koreksi yang signifikan sebelum pembangkit kesalahan menjadi sangat besar. Jadi kontroler diferensial dapat mengantisipasi pembangkit kesalahan, memberikan aksi yang bersifat korektif, dan cenderung meningkatkan stabilitas sistem .

Berdasarkan karakteristik kontroler tersebut, kontroler *derivatif* umumnya dipakai untuk mempercepat respon awal suatu sistem, tetapi tidak memperkecil kesalahan pada keadaan tunaknya. Kerja kontrolller *derivatif* hanyalah efektif pada lingkup yang sempit, yaitu pada periode peralihan. Oleh sebab itu kontroler *derivatif* tidak pernah digunakan tanpa ada kontroler lain sebuah sistem.

### 2.2.4 Kontroler PID

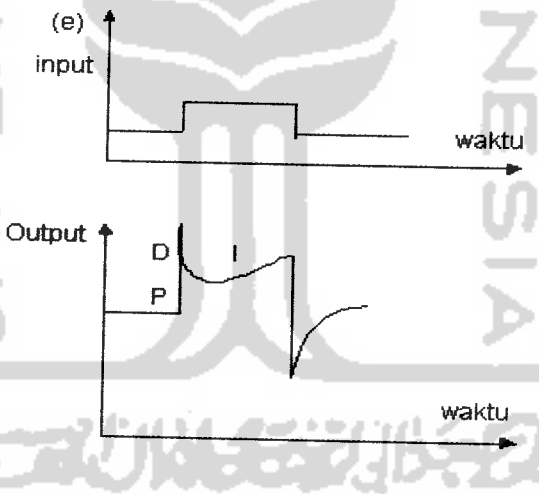
Setiap kekurangan dan kelebihan dari masing-masing kontroler P, I dan D dapat saling menutupi dengan menggabungkan ketiganya secara paralel menjadi kontroler *proposional* plus *integral* plus *derivetif* (kontroller *PID*). Elemen-elemen kontroller *P, I* dan *D* masing-masing secara keseluruhan bertujuan untuk mempercepat reaksi sebuah sistem, menghilangkan *offset* dan menghasilkan perubahan awal yang besar. Gambar 2.9 menunjukkan blok diagram kontroler *PID*.






Gambar 2.10 Blok diagram kontroler *PID*

Keluaran kontroller *PID* merupakan jumlahan dari keluaran kontroler *proporsional*, keluaran kontroler *integral dan Derivatif*. Gambar 2.11 menunjukkan hubungan tersebut.




Gambar 2.11 Hubungan dalam fungsi waktu antara sinyal keluaran dengan masukan untuk kontroller *PID*

Karakteristik kontroler *PID* sangat dipengaruhi oleh kontribusi besar dari ketiga parameter *P, I* dan *D*. Penyetelan konstanta *Kp, Ti*, dan *Td* 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.

## 2.3 Simulink

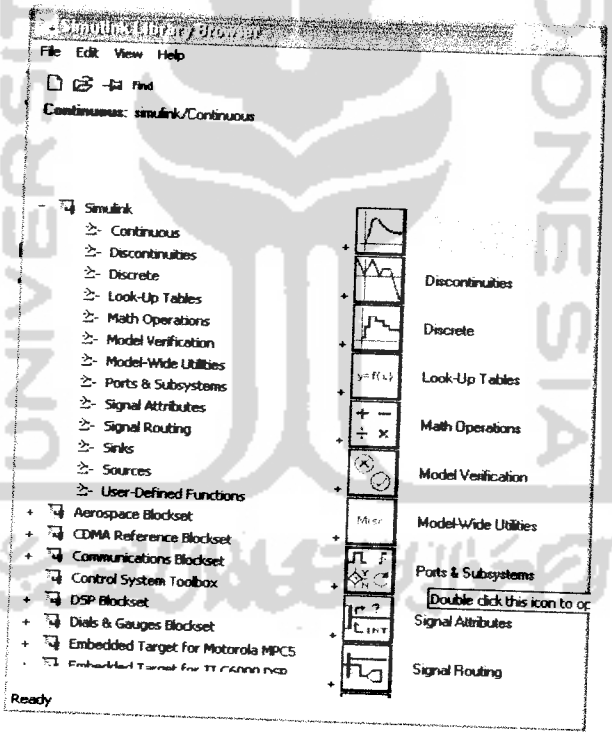
*Simulink* adalah software yang digunakan untuk memodelkan, melakukan simulasi dan analisa terhadap sistem dinamis. *Simulink* dapat digunakan untuk sistem linier maupun non linier, yang dimodelkan dalam waktu kontinyu, waktu cuplik, atau gabungan dari keduanya. Sistem tersebut multirate, yakni mempunyai bagian yang berbeda waktu cupliknya.

Untuk pemodelan, simulink menyediakan *Graphical User Interface* (*GUI*) untuk membangun model sebagai diagram blok, dengan menggunakan operasi mouse *click-and-drag*. Simulink menyediakan blok *library* yang lengkap meliputi *sinks* (keluaran),*sources* (masukan), komponen linier dan nonlinier dan konektor. *Simulink* juga menyediakan fasilitas untuk membuat blok yang didefinisikan sendiri oleh pengguna yakni melalui *s-function.*

Model bersifat hierarkis, sehingga dapat dibangun model seecara top-down maupun bottom-up. Model dapat dilihat sistem pada level tinggi, lalu *double-click* pada blok untuk melihat isi dibawah blok tersebut secara lebih detail. Hal ini memungkinkan untuk melihat bagaimana model itu diorganisasikan dan bagaimana setiap bagiannya saling berinteraksi. Setelah suatu model didefinisikan, dapat dilakukan simulasi terhadap model itu, menggunkan metode

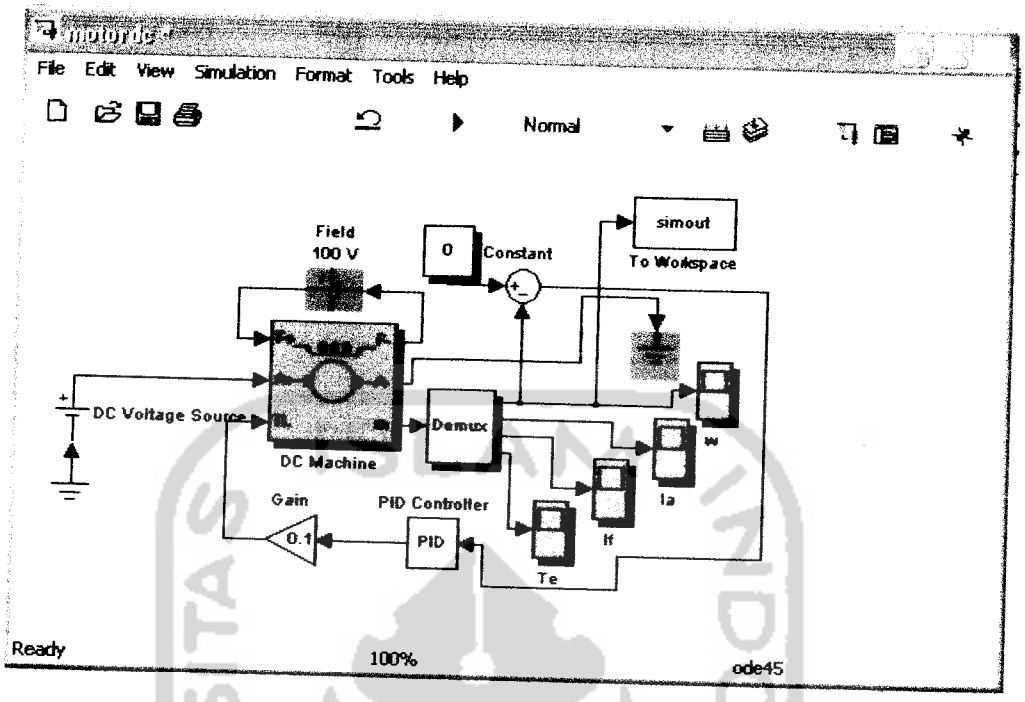
integrasi yang disediakan. Dengan *scope* atau *blok display* lainnya, dapat dilihat hasil simulasi saat simulasi sedang berjalan. Dan parameter dapat diubah dan dilihat pengaruhnya. Hasil simulasi juga dapat diletakkan di *MATLAB* workspace untuk keperluan analisa berikutnya dan visualisasi. Dan karena *MATLAB* dan *SIMULINK* terintegrasi, maka memungkinkan dilakukannya simulasi, analisa dan revisi model pada bagian lingkungan kerja *MATLAB*.

Masuk ke *MATLAB* simulink dengan cara mengklik tombol *simulink* di toolbar atau ketik *simulink* di *command window*, akan terbuka *editor simulink* seperti pada gambar 2.12 berikut:




Gambar 2.12 Jendela *simulink library browser*

Untuk membuat model caranya pilih *file*, *new*, *model*, kemudian kan terbuka sebuah editor untuk menempatkan blok-blok model seperti pada gambar 2.13 :



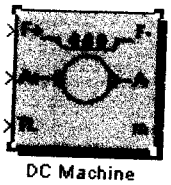

Gambar 2.13 Tampilan *editor simulink* untuk membuat suatu model

Blok-blok simulink yang digunakan adalah sebagai berikut:

1. *ground*: blok koneksi ke ground

2. *DC voltage source*: blok sumber tegangan *DC* ideal

3. *DC Machine*: blok mesin eksitasi terpisah yang dapat digunakan sebagai mesinDC hubungan shunt atau hubungan seri .



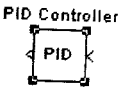

4. *Constant*: blok yang berisi nilai konstanta yang dapat ditentukan sendiri oleh user.


5. *sum*: blok penjumlahan atau pengurangan input sesuai denagn tanda operasinya, dan bias mempunyai lebih dari satu input.

6. *Demux*: blok ini memisahkan satu sinyal input menjadi beberapa sinyal output secara terpisah.

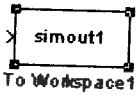

7. *Scope*: blok yang menampilkan keluaran dari inputnya yang dapat diamati oleh user.

8. *PID*: blok kontrol *PID* berisi ukuran dari *P, I, D*



9. *Gain*: blok penguat yang memiliki nilai keluaran tetap.

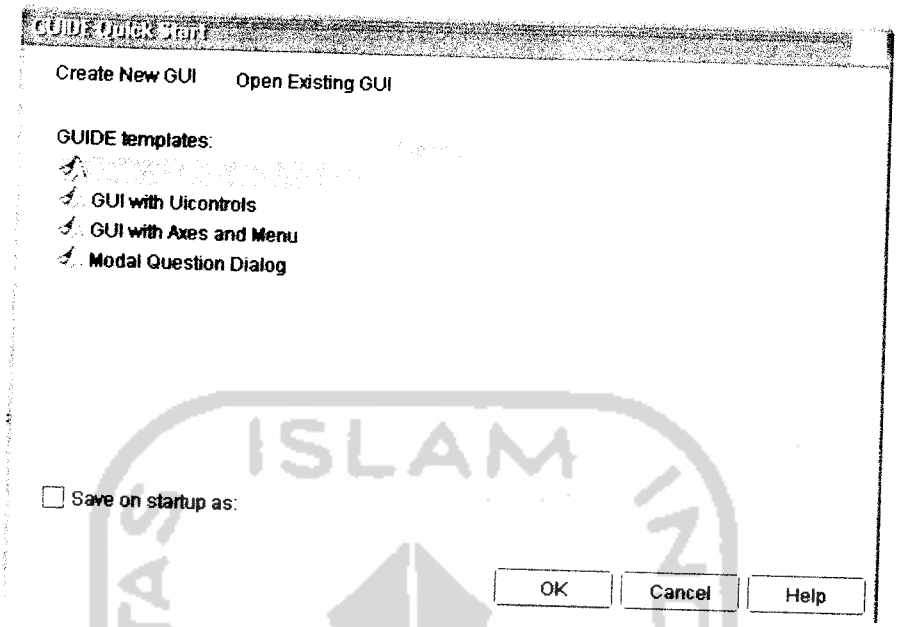

10. *To Workspace*: intuk mengirim data sinyal ke *workspace* di *command window*




## 2.4 Graphical User Interface (GUI)

*GUI (Graphical User Interface)* adalah sebuah tool yang disediakan didalam *MATLAB*. *GUI* adalah sebuah alat penghubung untuk membuat sebuah objek grafis / gambar didalam *MATLAB*, seperti tombol, bidang teks, sliders, menu dan lain sebagainya. Penggunaan *GUI* untuk mengontrol suatu sistem lebih mudah dipelajari dan digunakan, karena penggunanya tidak perlu mengetahui baris perintah untuk mengatur/mengendalikan suatu sistem.

*MATLAB* menyediakan tool untuk pembuatan *GUI*. Fasilitas ini dinamakan *GUIDE* (*GUI Development Environment*). Untuk pembuatan *GUI*, ketikkan perintah *guide* di *command window*. Perintah ini akan menampilkan *GUIDE Quick Start* sebagai berikut:

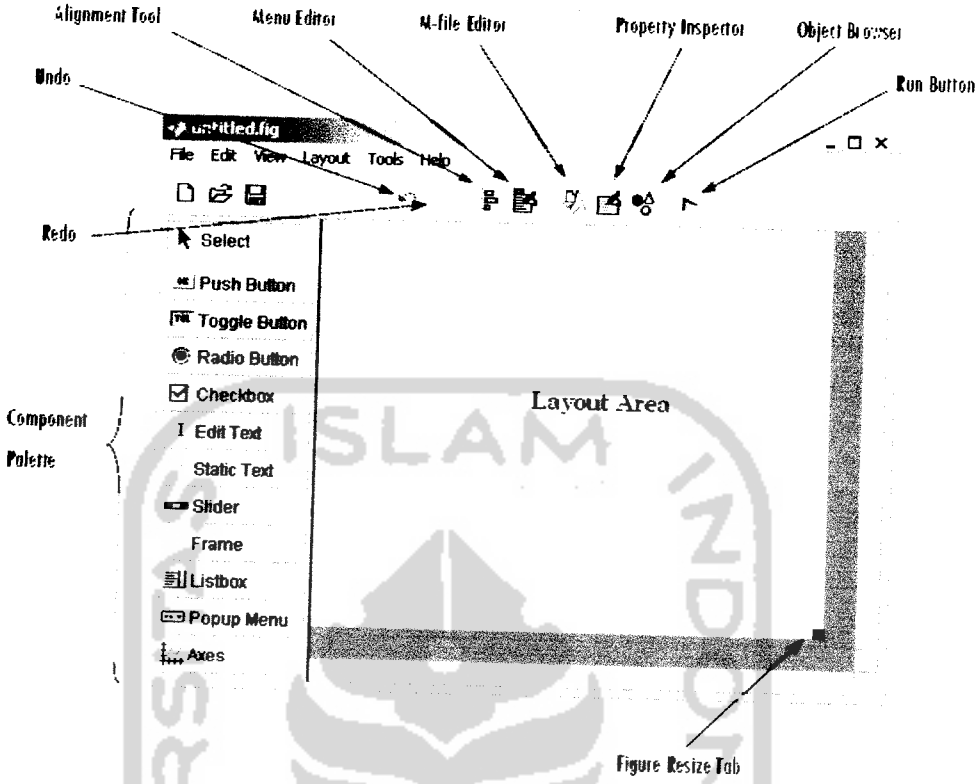



Gambar 2.14 Tampilan *guide quick start*

Dari *GUIDE quick Start* dialog dapat dilakukan:

1. Membuat *GUI* baru dari *GUIDE templates*. *GUI* yang sudah tersedia yang dapat dimodifikasi sesuai dengan keperluan dan dapat dipilih melalui menu *create new GUI.*
2. Membuka *GUI* yang ada atau *GUI* yang pernah dibuat dapat dibuka kembali melalui *open existing GUI.*

Jika sudah memilih salah satu, klik '*OK*' akan membuka *GUI* di *layout Editor.* Pada layout area dapat diletakkan panel-panel yang sudah disediakan pada *component palette* dengan cara *drag and drop.*




Gambar 2.15 Jendela *Toolset*

*GUIDE Toolset*:

1. *Layout Editor* : Tempat untuk menambahdan menyusun objek di jendela GUI
2. *Alignment Tools*: Untuk mengatur posisi masing-masing objek.
3. *Component Pallet*: Komponen yang disediakan di dalam *GUI.*
4. *Property Inspector*: memeriksa dan mengeset nilai-nilai (sifat) objek.
5. *Objek Browser*: Memantau hirarki objek-objek yang telah ada di layout.
6. *Menu editor*: Membuat menu dan submenu dari *GUI.*

Komponen *Palatte GUI*:

1. *Push Button*

   *Push Button* menghasilkan suatu tindakan ketika diklik. Sebagai contoh suatu tombol *OK* mungkin akan menutup suatu dialog kotak ketika tombol *OK* ditekan, tombol akan nampak ditekan dan *callback*nya akan melaksanakan perintah.

2. *Toggle Button.*

   Tombol *Toggle Button* akan menghasilkan suatu tindakan ketika diklik suatu tombol kotak akan nampak tertekan, dan ketika diklik lagi akan melepaskan kotak kembali normal dan masing-masing mempunyai *callback.*

3. *Radio Button*

   Fungsi komponen *Radio Button* sama seperti *toggle button.* Jika komponen *radio button* diberi tanda maka nilai akan diset 1, sebaliknya jika tidak diberi tanda maka akan bernilai 0.

4. *Check boxes*

   Sama seperti komponen *radio button*

5. *Edit text*

   Dalam penggunaanya, sebagian besar komponen *edit text* banyak difungsikan sebagai komponen untuk menginputkan suatu data.

6. *Slider*

   Komponen ini berfiungsi memberikan input angka dengan suatu range tertentu. *Property MAX-MIN* menunjukkan range yang digunakan oleh komponen *slider.*

7. *List Boxes*

   *List Boxes* lebih banyak digunakan untuk menampilkan suatu indeks data. Tiap baris dalam list boxes memiliki nilai dalam *property value*, sehingga kita dapat mengakses variable didalam *list boxes* menggunakan perintah *get*. Untuk menentukan range baris, diset menggunakan *MAX-MIN*.

8. *Menu Pop-up*

   Menu *pop-up callback* dapat diprogram dengan mengambil indeks dari nilai yang dipilih. Callback ini akan memeriksa indeks dari item yang dipilih dan men*switch* sesuai dengan nilai yang diperoleh.

9. *Axes*

   Axes adalah komponen yang berfungsi sebagai interface penampil grafik, baik berupa koordinat maupun gambar.

Dalam membuat *GUI*, proses yang dilakukan adalah:

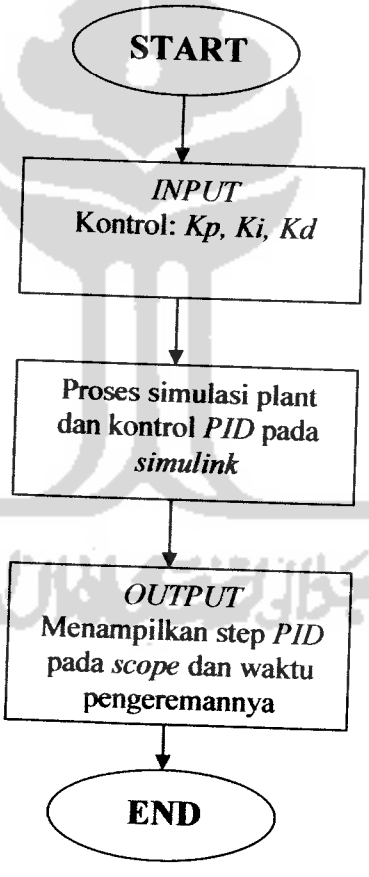
1. Perancangan tampilan *GUI*, akan lebih baik jika tampilan *GUI* digambar terlebih dahulu dalam kertas, penempatan dan pengatuan objek seperti *pushbutton, axes* dan lain-lain.
2. Pemrograman *GUI*, membuat program bagi objek-objek *GUI* di *M-file*.

   Ketika membuat layout *GUI* dan menyimpannya, maka *MATLAB* akan membuat *M-file* secara otomatis yang nantinya akan digunakan untuk pemrograman. Nama *M-file* harus sama dengan nama Figur dari *GUI* tersebut.

# BAB III

# PERANCANGAN SISTEM

Dalam bab III akan dibahas mengenai perancangan sistem pengendali motor dc dengan *PID MATLAB* dan disimulasikan dengan menggunakan *GUI MATLAB* dan program yang saling menghubungkan antara sistem tersebut dengan ***simulink*** sehingga keduanya saling berinteraksi berdasarkan teori-teori yang telah dibahas pada bab sebelumnya.




Gambar 3.1 *Flowchart* jalannya program *GUI*

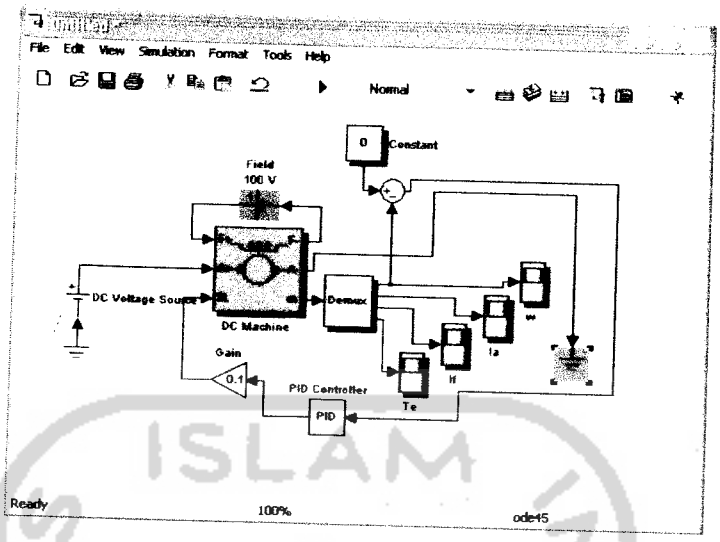
### 3.1 Perancangan Sistem pada Simulink

Membuat suatu system pengendali *PID* yang mengendalikan pengereman suatu motor DC pada *simulink* dimulai dengan membuka software *Matlab* kemudian pilih *file*, *new model*, setelah itu model windows akan terbuka untuk menempatkan blok-blok model.

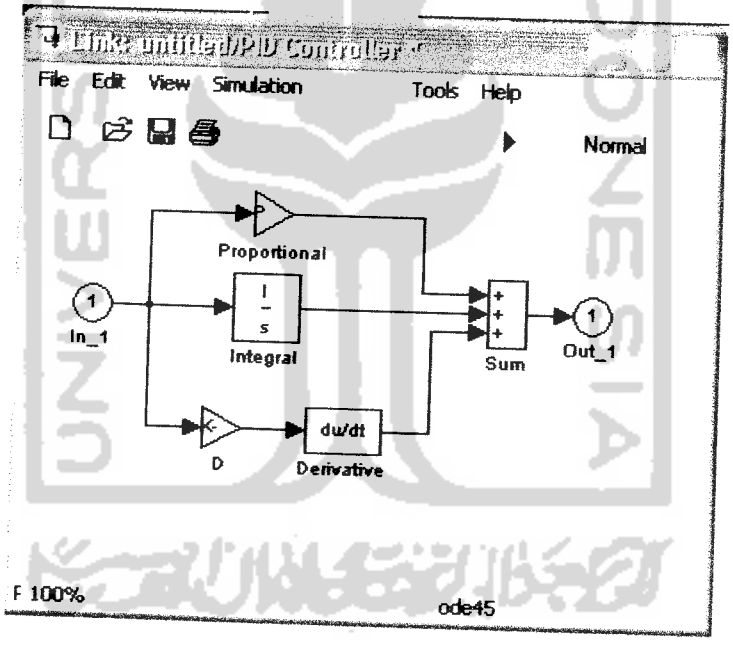
Untuk membuat sebuah system, perlu men*drag* blok model yang dibutuhkan dari *Simulink Blocks Library*. Blok yang dibutuhkan adalah:

1. *DC Voltage Source* sebagai input.
2. *Ground* sebagai pentanahan *DC Voltage* dan *motor dc*.
3. *DC machine*
4. *Demux* sebagai pengkombinasi satu input menjadi 4 output.
5. *Constant*
6. *PID Controller* sebagai kontroller
7. *Gain*
8. *Scope* sebagai tampilan output.

Blok-blok tersebut dapat diperoleh dari *Library Browser*. Untuk mendapatkan blok-blok yang belum diketahui tempatnya, maka dapat dicari melalui kotak *search* atau menu *find*. Setelah blok-bloknya ditemukan maka blok tersebut di*drag* dari *Simulink Library Browser* ke *model window* kemudian dihubungkan, setelah itu akan diperoleh gambar sebagai berikut:

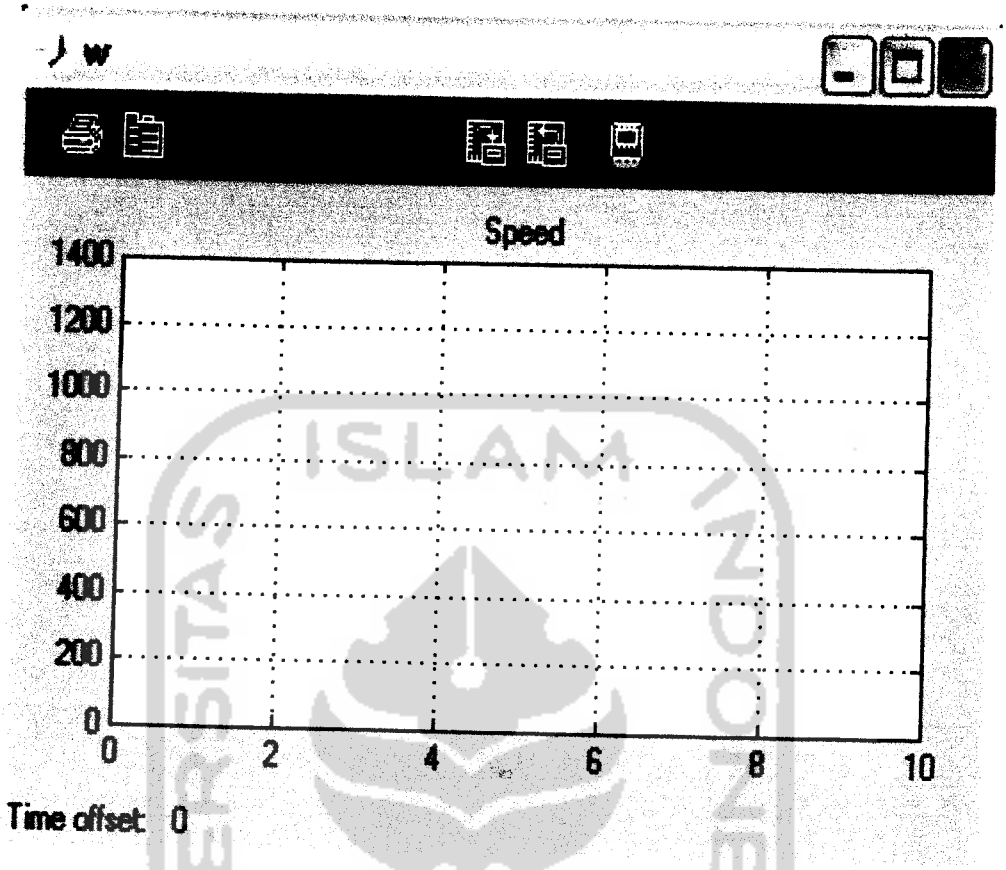



Gambar 3.2 Motor dc pada *simulink*




Gambar 3.3 Subsistem PID Controller

Setelah semua konektor dihubungkan, maka lakukan simulasi dengan cara menekan tombol *Run* dan melihat hasilnya dapat dilihat pada blok *Scope* seperti pada gambar berikut:

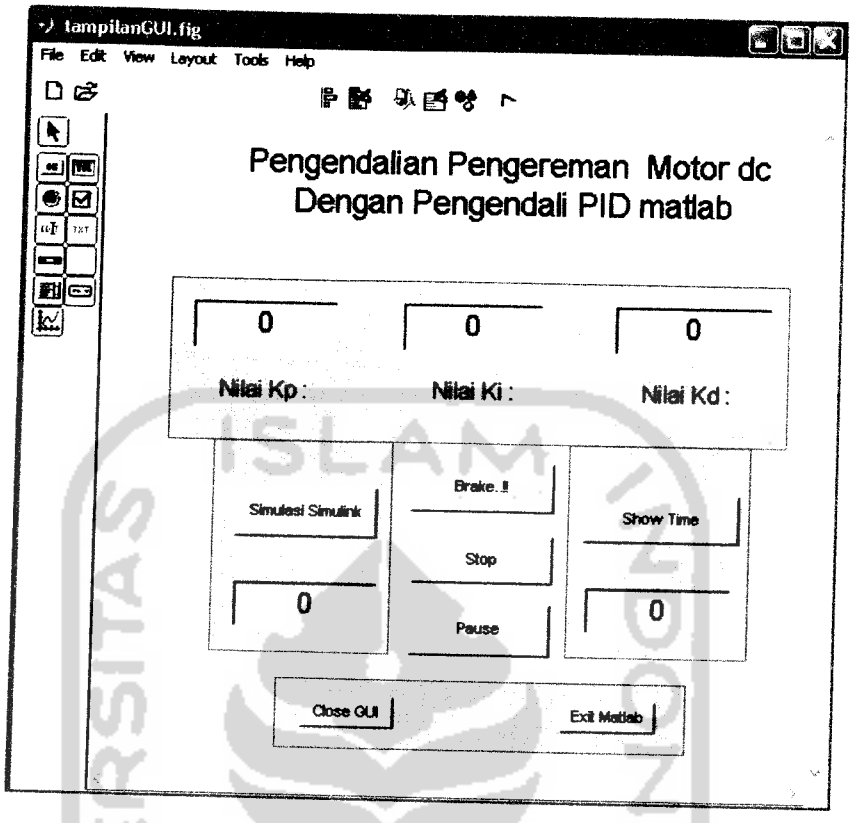



Gambar 3.4 Grafik Tampilan *scope*

### 3.2 Perancangan GUI Matlab

Komponen-komponen yang ada pada *GUI* dapat mengatur parameter *PID*, memasukan nilai input dan step input serta menjalankan simulasi pada *simulink*.

Pada pengaturan *PID* yang terdiri dari *Kp* sebagai *Proporsional*, *Ki* sebagai *Integral*, dan *Kd* sebagai *Derivatif* atau turunan dapat dilakukan dengan cara Mengetikkan besarnya nilai *Kp, Ki, Kd* yang diinginkan pada *edit text*.




Gambar 3.5 Desain Kontrol *PID* pada *GUI*

Komponen-komponen *GUI* yang digunakan adalah sebagai berikut:

1. *Push button* sebanyak 7: *Simulasi Simulink*, *Brake..!!*, *Stop, Pause, Show Time, close GUI, Exit Matlab.*
2. *Edit text* sebanyak 5: n_edit, waktu_rem, kp_edit, ki_edit, kd_edit,
3. *Group panel* sebanyak 4: *PID Controller*, Kecepatan Simulasi, Waktu Penegereman, *Exit.*

Untuk mengeset *property* dan *tag* dari tiap-tiap komponen, pilih *Property Inspector* dari menu *view* atau klik dua kali pada komponen tersebut.

1. *Static Text*

   Klik pada komponen *Static Text* , kemudian isikan *String Property* sesuai dengan nama yang diinginkan dan pada *Tag Property*nya diabaikan saja karena komponen ini tidak menghasilkan *callback* pada *M-file.*

2. *Edit Text*

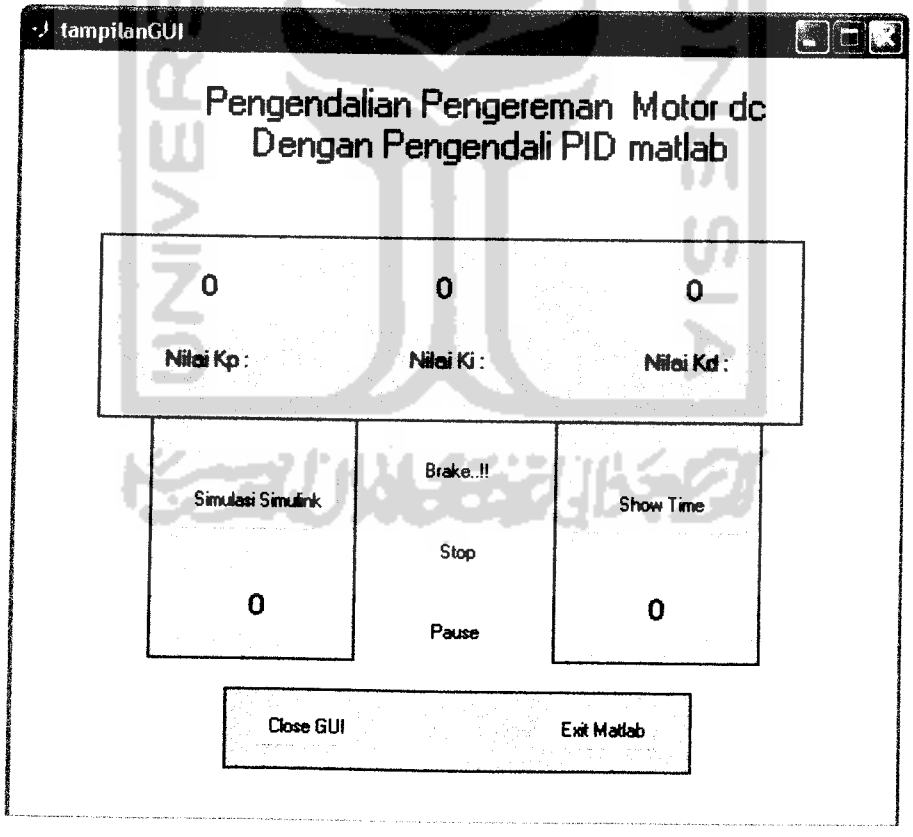
   Pada *PID Controller* ada tiga *Edit Text, Kp_Edit String*nya diisi "-0.004", *Ki_Edit String*nya diisi "-0.0004", *Kd_Edit String*nya diisi "-0.0005". Kemudian pada *TAG* Kp diisi "*kp_edit*", Ki diisi "*ki_edit*", Kd diisi "*kd*_edit". Pada *Group Panel Kecepatan Simulasi* ada satu *Edit Text,* kecepatan *string*nya diisi "100" dan *TAG* nya diisi dengan *n_edit.* Dan pada group panel *Waktu Pengereman* ada satu *Edit Text,* pengereman *string*nya diisi dengan "0", dan *TAG*nya diisi dengan "waktu_rem".

3. *Push Button*

   *Push Button* dari *Group panel kecepatan Simulasi* ada satu, pada *string*nya diisi "*start simulink*" dan *TAG*nya diisi dengan "*mulai*". Kemudian pada group panel *pengereman* ada 4 *Push Button, string*nya diisi "*Brake..!!*", kemudian *TAG*nya diisi "*rem*". Stringnya diisi *stop TAG*nya diisi dengan *Stop_simulasi,* stringnya diisi dengan *pause TAG*nya diisi dengan *paused.* Stringnya diisi dengan *Show Time TAG*nya diisi dengan *show*.Kemudian pada *Group Panel Exit* ada dua *Push Button* yaitu yang pertama *string*nya diisi "*Close GUI*", *TAG*nya diisi "tutup". Pada *Push Button* yang kedua, *string*nya diisi "*Exit Matlab*", *TAG*nya diisi dengan "*keluar*".

*Tag property* berfungsi untuk mendiskripsikan nama fungsi yang akan dipanggil. Setelah komponen selesai diatur nilainya kemudian disimpan dan diaktifkan dengan memilih *run*, dengan cara klik icon *run* atau pilih *tool* kemudian *run* hal ini akan menghasilkan dua file, yaitu *file * fig (file figure)* dan *file * m (untuk pemrograman GUI)* biasa disebut dengan *M-file*. Komponen-komponen yang harus diprogram adalah *Edit Text* dan *Push Button*. Program lengkap dari komponen-komponen GUI Matlab terlampir pada lampiran.

Setelah melakukan pemrograman, lakukan pengujian GUI dengan cara mengaktifkan GUI. Pengaktifan GUI melalui M-file dapat dilakukan dengan cara klik icon *run* atau tekan F5 sehingga diperoleh gambar sebagai berikut:




Gambar 3.6 Pengaktifan *GUI*

# BAB IV

# ANALISIS DAN PEMBAHASAN

Pada bab ini akan dilakukan pengujian sistem. Pengujian dilakukan pada kecepatan 100, 200, 300, 400, 500, 600, 700, 800, 1000, 1200, 1500, 1600, 1700. Parameter-parameter PID adalah sebagai berikut:

P = -0.004

I = -0.0004

D = -0.0005

Ketiga parameter tersebut diperoleh dari mencoba-coba mengubah masing-masing parameter tersebut sampai mendapatkan angka-angka yang mewakili karakteristik sinyal yang diinginkan (dalam hal ini mengerem), sebagaimana tertulis diatas.

Kemudian akan dilakukan pula pengujian sistem dengan berbagai variasi nilai PID dan akan terlihat efek dari masing-masing pengndali dan kombinasi dari pengndali-pengendali tersebut

*Step response* yang diamati adalah waktu pengereman, yaitu ketika sampai pada kecepatan tertentu yang diinginkan, maka tegangannya dilepas (diberi nilai "0"), setelah itu baru dimasukkan parameter-parameter *PID*nya untuk mengendalikan putaran sisa motor dc tersebut.

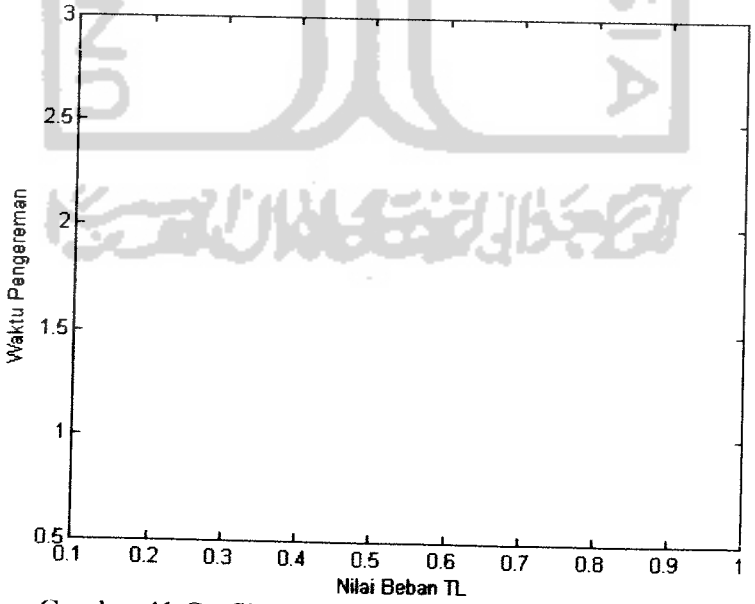
Pada pengereman sistem ini, pengendali *PID* mengendalikan *Torsi Load (TL)* yang telah diberi *Gain* sebagai beban dengan nilai "0.1" seperti pada gambar 2.12. Semakin besar nilai *gain* maka semakin kecil *torsi load*nya.

Sebelum masuk ke percobaan, Berikut akan dilihat hubungan variasi Torsi Load dengan waktu pengereman. Pada percobaan ini digunakan pengendali PID dengan nilai seperti yang tertera diatas. Berikut ini table perbandingannya:

Tabel 4.1 Tabel perbandingan Torsi Load dengan bebannya

| Gain Torsi Load | Waktu Pengereman |
|---|---|
| 0.1 | 2.827 |
| 0.2 | 1.672 |
| 0.3 | 1.219 |
| 0.4 | 0.979 |
| 0.5 | 0.828 |
| 0.6 | 0.725 |
| 0.7 | 0.653 |
| 0.8 | 0.613 |
| 0.9 | 0.549 |
| 1 | 0.515 |

Hubungan nilai beban TL terhadap waktu dapat dilihat pada grafik berikut ini:



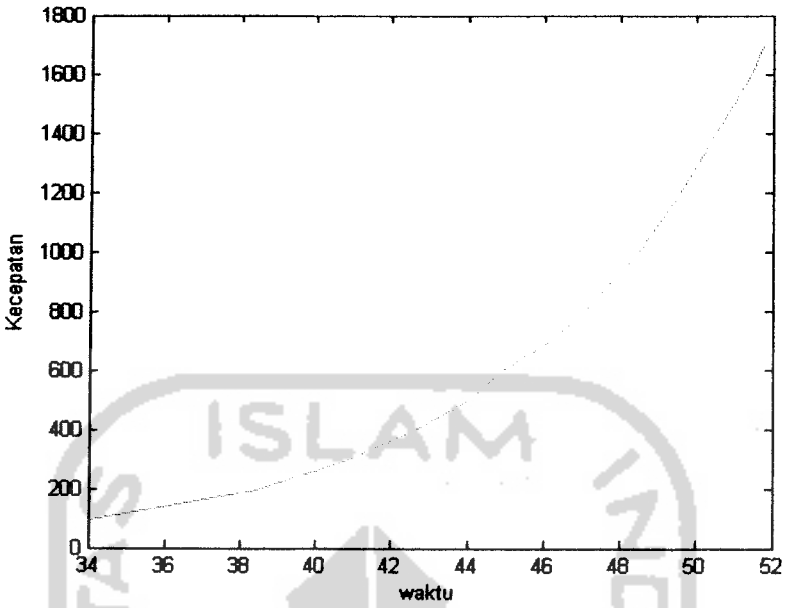

Gambar 41 Garfik Beban TL terhadap Waktu Pengereman

Untuk akurasi hasil percobaan ini, maka dilakukan 4 kali percobaan untuk setiap kecepatan pada masing-masing pengendali. Sehingga didapatkan waktu pengereman rata-rata dari setiap kecepatan pada masing-masing pengendali tersebut. Maka pada uraian dibawah ini, yang digunakan adalah waktu pengereman rata-ratanya.

Tabel 4.2 Pengereman tanpa PID

| Kecepatan | Percobaan 1 | Percobaan 2 | Percobaan 3 | Percobaan 4 | RATA-RATA |
|---|---|---|---|---|---|
| 100 | 34,073 | 34,436 | 34,04 | 33,707 | 34,064 |
| 200 | 38,699 | 38,136 | 38,485 | 38,445 | 38,44125 |
| 300 | 40,987 | 40,924 | 40,736 | 40,813 | 40,865 |
| 400 | 42,612 | 42,609 | 42,703 | 42,597 | 42,63025 |
| 500 | 44,002 | 44,053 | 44,075 | 44,09 | 44,055 |
| 600 | 45,169 | 45,207 | 44,172 | 45,193 | 44,93525 |
| 700 | 46,248 | 46,182 | 46,191 | 46,244 | 46,21625 |
| 800 | 46,991 | 46,996 | 47,137 | 47,036 | 47,04 |
| 1000 | 48,476 | 48,484 | 48,498 | 48,478 | 48,484 |
| 1200 | 49,506 | 49,599 | 49,618 | 49,537 | 49,565 |
| 1500 | 50,908 | 51,023 | 51,047 | 51,019 | 50,99925 |
| 1600 | 51,429 | 51,417 | 51,431 | 51,433 | 51,4275 |
| 1700 | 51,809 | 51,689 | 51,815 | 51,748 | 51,76525 |

Berikut ini akan digambarkan grafik hubungan antara kecepatan dan waktu pengereman pengereman tanpa pengendali *PID*:



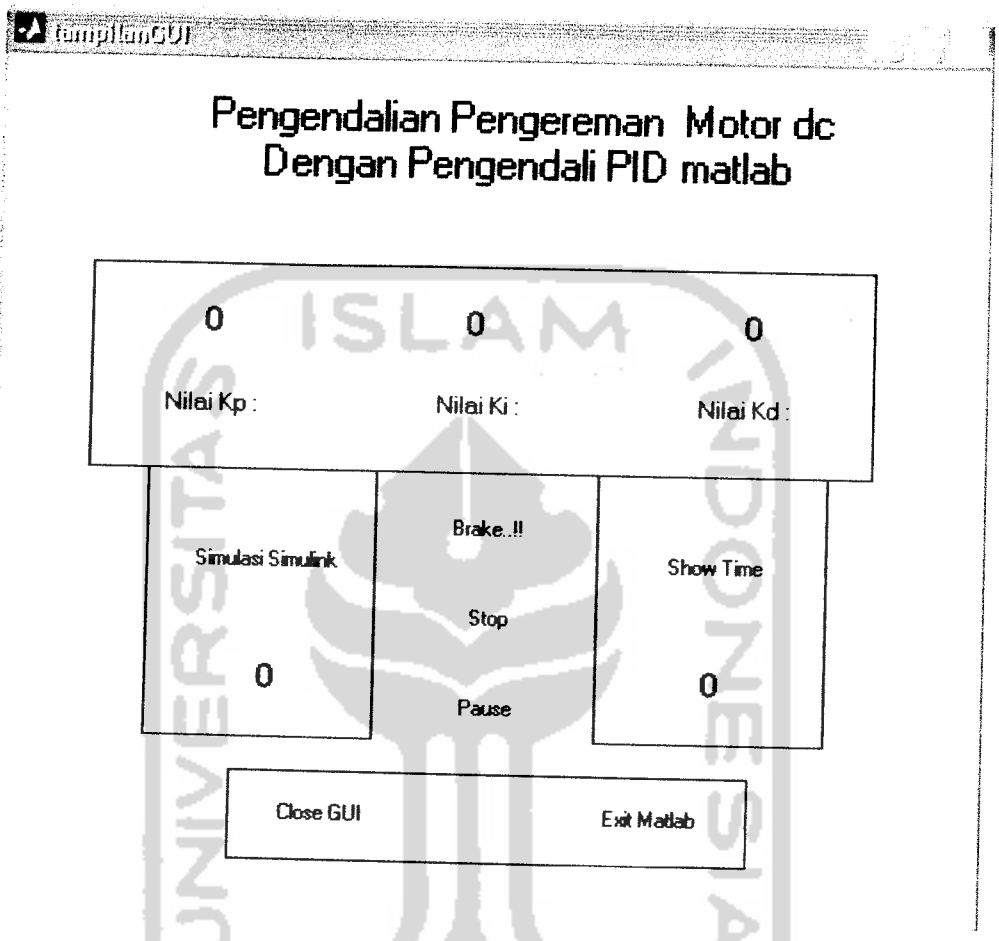

Gambar 4.2 Garfik Kecepatan tehadap waktu pengereman tanpa PID

Pengereman sistem ini dengan tanpa pengendali PID, tidak dapat menghentikan motor dc ini, sehingga dibatasi sampai 0.5 sampletime. Dari table dan grafik diatas dapat dilihat bahwa semakin besar kecepatan yang ditempuh, maka semakin lama waktu pengereman yang dibutuhkan.

### 4.1 Pengujian pengereman sistem dengan pengendali *Proporsional.*

Langkah pertama yang dilakukan untuk menganalisis adalah memasukkan nilai kecepatan yang diinginkan pada tampilan *GUI.* Kemudian tekan tombol "*Similasi Simulink*". Setelah *step respon* mencapai *steady state* pada kecepatan yang diinginkan, maka lakukan pengendalian dengan menggunakan pengendali PID untuk mendapatkan hasil *step respon* pengereman yang diinginkan dengan menekan tombol "*Brake..!!*" setelah sebelumnya mengisi parameter pengendali

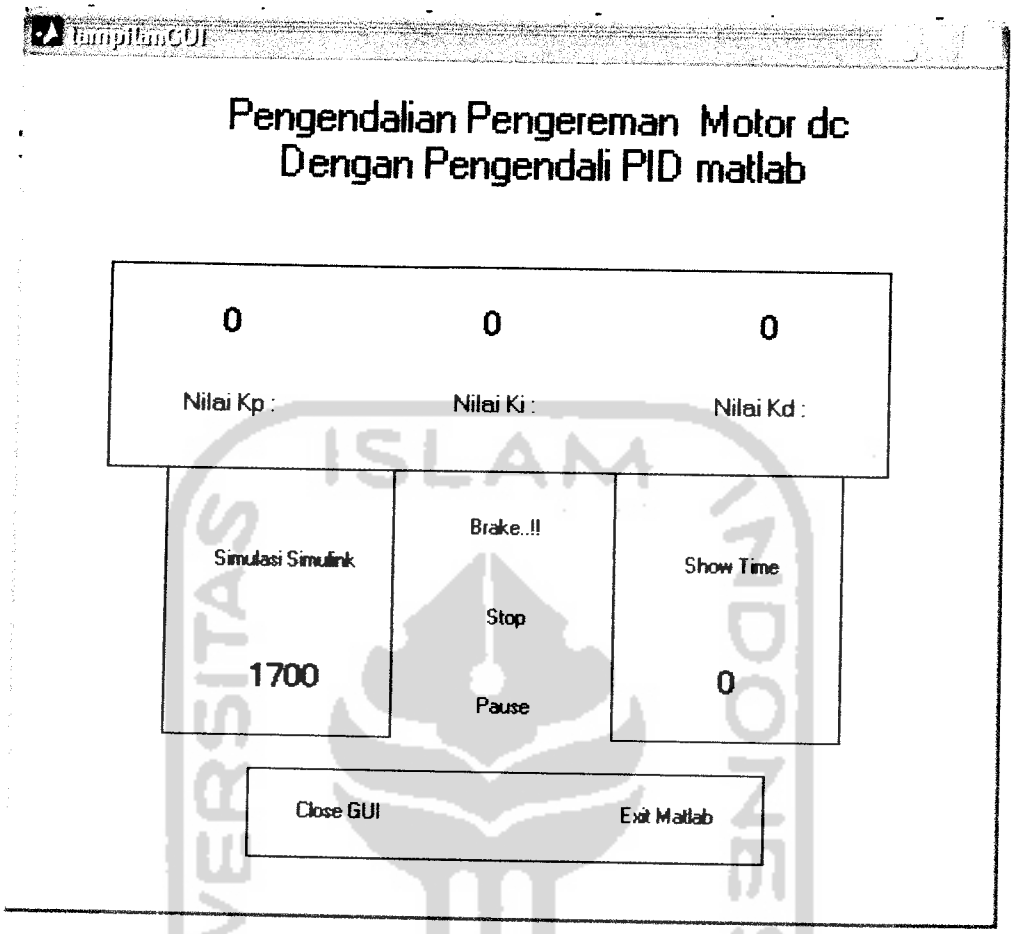
PIDnya. Maka tampilan GUI akan menampilkan secara otomatis waktu pengeremannya.



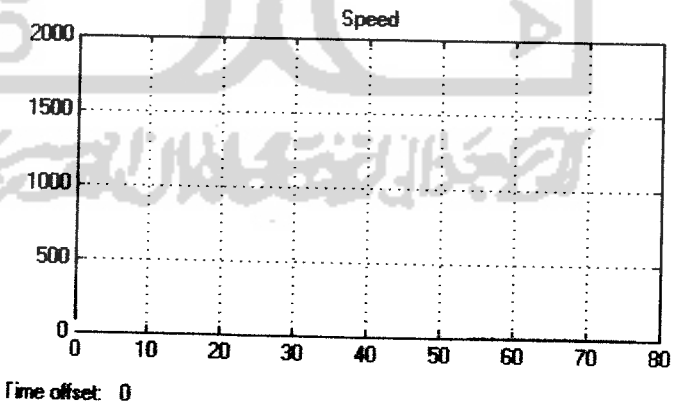

Gambar 4.3 Tampilan GUI

4.1.1 Percobaan 1 tanpa pengendali PID dengan kecepatan 1700

Masukkan nilai kecepatan dengan nilai "1700", kosongkan nilai PID (di "0" kan), kemudian tekan tombol "*Start Simulation*". Setelah *Steady State* pada kecepatan 1700, maka tekan tombol "*Brake..!!*"




Gambar 4.4 Tampilan GUI dengan Kecepatan 1700

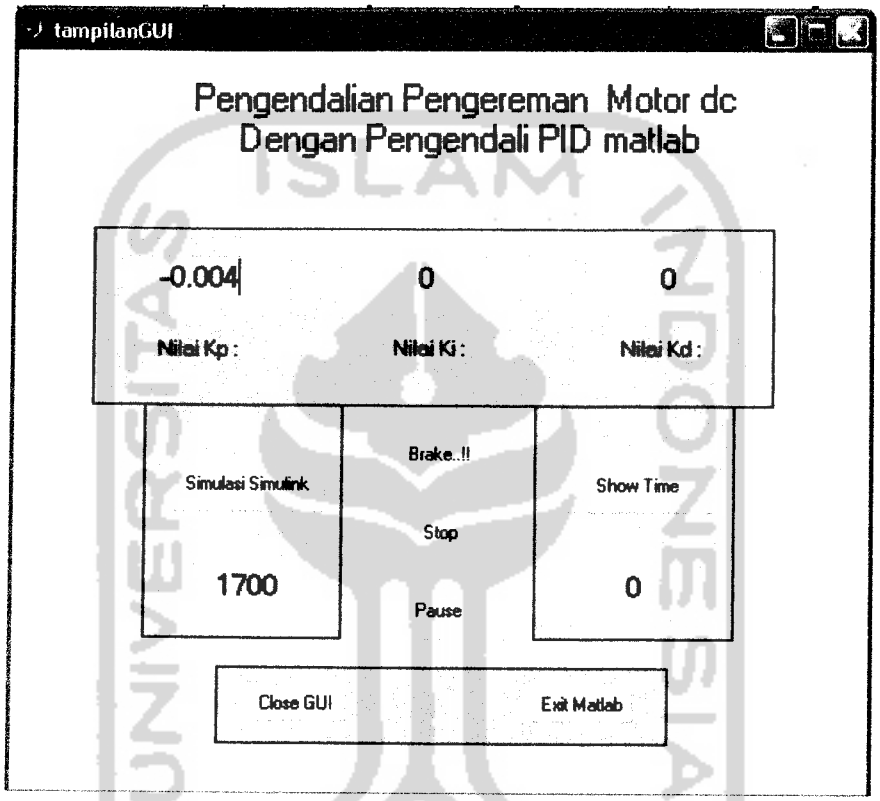



Gambar 4.5 Grafik Pengereman tanpa PID

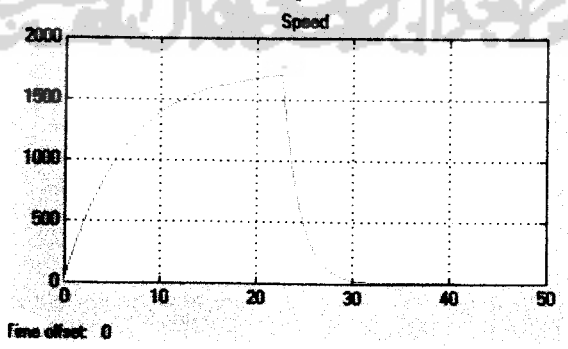
pada kecepatan 1700

4.1.2 Percobaan 2 Pengereman dengan Pengandali *Proporsional (P)* pada kecepatan 1700

Kemudian dengan langkah yang sama, masukkan nilai *Proporsional* pada kecepatan 1700, sehingga diperoleh hasil sebagai berikut:




Gambar 4.6 Tampilan GUI dengan Pengendali *Proposional* pada Kecepatan 1700




Gambar 4.7 Grafik respon dari pengendali P pada kecepatan 1700

Tabel 4.3 Hasil percobaan dengan *Proporsional*

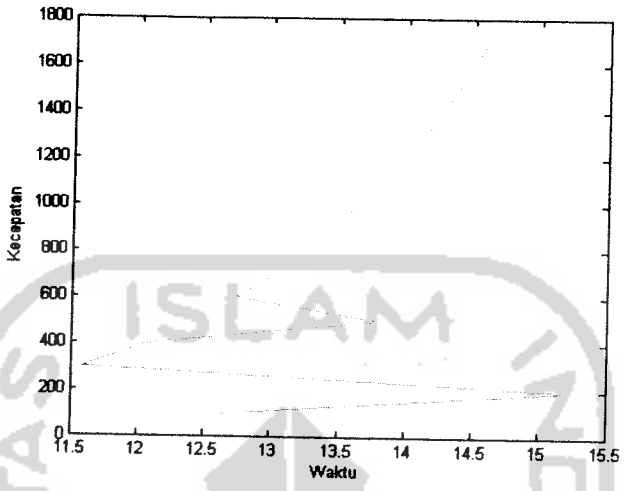
| P | I | D | Kecepatan | Waktu Pengereman | |
|---|---|---|---|---|---|
| | | | | Tanpa PID | Proporsional |
| -0.004 | 0 | 0 | 1700 | 51,76525 | 12.61875 |

Dari percobaan diatas, pengereman dengan menggunakan pengendali *Proporsional* saja tidak dapat menghentikan putaran motor. Sehingga data yang diambil hanya dibatasi pada kecepatan sisa putaran motor = 0.5 kecuali pada pengereman dengan kecepatan 100 & 200 yang dibatasi dengan kecepatan 0.1. berikut table selengkapnya:

Tabel 4.4 Pengereman dengan pengendali *Proporsional* (nilainya = -0.004)

| KECEPATAN | Percobaan 1 | Percobaan 2 | Percobaan 3 | Percobaan 4 | RATA-RATA |
|---|---|---|---|---|---|
| 100 | 12,803 | 12,496 | 12,65 | 12,526 | 12,61875 |
| 200 | 15,173 | 15,226 | 15,231 | 15,233 | 15,21575 |
| 300 | 11,555 | 11,725 | 11,491 | 11,594 | 11,59125 |
| 400 | 12,089 | 12,112 | 12,051 | 12,066 | 12,0795 |
| 500 | 13,828 | 13,817 | 13,753 | 13,855 | 13,81325 |
| 600 | 12,733 | 12,74 | 12,746 | 12,739 | 12,7395 |
| 700 | 13,028 | 13,032 | 13,038 | 13,051 | 13,03725 |
| 800 | 13,255 | 13,252 | 13,253 | 13,252 | 13,253 |
| 1000 | 13,627 | 13,649 | 13,65 | 13,646 | 13,643 |
| 1200 | 14,016 | 13,997 | 14,002 | 14,032 | 14,01175 |
| 1500 | 14,383 | 14,38 | 14,383 | 14,387 | 14,38325 |
| 1600 | 14,49 | 14,488 | 14,493 | 14,489 | 14,49 |
| 1700 | 14,6 | 14,617 | 14,599 | 14,597 | 14,60325 |

Berikut grafik hubungan antara Kecepatan dengan waktu pengeremannya dengan pengendali *Peroposional :*



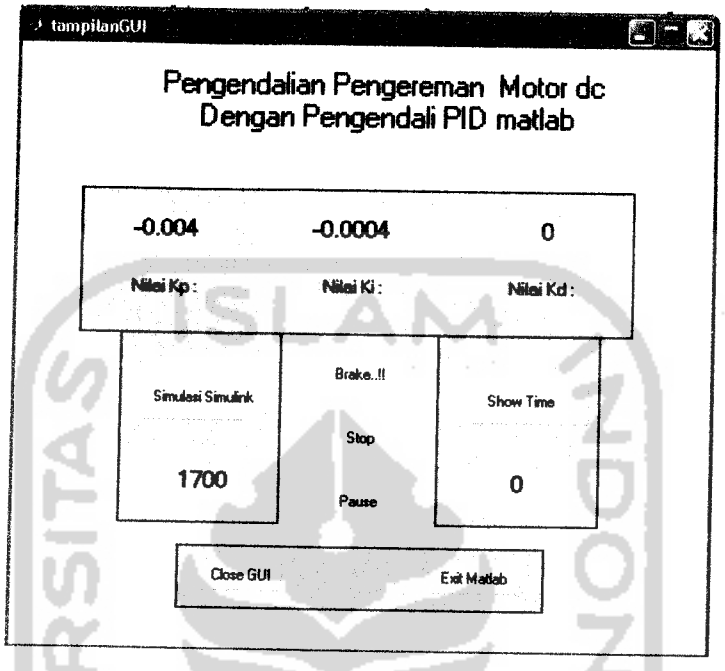

Gambar 4.8 Garfik Hubungan Kecepatan terhadap waktu pengereman dengan pengendali Proporsional

## 4.2 Pengujian pengereman sistem dengan pengendali *Proporsional + Integral*

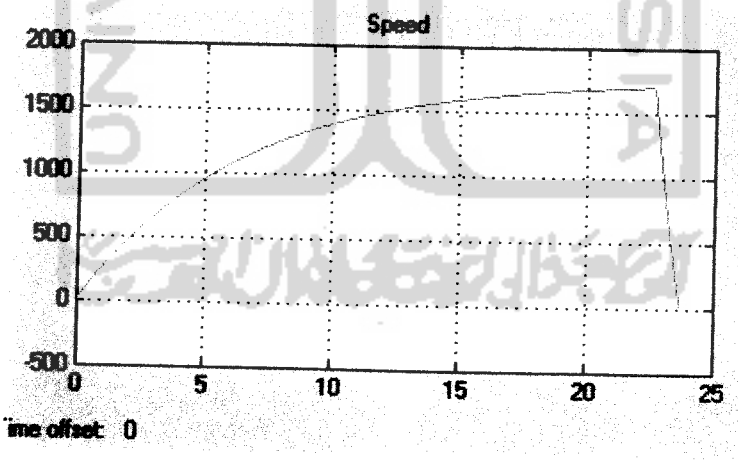
Untuk mulai menganalisis, kita masukkan nilai kecepatan yang diinginkan pada tampilan *GUI.* Kemudian tekan tombol "*Start Simulation*". Setelah *step respon* mencapai *steady state* pada kecepatan yang diinginkan , maka lakukan pengendalian dengan menggunakan pengendali *PID* untuk mendapatkan hasil *step respon* pengereman yang diinginkan dengan menekan tombol "*Brake..!!*" setelah sebelumnya mengisi parameter pengendali *PID*nya. Maka tampilan *GUI* akan menampilkan secara otomatis waktu pengeremannya. Lihat gambar 4.1.

### 4.2.1 Percobaan 2 Pengereman dengan Pengandali *Proporsional+Integral (P&I)* pada kecepatan 1700 .

Kemudian dengan langkah yang sama, masukkan nilai *Proporsional* dan *Integral* pada kecepatan 1700, sehingga kita peroleh hasil sebagai berikut:




Gambar 4.9 Tampilan *GUI* dengan Pengendali *Proposional+Integral* pada Kecepatan 1700




Gambar 4.10 Grafik Respon dari pengendali *P & I* pada kecepatan 1700

Tabel 4.5 Hasil percobaan pengendali *P & I*

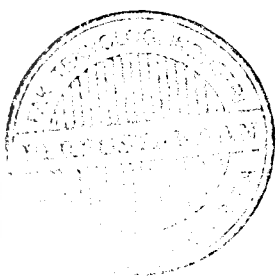
| P | I | D | Kecepatan | Waktu Pengereman | |
|---|---|---|---|---|---|
| | | | | Tanpa PID | P & I |
| -0.004 | -0.0004 | 0 | 1700 | 51,76525 | 1,059 |

Dari percobaan diatas, pengereman dengan menggunakan pengendali *PI*, dapat mempercepat proses pengereman dibandingkan pengereman tanpa *PID*, juga jika dibandingkan dengan pengereman dengan pengendali *P*. Dari hasil percobaan diatas, diketahui bahwa :

1. Pengendali *Proposional & Integral* dapat menghentikan putaran motor.
2. Waktu pengereman yang dibutuhkan motor dc dengan menggunakan pengendali *P&I* lebih cepat dibandingkan dengan hanya menggunakan pengendali *Proporsional.*
3. Selisih waktu yang terjadi antara pengereman dengan pengendali *P &I* dan pengendali *P* adalah:

   Waktu pengereman *PI* – Waktu *pengereman P* :

   14.60325 – 1.059 = 13.54425

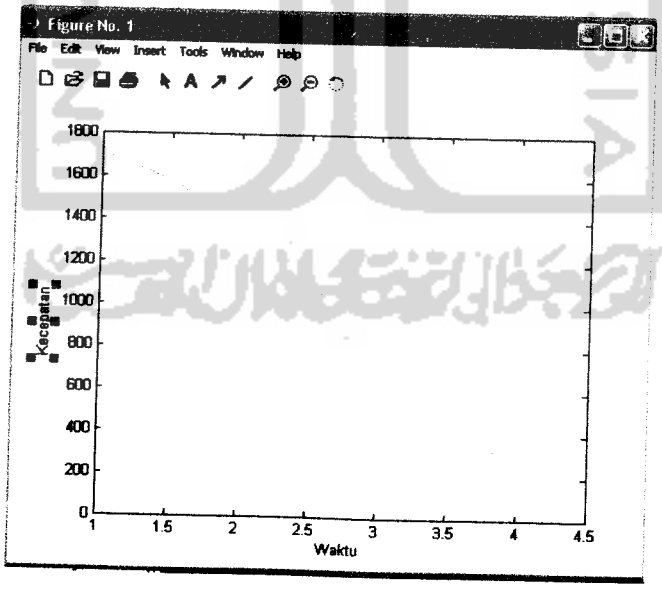


4. Pengendali *P&I* saja sudah cukup untuk mengendalikan (mengerem) sisa putaran motor tanpa disertakan pengendali derivatif.

Tabel 4.6 Pengereman dengan pengendali *P&I* (*P*=-0.004 & *I*=-0.0004)

| KECEPATAN | Percobaan1 | Percobaan2 | Percobaan3 | Percobaan4 | RATA-RATA |
|---|---|---|---|---|---|
| 100 | 4,239 | 4,243 | 4,247 | 4,245 | 4,2435 |
| 200 | 4,068 | 4,07 | 4,075 | 4,073 | 4,0715 |
| 300 | 3,897 | 3,892 | 3,894 | 3,892 | 3,89375 |
| 400 | 3,715 | 3,724 | 3,72 | 3,721 | 3,72 |
| 500 | 3,555 | 3,556 | 3,544 | 3,553 | 3,552 |
| 600 | 3,377 | 3,38 | 3,394 | 3,387 | 3,3845 |
| 700 | 3,228 | 3,218 | 3,224 | 3,221 | 3,22275 |
| 800 | 3,046 | 3,048 | 3,044 | 3,058 | 3,049 |
| 1000 | 2,702 | 2,709 | 2,709 | 2,712 | 2,708 |
| 1200 | 2,343 | 2,343 | 2,338 | 2,338 | 2,3405 |
| 1500 | 1,719 | 1,721 | 1,719 | 1,72 | 1,71975 |
| 1600 | 1,445 | 1,443 | 1,443 | 1,445 | 1,444 |
| 1700 | 1,059 | 1,059 | 1,059 | 1,059 | 1,059 |

Berikut ini akan terliahat grafik hubungan antara kecepatan dengan waktu pengeremannya dengan menggunakan pengendali *P&I:*



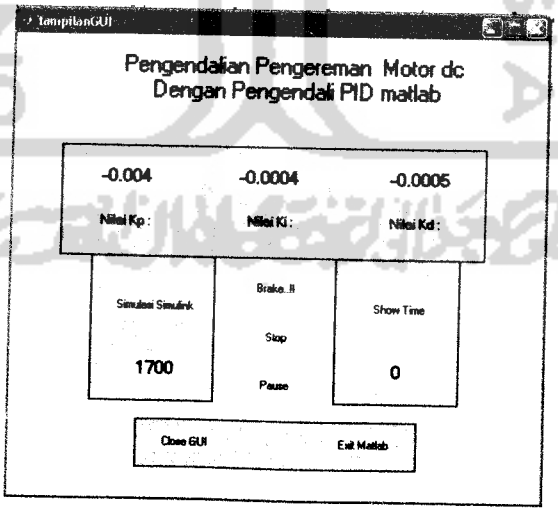

Gambar 4.11 Grafik Hubungan kecepatan dan waktu pengereman dengan pengendali P & I

### 4.3 Pengujian pengereman sistem dengan pengendali *Proporsional + Integral + Deriative*

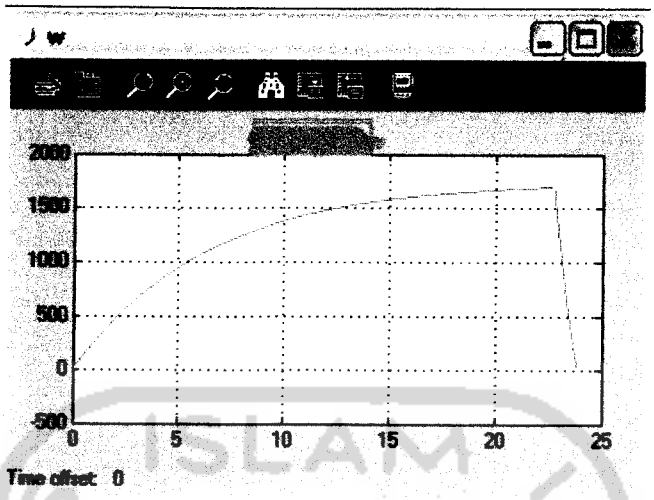
Untuk mulai menganalisis, kita masukkan nilai kecepatan yang diinginkan pada tampilan *GUI*. Kemudian tekan tombol "*Start Simulation*". Setelah *step respon* mencapai *steady state* pada kecepatan yang diinginkan , maka lakukan pengendalian dengan menggunakan pengendali *PID* untuk mendapatkan hasil *step respon* pengereman yang diinginkan dengan menekan tombol "*Brake..!!*" setelah sebelumnya mengisi parameter pengendali *PID*nya.. Lihat gambar 4.1.

#### *4.3.1* Percobaan 2 Pengereman dengan Pengandali *Proporsional +Integral+Derivatif (PID)* pada kecepatan 1700 .

Kemudian dengan langkah yang sama, masukkan nilai *PID*nya pada kecepatan 1700, sehingga diperoleh hasil sebagai berikut:




Gambar 4.12 Tampilan GUI dengan Pengendali *PID* pada Kecepatan 1700




Gambar 4.13 Grafik Respon dari pengendali *PID* pada kecepatan 1700

Table 4.7 Hasil percobaan pengendali *PID*

| P | I | D | Kecepatan | Waktu Pengereman | |
|---|---|---|---|---|---|
| | | | | Tanpa PID | PID |
| -0.004 | -0.0004 | -0.0005 | 1700 | 51,76525 | 1,1075 |

Dari percobaan diatas, pengereman dengan menggunakan pengendali *PID*, dapat mempercepat proses pengereman dibandingkan pengereman tanpa *PID*, namun jika dibandingkan dengan pengendali *P&I*, maka pengendali *PID* membutuhkan waktu lebih lama untuk mengerem dibandingkan dengan pengendali *P&I*. Dari hasil percobaan diatas, diketahui bahwa :

1. Pengendali *PID* dapat menghentikan putaran motor.
2. Waktu pengereman yang dibutuhkan motor dc dengan menggunakan pengendali *PID* lebih lambat dibandingkan dengan menggunakan pengendali *Proporsional+Integral*.

3. Selisih waktu yang terjadi antara pengereman dengan pengendali *PID* dan pengendali *P&I* adalah:

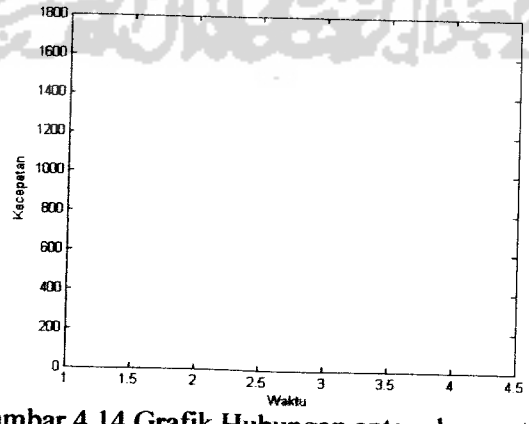
Waktu pengereman *PID* – Waktu *pengereman PI* :

1,1075– 1.059 = 0.0485

Tabel 4.8 Pengereman dengan pengendali *PID* (*P* = -0.004; *I* = -0.0004; *D* = -0.0005)

| KECEPATAN | Percobaan1 | Percobaan2 | Percobaan3 | Percobaan4 | RATA-RATA |
|---|---|---|---|---|---|
| 100 | 4,403 | 4,397 | 4,4 | 4,4 | 4,4 |
| 200 | 4,221 | 4,227 | 4,223 | 4,217 | 4,222 |
| 300 | 4,035 | 4,047 | 4,036 | 3,897 | 4,00375 |
| 400 | 3,864 | 3,861 | 3,868 | 3,863 | 3,864 |
| 500 | 3,696 | 3,69 | 3,694 | 3,69 | 3,6925 |
| 600 | 3,519 | 3,515 | 3,517 | 3,519 | 3,5175 |
| 700 | 3,52 | 3,53 | 3,535 | 3,521 | 3,5265 |
| 800 | 3,174 | 3,185 | 3,179 | 3,175 | 3,17825 |
| 1000 | 2,827 | 2,828 | 2,812 | 2,818 | 2,82125 |
| 1200 | 2,452 | 2,447 | 2,447 | 2,447 | 2,44825 |
| 1500 | 1,797 | 1,8 | 1,806 | 1,803 | 1,8015 |
| 1600 | 1,504 | 1,514 | 1,51 | 1,512 | 1,51 |
| 1700 | 1,107 | 1,107 | 1,107 | 1,109 | 1,1075 |

Berikut akan dilihat grafik hubungan antara kecepatan dengan waktu pengeremannya dengan pengendali *PID*:




Gambar 4.14 Grafik Hubungan antara kecepatan

Demikianlah hasil dari beberapa kali percobaan yang dilakukan. Untuk mendapatkan waktu pengereman rata-rata, telah melalui 4 kali percobaan pada masing-masing kecepatan untuk setiap pengendali.

## 4.4 Pengujian sistem dengan variasi nilai pengndali

Untuk melihat lebih detail pengaruh masing-masing pengendali PID, maka akan dilakukan beberapa pengujian sistem. Percobaan ini dilakukan pada kecepatan 1000.

### *4.4.1* Pengendali *Proporsional*

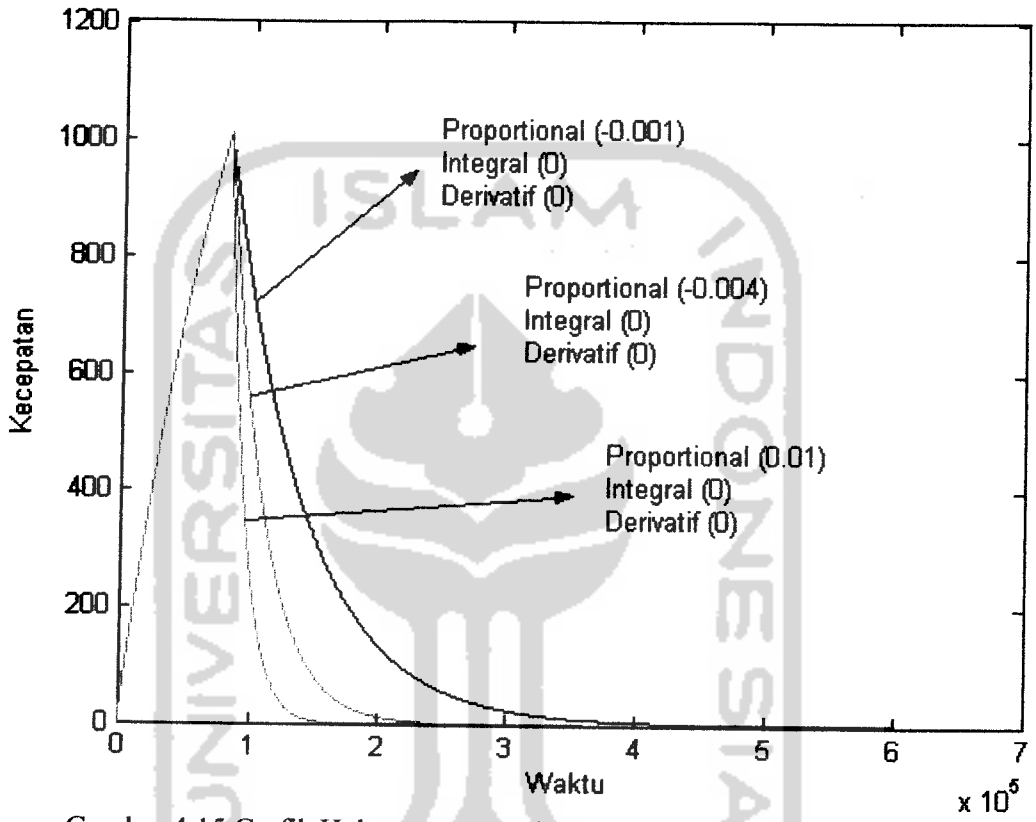
Tabel 4.9 Pengendali *Proporsional*

| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proposional* | *Integral* | *Derivatif* | |
| -0.001 | 0 | 0 | 29.608 |
| -0.004 | 0 | 0 | 13.647 |
| -0.01 | 0 | 0 | 6.575 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1. Pengendali *Proposional* memberikan pengaruh mempercepat pengereman motor dc dibandingkan dengan tanpa pengendali PID. Sesuai dengan karakteristik pengendali *Proposional,* yaitu mempersingkat waktu pengereman.
2. Semakin kecil nilai pengendali *Proposional* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin singkat.

3. Sangat lambat untuk memberhentikan (stop) motor dc. Sehingga untuk waktu pengereman untuk percobaan ini diambil pada saat kecepatan 0.5 *sample time*.




Gambar 4.15 Grafik Hubungan antara kecepatan dan waktu pengendali *P*

### *4.4.2* Pengendali *Integral*

Tabel 4.10 Pengendali Integral

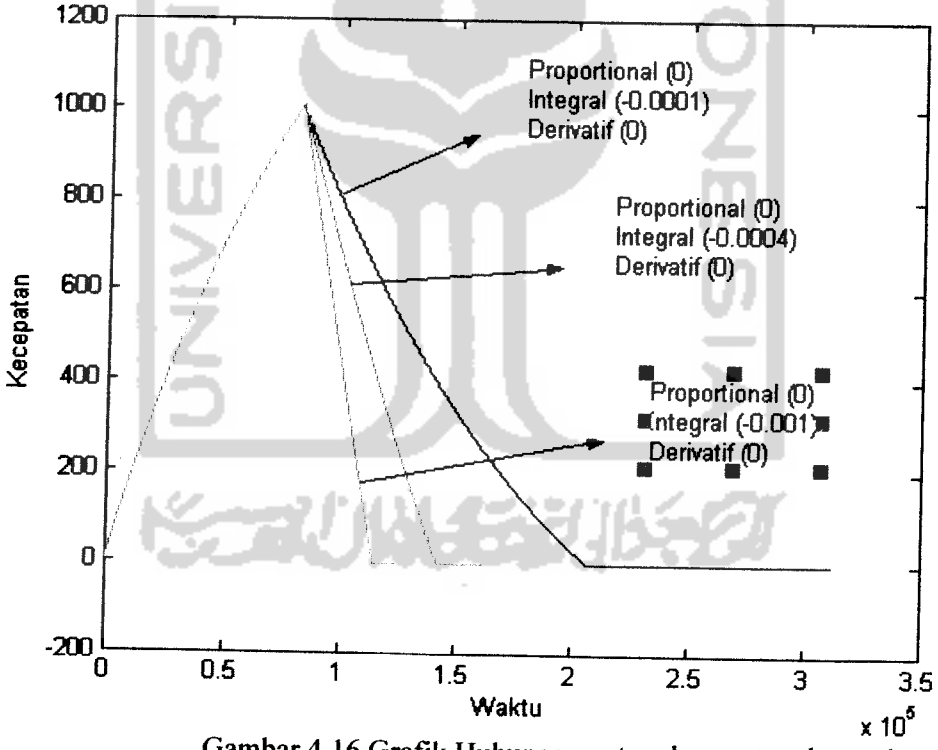
| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proporsional* | *Integral* | *Derivatif* | |
| 0 | -0.0001 | 0 | 8.295 |
| 0 | -0.0004 | 0 | 4.01 |
| 0 | -0.001 | 0 | 2.18 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1 Pengendali *Integral* memberikan pengaruh mempercepat pengereman motor dc dibandingkan dengan tanpa pengendali *PID* dan juga dibandingkan dengan pengendali *Propsional*. Sesuai dengan karakteristik pengendali *Integral*, yaitu akan memberikan efek peningkatan osilasi dan bila bernilai besar akan menghilangkan *offset*.

2 Semakin kecil nilai pengendali *Integral* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin singkat.

3 Motor dc berhenti dalam waktu yang singkat.




Gambar 4.16 Grafik Hubungan antara kecepatan dan waktu pengendali *I*

### 4.4.3 Pengendali *Derivatif*

Tabel 4.11 Pengendali *Derivatif*

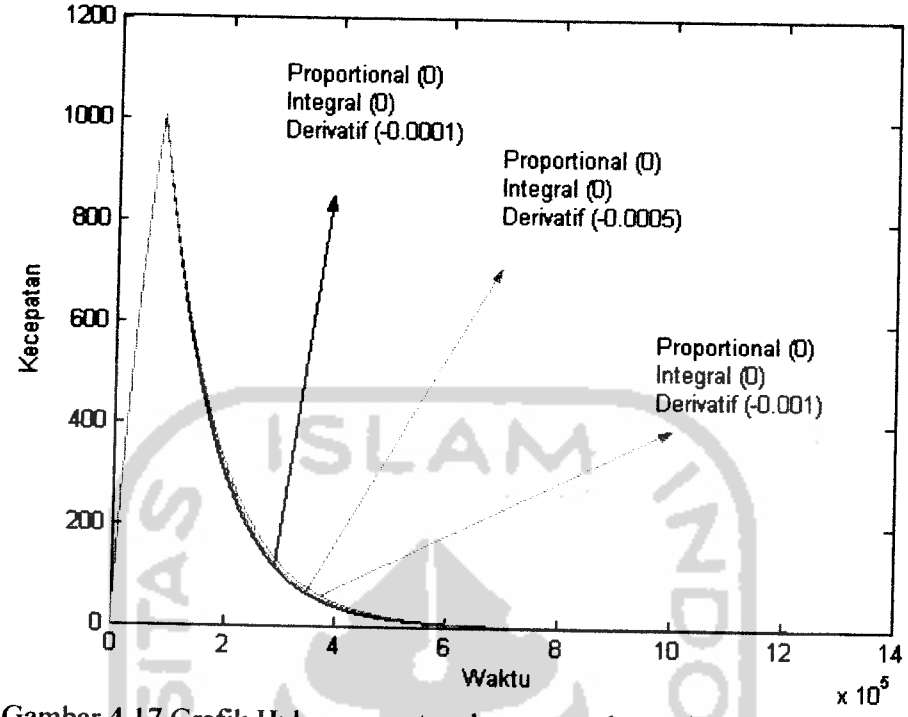
| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proporsional* | *Integral* | *Derivatif* | |
| 0 | 0 | -0.0001 | 48.654 |
| 0 | 0 | -0.0005 | 50.822 |
| 0 | 0 | -0.001 | 53.165 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1. Pengendali *Derivatif* memberikan pengaruh memperlambat pengereman motor dc dibandingkan dengan tanpa pengendali *PID* dan juga dibandingkan dengan pengendali *P* dan *I*. Sesuai dengan karakteristik pengendali *Derivatif,* yang cendrung meningkatkan stabilitas sistem dan menghilangkan *overshoot*, serta besarnya perubahan yang dialami sangat tergantung pada masukan kontroler, jika perubahannya mendadak maka akan mengakibatkan perubahan yang sangat besar dan cepat

1 Semakin kecil nilai pengendali *Derivatif* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin lama.

2 Sangat lambat untuk memberhentikan (stop) motor dc. Sehingga untuk waktu pengereman untuk percobaan ini diambil pada saat kecepatan 0.5 *sample time*.




Gambar 4.17 Grafik Hubungan antara kecepatan dan waktu pengendali *D*

### 4.4.4 Pengendali *Proporsional & Integral*

Tabel 4.12 Pengendali *Proporsional & Integral*

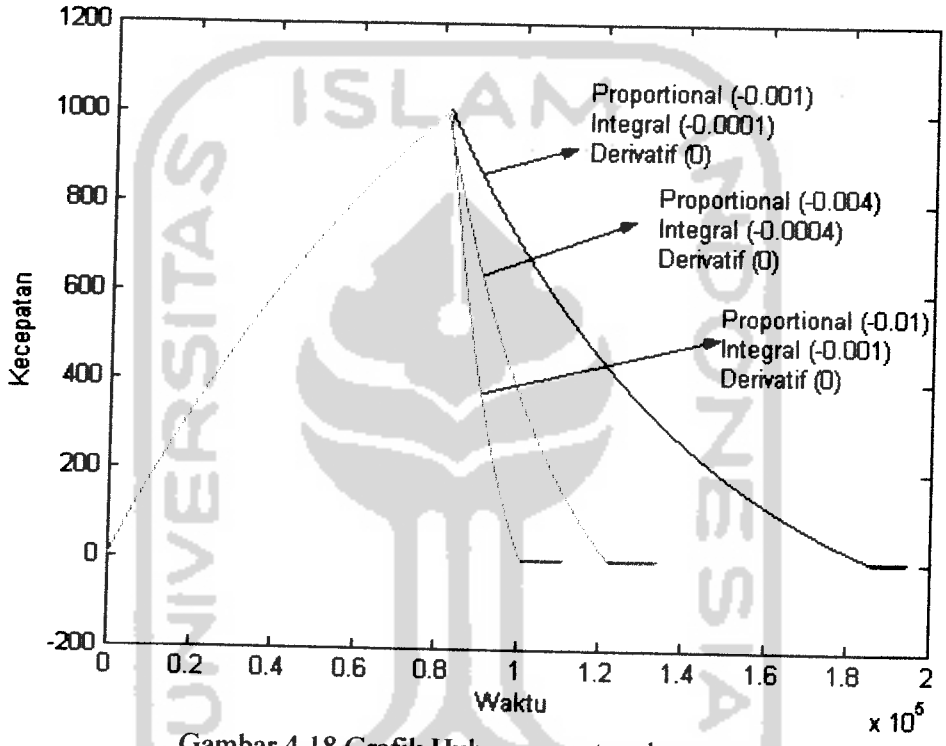
| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proposional* | *Integral* | *Derivatif* | |
| -0.001 | -0.0001 | 0 | 6.887 |
| -0.004 | -0.0004 | 0 | 2.703 |
| -0.01 | -0.001 | 0 | 1.256 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1 Gabungan pengendali *Proporsional & Integral* memberikan pengaruh mempercepat pengereman motor dc dibandingkan dengan tanpa pengendali *Proposional, Integral dan Derivatif.* Karena gabungan karakteristik 2 pengendali *Proporsional & Integral,*yang cendrung mengurangi waktu naik

dan memberikan efek peningkatan osilasi dan menghilangkan *offset* sehingga dapat mempersingkat waktu pengereman.

2. Semakin kecil nilai pengendali *Proporsional & Integral* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin singkat.
3. Motor dc dapat berhenti dalam waktu yang singkat.




Gambar 4.18 Grafik Hubungan antara kecepatan dan waktu pengendali *Proportional & Integral*

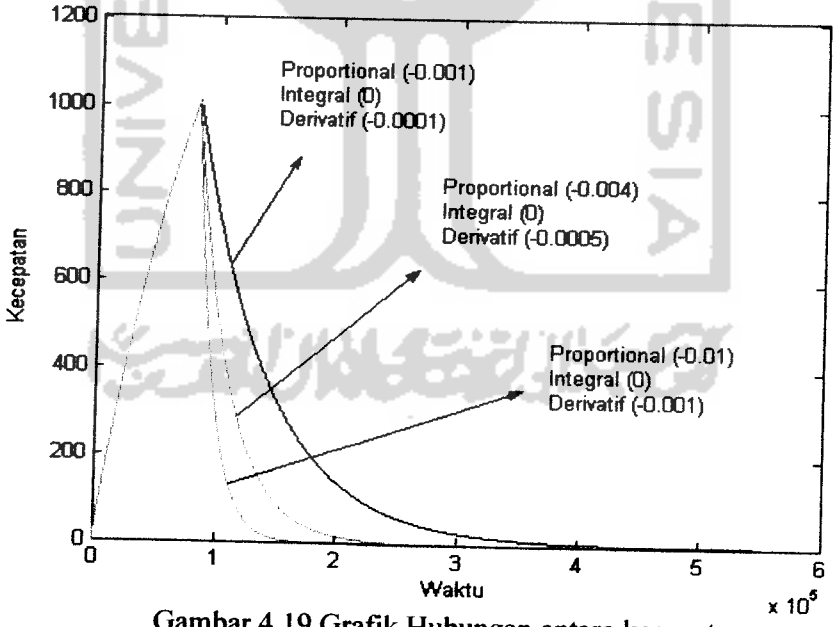
4.4.5 Pengendali *Proporsional & Derivatif*

Tabel 4.13 Pengendali *Proporsional &Derivatif*

| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proporsional* | *Integral* | *Derivatif* | |
| -0.001 | 0 | -0.0001 | 29.885 |
| -0.004 | 0 | -0.0005 | 14.323 |
| -0.01 | 0 | -0.001 | 7.223 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1. Tambahan pengendali *Derivatif* pada pengendali *Proporsional*, tidak memberikan pengaruh yang signifikan, karena perubahan yang tidak terlalu besar dari masukan pengendali *Derivatif* yaitu pengendali *Proporsional*.sehingga efek Pengendali *Derivatif* tidak terlalu signifikan, hanya sedikit memperlambat waktu pengereman dibandingkan dengan pengendali *P* saja.
2. Semakin kecil nilai pengendali *Proporsional & Derivatif* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin singkat.
3. Sangat lambat untuk memberhentikan (stop) motor dc. Sehingga untuk waktu pengereman untuk percobaan ini diambil pada saat kecepatan 0.5 *sample time*.




Gambar 4.19 Grafik Hubungan antara kecepatan dan waktu pengendali *Proporsional & Derivatif*

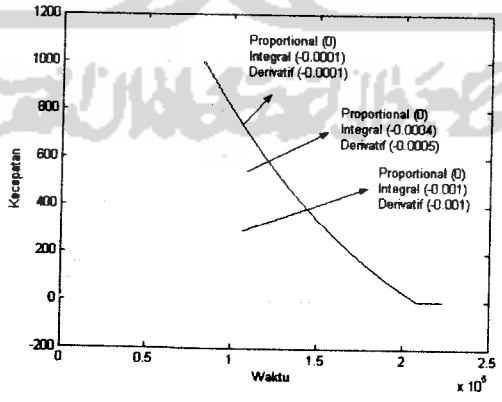
### 4.4.6 Pengendali *Integral & Derivatif*

Tabel 4.14 Pengendali *Integral & Derivatif*

| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proporsional* | *Integral* | *Derivatif* | |
| 0 | -0.0001 | -0.0001 | 8.343 |
| 0 | -0.0004 | -0.0005 | 4.167 |
| 0 | -0.001 | -0.001 | 2.356 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1. Penambahan pengendali *Derivatif* pada pengendali *Integral* akan memperlambat waktu pengereman dibandingkan dengan pengndali *I* saja. Hal ini karena dengan ditambahkannya pengendali *Derivatif* yang efeknya cendrung menstabilkan sistem, sehingga pada percobaan ini menimbulkan efek memperlambat waktu pengereman.
2. Semakin kecil nilai pengendali *Integral & Derivatif* (semakin mendekati "-1"), maka waktu pengereman yang dibutuhkan motor dc semakin singkat.
3. Motor dc dapat berhenti dalam waktu yang singkat.




Gambar 4.20 Grafik Hubungan antara kecepatan dan waktu pengendali *Integral & Derivatif*

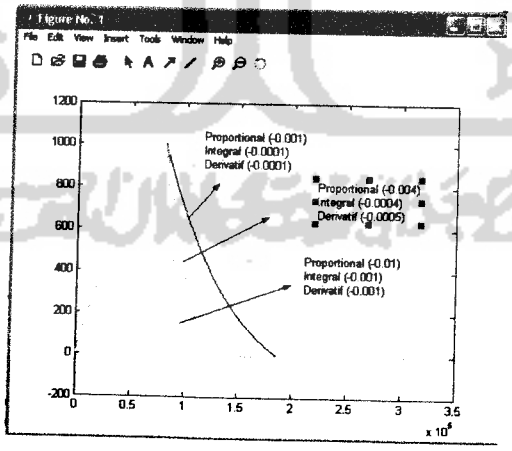
### 4.4.7 Pengendali *Proporsional, Integral, Derivatif*

Tabel 4.15 Pengendali *PID*

| Pengendali | | | Waktu pengereman |
|---|---|---|---|
| *Proporsional* | *Integral* | *Derivatif* | |
| -0.001 | -0.0001 | -0.0001 | 6.962 |
| -0.004 | -0.0004 | -0.0005 | 2.833 |
| -0.01 | -0.001 | -0.001 | 1.375 |

Dari percobaan diatas, dapat ditarik beberapa kesimpulan:

1. Gabungan pengendali *Proporsional, Integral & Derivatif* secara umum dapat mempercepat waktu pengereman sesuai dengan karakteristik masing-masing yang telah dijelaskan diatas.
2. Waktu pengereman akan lebih singkat lagi jika pengendali *D* tidak disertakan.
3. Semakin kecil nilai pengendali *PID*, maka waktu pengereman yang dibutuhkan semakin singkat.
4. Motor dc dapat berhenti. Dalam waktu yang singkat.




Gambar 4.21 Grafik Hubungan antara kecepatan dan waktu pengendali *Proporsional Integral & Derivatif*

# BAB V

# PENUTUP

## 5.1 Kesimpulan

Berdasarkan pengujian pengereman motor dc dengan pengendali PID yang ditampilkan melalui GUI Matlab yang telah dilakukan, maka dapat diambil beberapa kesimpulan sebagai berikut:

1. Pengendali PID dapat mengendalikan pengereman pada motor dc.
2. Parameter yang digunakan dalam penelitian ini adalah : *Proporsional (P) = -0.004, Integral (I) = -0.0004, Derivatif = -0.0005.* Dan parameter-parameter tersebut hanya mewakili saja untuk membuktikan bahwa pengendali PID dapat mempercepat pengereman motor.
3. Penggunaan kendali *Proporsional* tidak bisa menghentikan motor dc bila tidak dikombinasikan dengan pengendali PID lainnya yaitu *Integral* dan *Derivatif.*
4. Pengendali *P&I* saja sudah cukup untuk mengendalikan (mengerem) sisa putaran motor tanpa disertakan pengendali derivative, walaupun dengan pengendali *PID* juga mampu untuk mengerem sisa putara motor. Namun pengendali *PID* membutuhkan waktu pengereman lebih lama dibandingkan dengan pengendali *P&I.*
5. Waktu pengereman yang dibutuhkan motor dc dengan menggunakan pengendali *P&I* lebih cepat dibandingkan dengan hanya menggunakan pengendali *Proporsional* atau *PID.*

6. Pengendali *Derivative* sifatnya cendrung hanya menbuat sistem menjadi stabil sehingga ia akan menetralkan efek dari pengendali *Integral* yang cenrung menimbulkan *overshoot* pada respon sistem. Sedangkan tanpa pengendali derivative maka pengereman semakin cepat.

## 5.2 Saran

1. Pembahasan feature pengendali PID dengan GUI Matlab yang lebih lengkap.
2. Pengendali PID dapat diaplikasikan pada alat yang real.

# DAFTAR PUSTAKA


1. Philips,Charles L.,& Harbor,Royce D. 1996. *Dasar-Dasar Sistem Kontrol.* Alih Bahasa:Prof. R. J. Widodo. Jakarta. PT Prenhallindo.
2. Kuo, Benjamin C. 1995. *Teknik Kontrol Automatik.* Alih Bahasa: Ir. Mhd. Zulfan. Jakarta. PT Prenhallindo.
3. Wiyoso,Wisang. 2006. *Pengendalian PID Dengan GUI.* Skripsi, tidak diterbitkan. Jogjakarta: Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Wiryadinata, Romi. 2005. *Simulasi Jaringan Syaraf Tiruan Berbasis Metode Back Propagation Sebagai Pengendali Kecepatan Motor DC.* Skripsi, tidak diterbitkan. Jogjakarta: Fakultas Teknologi Industri Universitas Islam Islam.

```
function varargout = tampilanGUI(varargin)
% TAMPILANGUI M-file for tampilanGUI.fig
%      TAMPILANGUI, by itself, creates a new TAMPILANGUI or raises the existing
%      singleton*.
%
%      H = TAMPILANGUI returns the handle to a new TAMPILANGUI or the handle to
%      the existing singleton*.
%
%      TAMPILANGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TAMPILANGUI.M with the given input arguments.
%
%      TAMPILANGUI('Property','Value',...) creates a new TAMPILANGUI or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before tampilanGUI_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to tampilanGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help tampilanGUI

% Last Modified by GUIDE v2.5 20-Jan-2007 10:12:07

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @tampilanGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @tampilanGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tampilanGUI is made visible.
function tampilanGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to tampilanGUI (see VARARGIN)
fig = handles.stop;
model_open(handles)

% Choose default command line output for tampilanGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tampilanGUI wait for user response (see UIRESUME)
% uiwait(handles.sip);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = tampilanGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function model_open(handles)
% Make sure the simulink jst2 is still open
if  isempty(find_system('Name','motordc')),
        open_system('motordc');
        figure(handles.sip);
end

% --- Executes during object creation, after setting all properties.
function kp_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kp_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function kp_edit_Callback(hObject, eventdata, handles)
% hObject    handle to kp_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kp_edit as text
%        str2double(get(hObject,'String')) returns contents of kp_edit as a
double


kp=str2double(get(handles.kp_edit,'string'));
if isnan(kp)
    set(hObject,'string',[]);
    errordlg('Masukan harus berupa angka');
end;
if(kp<-1)|(kp>0)
    set(hObject,'string',[]);
    errordlg('Nilai antara -1 sampai 0');
end;


% --- Executes during object creation, after setting all properties.
function ki_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ki_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function ki_edit_Callback(hObject, eventdata, handles)
% hObject    handle to ki_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ki_edit as text
%        str2double(get(hObject,'String')) returns contents of ki_edit as a
double
ki=str2double(get(handles.ki_edit,'string'));
if isnan(ki)
    set(hObject,'string',[]);
    errordlg('Masukan harus berupa angka');
end;
if(ki<-1)|(ki>0)
    set(hObject,'string',[]);
    errordlg('Nilai antara -1 sampai 0');
end;


% --- Executes during object creation, after setting all properties.
function kd_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function kd_edit_Callback(hObject, eventdata, handles)
% hObject    handle to kd_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd_edit as text
%        str2double(get(hObject,'String')) returns contents of kd_edit as a
double

kd=str2double(get(handles.kd_edit,'string'));
if isnan(kd)
    set(hObject,'string',[]);
    errordlg('Masukan harus berupa angka');
end;
if(kd<-1)|(kd>0)
    set(hObject,'string',[]);
    errordlg('Nilai antara -1 sampai 0');
end;


% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
set_param('motordc/DC Voltage Source','V',num2str(0));
set_param('motordc/PID Controller','P',get(handles.kp_edit,'String'));
    set_param('motordc/PID Controller','I',get(handles.ki_edit,'String'));
    set_param('motordc/PID Controller','D',get(handles.kd_edit,'String'));
     set_param ('motordc','simulationcommand','update')
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in tutup.
function tutup_Callback(hObject, eventdata, handles)
% hObject    handle to tutup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.sip);


% --- Executes on button press in keluar.
function keluar_Callback(hObject, eventdata, handles)
% hObject    handle to keluar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
exit;


% --- Executes on button press in mulai.
function mulai_Callback(hObject, eventdata, handles)
% hObject    handle to mulai (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


x = str2num(get(handles.n_edit,'string'));
N = x
Vt = (150/1750)*N;
set_param('motordc/DC Voltage Source','V',num2str(Vt));
status = get_param('motordc','simulationstatus')
if strcmp (status,'stopped')
    set_param ('motordc','simulationcommand','start')
else if strcmp (status,'running')
        set_param ('motordc','simulationcommand','update')
    end
end


% --- Executes during object creation, after setting all properties.
function n_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to n_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function n_edit_Callback(hObject, eventdata, handles)
% hObject    handle to n_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of n_edit as text
%        str2double(get(hObject,'String')) returns contents of n_edit as a double

N=str2double(get(handles.n_edit,'string'));
if isnan(N)
    set(hObject,'string',[]);
    errordlg('Masukan harus berupa angka');
```

```
end;
if(N<0)|(N>1750)
    set(hObject,'string',[]);
    errordlg('Nilai antara 1 sampai 1750');
end;


% --- Executes during object creation, after setting all properties.
function edit_show_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_show (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_show_Callback(hObject, eventdata, handles)
% hObject    handle to edit_show (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_show as text
%        str2double(get(hObject,'String')) returns contents of edit_show as a
double


% --- Executes on button press in show.
function show_Callback(hObject, eventdata, handles)
% hObject    handle to show (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

waktu=handles.tstop-handles.t1
waktustr=num2str(waktu);
set (handles.edit_show,'string',waktustr);
guidata(hObject,handles)

% --- Executes on button press in stopsimulasi.
function stopsimulasi_Callback(hObject, eventdata, handles)
% hObject    handle to stopsimulasi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set_param ('motordc','simulationcommand','pause')
handles.tstop=get_param('motordc','simulationtime')
set_param ('motordc','simulationcommand','stop')
guidata(hObject,handles)

% --- Executes on button press in paused.
function paused_Callback(hObject, eventdata, handles)
% hObject    handle to paused (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

status = get_param('motordc','simulationstatus')
if strcmp (status,'running')
    set_param ('motordc','simulationcommand','pause')
else if strcmp (status,'paused')
        set_param ('motordc','simulationcommand','continue')
    end
end
```

```
                                        tampilanGUI
handles.t1=get_param('motordc','simulationtime')
guidata(hObject,handles)
```