Template Laporan Hardware In The Loop

Nama Anggota: 1) Richard

2) Christian Reivan

NIM : 1) 13219011

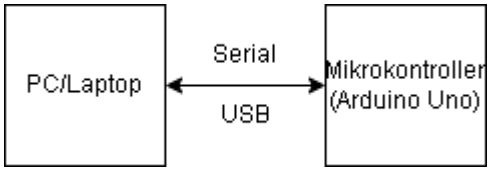
2) 13219005

Contents

1 Perancangan Hardware

Skema rangkaian & penjelasan

Pada percobaan yang dilakukan, *hardware* yang digunakan terdiri atas PC/laptop, kabel USB, dan Arduino Uno. Berikut ini adalah skema rangkaiannya.



Gambar 1 Skema Rangkaian Perancangan Hardware

PC/laptop berisi *software* Simulink yang berfungsi untuk memberikan sinyal *input* berupa *unit step*. Sinyal *input* tersebut dibandingkan dengan sinyal umpan balik. Hasil perbandingan tersebut adalah *error* yang dikirimkan ke Arduino Uno melalui *port serial* USB. Arduino Uno yang telah di-*upload* program PID beserta konstanta-konstantanya berfungsi untuk menghitung PID berdasarkan nilai *error* yang diperoleh. Hasil perhitungan PID tersebut menjadi sinyal kontrol yang kemudian dikirimkan kembali ke PC/laptop melalui *port serial* USB yang sama. Pada PC (Simulink), sinyal kontrol berfungsi sebagai sinyal *input* bagi *plant* yang berbentuk fungsi transfer motor DC orde satu. Output *plant* kemudian ditampilkan dalam bentuk grafik serta diumpanbalikkan untuk dibandiingkan dengan sinyal *input unit step*.

2 Perancangan Software

2.1 Komunikasi Data

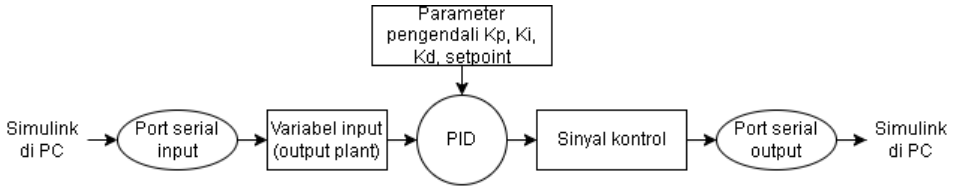
Format data untuk komunikasi serial anatara PC ke Arduino dan Arduino ke PC adalah sama. Data dikirim dalam bentuk blok data, yang terdiri dari 4 *byte* data dan *terminator*. Sebenarnya dapat juga digununakan *header* di depan 4 *byte data* sebagai penanda awal blok data, namun pada kasus ini tidak menggunakan *header*. Dalam satu blok data terdiri dari 4 *byte* data karena data yang dikirim bertipe *single precision*. Lalu, diakhir blok data diberikan terminator sebagai penanda akhir blok data, yaitu '\r\n' atau dalam *integer* yang merepresentasikan karakter ASCII adalah 13 dan 10. Berikut ini adalah contoh ilustrasi blok data yang digunakan pada komunikasi serial yang dilakukan.

Byte 1	Byte 2	Byte 3	Byte 4	'\r' (13)	'\n' (10)

Gambar 2 Ilustrasi Blok Data pada Komunikasi Serial antara PC dan Arduino

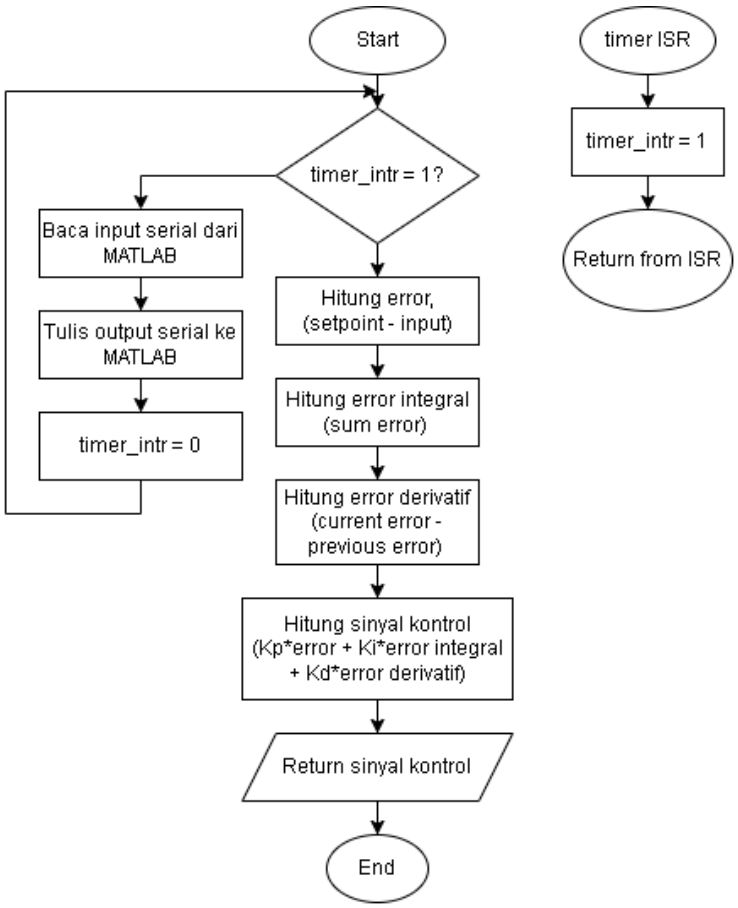
2.2 Pengendali

Berikut ini adalah diagram blok perangkat lunak di pengendali (Arduino Uno) dalam format *data flow diagram*.



Gambar 3 Data Flow Diagram Pengendali

Berikut ini adalah diagram alir perangkat lunak di Arduino Uno (pengendali).



Gambar 4 Flowchart Pengendali

2.2.1 Port Serial

Pada percobaan yang dilakukan, PC dan Arduino Uno berkomunikasi secara serial. Komunikasi dilakukan untuk mengirimkan sinyal kontrol dari Arduino (pengendali PID) ke PC (Simulink) dan mengirimkan sinyal *output plant* dari PC ke Arduino. Oleh karena itu, pada Arduino (pengendali) perlu dilakukan inisialisasi serial pada bagian *setup* menggunakan kode `Serial.begin(115200)`. *Baud rate,* menyatakan kecepatan transmisi data secara serial, yang digunakan 115200 *bits per second.* Angka *Baud rate* tersebut harus sama dengan pengaturan *baud rate* pada PC (Simulink).

2.2.2 PID

Pada kode Arduino, dibuat fungsi yang menghitung PID berdasarkan sinyal *input* Arduino (sinyal *output plant*), *setpoint* (*unit step*), dan konstanta pengendali Kp, Ki, dan Kd. Kode dibuat berdasarkan definisi pengendali PID berikut ini.

- Proporsional
  Dalam domain waktu kontinyu, hubungan antara sinyal eror e dengan sinyal kontrol u dinyatakan dalam persamaan berikut:
  $$u(t) = K_p e(t)$$
  Dari persaamaan diatas terlihat bahwa pengendali proporsional menghasilkan sinyal kontrol berupa sinyal eror yang dikalikan (proporsional) dengan konstanta proporsonal Kp. Pengendali proporsional digunakan untuk memperbesar penguatan dan mempercepat respon transien.
- Integral
  Dalam pengendali integral, nilai eror e diumpankan sebagai laju perubahan sinyal kontrol u sebagaimana dinyatakan dalam persamaan berikut ini:
  $$u(t) = K_i \int_0^t e(t)dt$$
  Pengendali integral berfungsi untuk menghilangkan galat atau steady state error meskipun juga dapat menyebabkan terjadinya overshoot dan osilasi yang mengakibatkan keadaan tunak lama dicapai.
- Derivatif
  Pengendali derivatif akan memberikan suatu sinyal kontrol u yang bersesuaian dengan laju perubahan sinyal eror e sebagaimana dinyatakan dalam persamaan berikut ini:
  $$u(t) = K_d \frac{de(t)}{dt}$$
  Pengendali ini digunakan untuk mempercepat respon transien meskipun memiliki kekurangan, yaitu dapat meningkatkan derau sistem

Berikut ini adalah implementasi PID pada kode Arduino.

```
void PIDCompute(){
 float error, errorI, errorD, prevError=0;
 error = setPoint - input; errorI += error;
 errorD = (error-prevError);
 output = (Kp*error)+(Ki*errorI)+(Kd*errorD);
 prevError=error;
}
```

Perhitungan PID pada Arduino dilakukan pada domain diskrit oleh karena itu integral diganti dengan penjumlahan dan derivatif diganti dengan pengurangan/selisih.

2.2.3 Parameter Kendali

Seperti yang telah dijelaskan di atas, pengendali yang digunakan adalah pengendali proporsional, integral, dan derivatif. Oleh karena itu, parameter kendali yang digunakan ada 3, yaitu konstanta proporsional Kp, konstanta integral Ki, dan konstanta derivatif Kd. Nilai konstanta PID tersebut ditentukan dengan cara melakukan *tuning* menggunakan metode yang dikembangkan oleh Tim Wescott, [1]. Langkah-langkahnya adalah sebagai berikut.

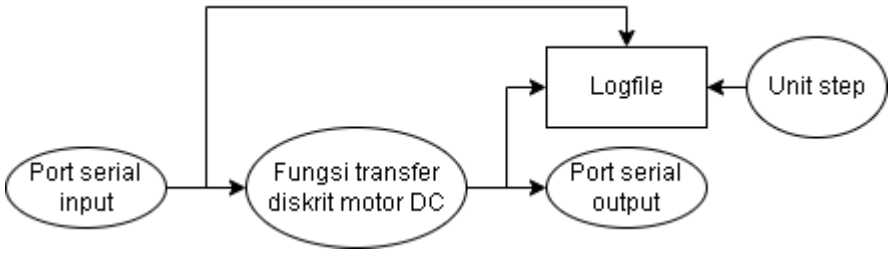
1. Pertama-tama adalah menyetel semua konstanta PID 0. Setelah itu, set konstanta proporsional Kp cukup kecil hingga tidak terjadi osilasi atau osilasinya lambat. Jika terjadi osilasi maka berikan nilai pada konstanta derivatif Kd mulai dari 100 kali lebih besar dari Kp. Jika osilasi bertambah setelah diberi pengendali derivatif maka perkecil nilai Kd dengan faktor dua hingga osilasi berhenti. Jika tidak perlu mengurangi Kd untuk menghilangkan osilasi, maka nilai Kd dapat dinaikkan hingga terjadi osilasi kemudian kurangi nilai Kd menjadi setengah atau seperempatnya.

Selanjutnya adalah menyetel konstanta proporsional, yaitu dimulai dari 1/100 Kd. Jika terjadi osilasi maka Kp diturunkan menjadi seperdelapan atau sepersepuluh hingga osilasi berhenti. Jika tidak terjadi osilasi, maka naikkan nilai Kp dengan faktor 8 atau 10 hingga terjadi osilasi kemudian perkecil kembali dengan faktor 2 atau 4.

Untuk menyetel konstanta integral Ki, dapat dimulai dengan mengeset nilai Ki lebih kecil dari Kp dengan rasio yang sama seperti Kp terhadap Kd. Jika terjadi osilasi maka perkecil nilai Ki dengan faktor 8 atau 10. Jika tidak terjadi osilasi, maka naikkan nilai Ki dengan faktor 8 atau 10 hingga terjadi osilasi kemudian perkecil kembali dengan faktor 2 atau 4.

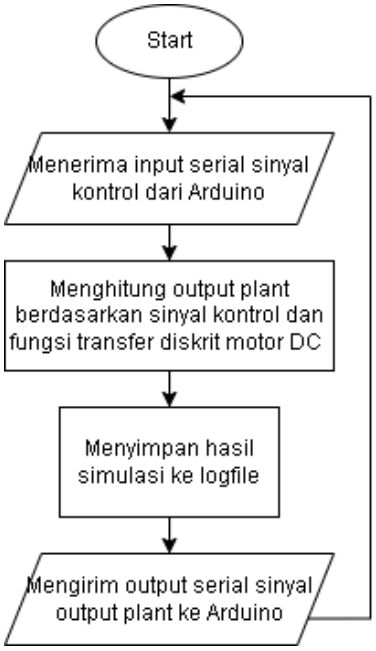
2.3 Simulator

Berikut ini adalah diagram blok perangkat lunak di simulator dalam format *data flow diagram*.



Gambar 5 Data Flow Diagram Simulator

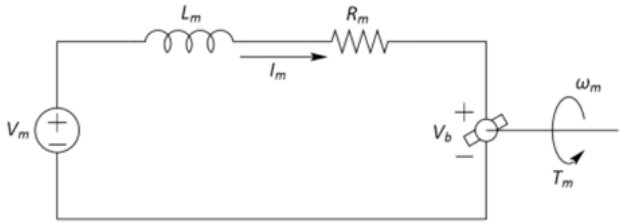
Diagram alir perangkat lunak di simulator dapat dilihat pada gambar selanjutnya.



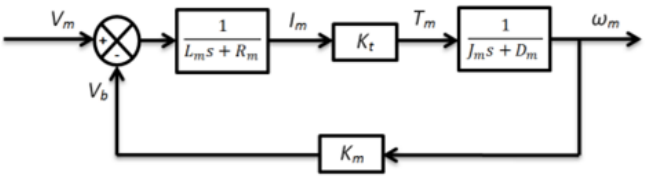
Gambar 6 Flowchart Simulator

2.3.1 Plant

Plant pada Simulink merupakan sebuah fungsi transfer yang merepresentasikan motor DC. Fungsi transfer tersebut diperoleh dari hasil pengukuran dan percobaan pada praktikum sistem kendali. Penurunan fungsi transfer dimulai dari memodelkan motor DC menjadi diagram blok seperti berikut.



Gambar 7 Model Rangkaian Motor DC



Gambar 8 Diagram Blok yang Merepresentasikan Motor DC

Secara umum, dalam domain Laplace, hubungan antara tegangan masukan motor Vm dengan kecepatan putaran rotor ωm dinyatakan dalam persamaan berikut:

$$\frac{\omega_m(s)}{V_m(s)} = \frac{K_t}{J_m L_m s^2 + (J_m R_m + D_m L_m)s + R_m D_m + K_m K_t}$$

Pada umumnya, Lm cukup kecil bila dibandingkan dengan Rm, sehingga persamaan di atas dapat disederhanakan menjadi berikut:

$$\frac{\omega_m(s)}{V_m(s)} = \frac{\frac{K_t}{J_m R_m}}{s + \frac{D_m}{J_m} + \frac{K_m K_t}{J_m R_m}}$$

Atau dapat juga disederhanakan menjadi bentuk umum sebagai berikut :

$$\frac{\omega_m(s)}{V_m(s)} = \frac{K}{\tau s + 1}$$

Dari hasil percobaan diperoleh fungsi transfer motor DC adalah

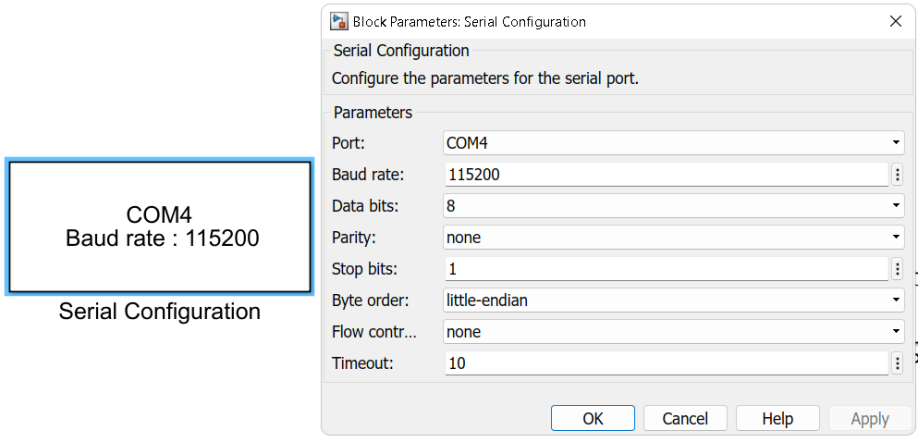
$$\frac{\omega_m(s)}{V_m(s)} = \frac{20.102}{s + 0.7471}$$

Fungsi transfer tersebut masih dalam domain kontinu s. Untuk melakukan simulasi *hardware in the loop* maka perlu diubah ke domain diskrit karena semua prosesnya dilakukan pada domain digital. Oleh karena itu, dengan transformasi bilinear atau transformasi Tustin, fungsi transfer motor DC tersebut diubah ke dalam domain z menjadi seperti berikut.

$$\frac{\omega_m(z)}{V_m(z)} = \frac{3.73 \times 10^{-4} z + 3.73 \times 10^{-4}}{z - \frac{2409}{2411}}$$

2.3.2 Port Serial

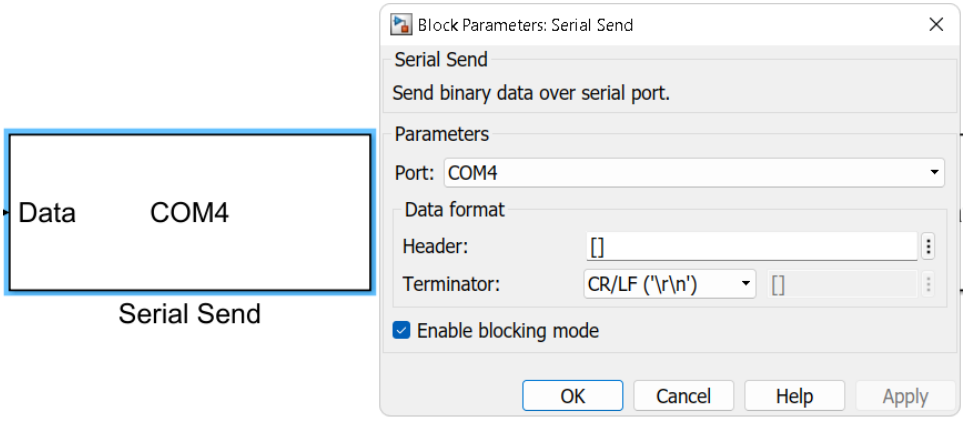
Untuk melakukan komunikasi serial antara PC (Simulink) dan Arduino (pengendali PID), pada Simulink menggunakan *instrument control toolbox*. Pada *toolbox* tersebut terdapat blok *serial send*, *serial receive*, dan *serial configuration*.



Gambar 3

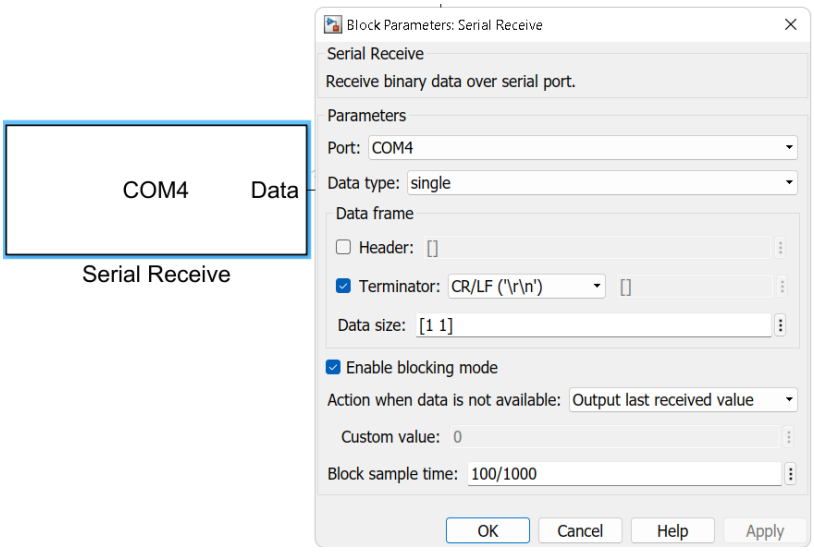
Untuk melakukan pengiriman dan penerimaan data secara serial maka perlu melakukan konfigurasi parameter *serial port* menggunakan blok *serial configuration*. Parameter yang dapat diatur dapat dilihat

di atas, namun yang terpenting adalah *port*, *baud rate*, dan *data bits*, yang lainnya dapat dibiarkan sesuai pengaturan *default*-nya. Pada parameter *port*, pilih *port serial* yang sesuai dengan yang digunakan untuk Arduino (dapat dilihat pada *device manager* dan Arduino harus dihubungkan ke *port serial* sebelum Matlab dan Simulink dibuka supaya dapat terdeteksi). *Baud rate* menyatakan kecepatan transmisi data secara serial dalam *bits per second*. Pada kasus ini, digunakan *baud rate* 115200 dan angka tersebut harus sama antara Simulink dan Arduino. Terakhir, data dikirimkan dalam bentuk *byte* maka *data bits* yang digunakan adalah 8.



Gambar 4

Blok *serial send* berfungsi untuk mengirimkan data melalui *serial port*. Pada kasus ini, data yang dikirimkan dalam format *single precision* (4 *byte*). Pada data yang dikirimkan dapat diatur *header dan terminator*. *Header* terletak di depan blok data berfungsi sebagai penanda awal blok data. *Header* dapat diatur menggunakan *integer* yang merepresentasikan karakter ASCII, namun pada *default* tidak menggunakan *header*. *Terminator* mirip dengan *header* namun berada di belakang blok data sebagai penanda akhir blok data. Pada kasus ini digunakan *terminator* '\r\n'.

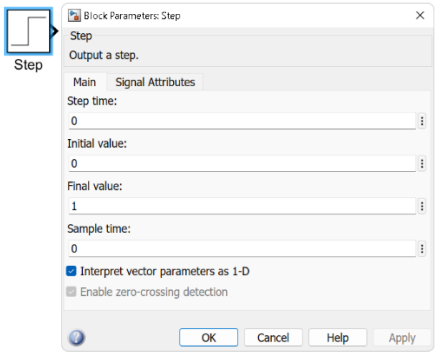


Gambar 5

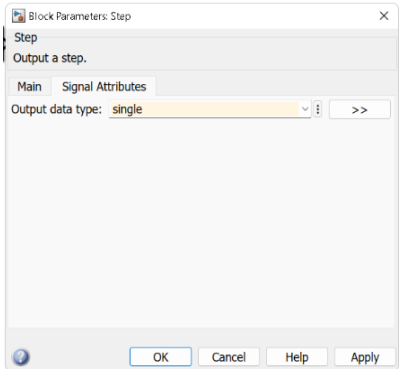
Blok *serial receive* berfungsi untuk menerima data secara serial dari Arduino. Sama seperti *serial send*, parameter yang dapat diatur adalah *port*, *data type*, *header*, dan *terminator*. Untuk memudahkan, pengaturan *serial receive* disamakan dengan *serial send*. Selain itu, ada beberapa parameter tambahan yang dapat diatur, yaitu *enable blocking mode* berfungsi untuk menunggu untuk waktu yang telah ditentukan *timeout* pada blok *serial configuration* hingga blok *serial receive* menerima data. Jika data tidak ada yang diterima maka aksi yang dilakukan dapat dipilih pada *action when data is not available*, pada kasus ini dipilih *output last received value*. Terakhir, terdapat parameter *block sample time* yaitu waktu *sampling* blok atau dalam kata lain periode blok *serial receive* tersebut dieksekusi pada saat simulasi.

2.3.3 Signal Generator

Pada Simulink sinyal *input/setpoint yang* berupa sinyal *unit step* dihasilkan oleh blok *step.* Berikut ini adalah gambar bloknya.



Gambar 6

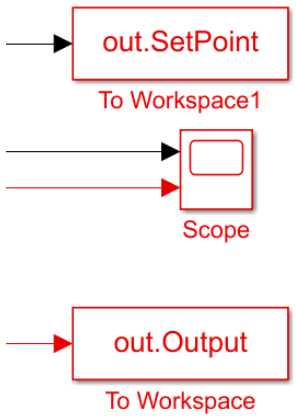


Gambar 7

Ada beberapa parameter yang dapat diatur pada blok *step*. *Step time* yaitu waktu terjadinya *step*, *initial value* yaitu nilai *output* sebelum terjadi *step, final value* yaitu nilai *output* setelah terjadi *step*, *sample time* yaitu periode *sampling* sinyal *step,* dan pada bagian *signal attributes* dapat dipilih tipe data dari *output* blok.

2.3.4 Mekanisme Log

Hasil simulasi yang diperoleh dapat ditampilkan dalam bentuk grafik menggunakan *scope*. Pada percobaan yang dilakukan, *scope* dibuat menjadi dua kanal yaitu kanal pertama untuk *setpoint* dan kanal kedua untuk sinyal keluaran *plant*. Selain itu, digunakan juga blok *to workspace* untuk memasukkan data hasil simulasi ke dalam variabel pada *workspace* MATLAB. Pada MATLAB, data hasil simulasi tersebut dapat diolah menjadi grafik dan dihitung parameter transien responnya, seperti *rise time*, *settling time*, *overshoot*, *peak*, *peak time*, dan lain-lain menggunakan fungsi *plotting_and_stepinfo dibuat. Fungsi tersebut dapat dilihat pada bagian lampiran.*



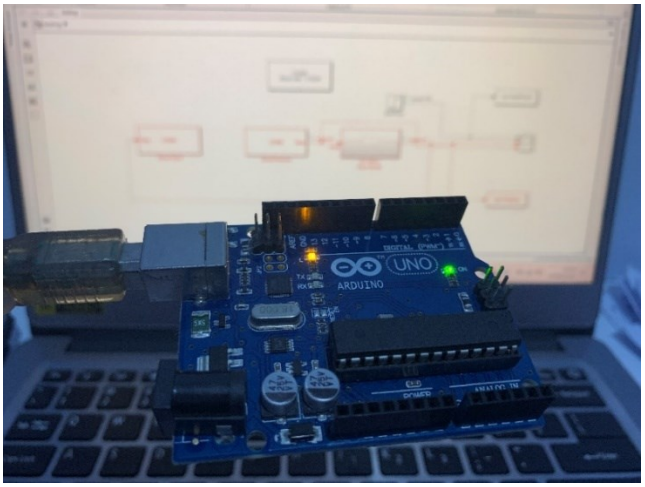
Gambar 8

3 Implementasi Hardware

Pada tugas ini, skema pengerjaan kami adalah pertama-tama menguji pada software saja (MATLAB), kemudian menguji dengan metode processor-in-the-loop (plant di MATLAB & controller di processor Atmega328P), dan terakhir mengimplementasikan hardware-in-the-loop.

Pada pengujian yang dilakukan dengan software saja, perangkat yang diperlukan hanyalah laptop karena akseleratornya berupa CPU Laptop.

Pada pengujian yang dilakukan dengan PIL (*processor-in-the-loop*) dan HIL (*hardware-in-the-loop*), perangkat keras yang digunakan berupa sebuah mikrokontroler Arduino Uno.



Gambar 9: Perangkat Keras yang Digunakan

Seperti yang telah disebut sebelumnya, kami hanya menggunakan sebuah Arduino Uno sebagai akselerator Controller PID. Model plant dari Motor DC dilakukan pada software SIMULINK. Akan tetapi, sekalipun plant dilakukan pada SIMULINK, proses sistem kendali ini berjalan real-time. Hal ini direalisasikan dengan bantuan Instrumentation System Toolbox dan Embedded Coder Toolbox dari MATLAB yang memungkinkan komunikasi dua arah (two-way communication) secara asinkron dengan menggunakan protokol komunikasi serial.

Pada percobaan ini kita tidak memerlukan peripheral I/O karena semuanya dijalankan langsung (deploy) pada chip Arduino Uno itu sendiri, Atmega328P.

4 Implementasi Software

4.1 Pengendali

Untuk simulasi HIL, pengendali PID diimplementasikan pada Arduino Uno dengan membuat kode C++ pada environment Arduino IDE. Implementasinya menggunakan bantuan timer interrupt dengan timer1. Periode sampling atau interrupt time sebesar 5 ms. Nilai ini cukup cepat dibandingkan sampling time yang digunakan pada software SIMULINK yang bernilai 100 ms. Kemudian, berdasarkan observasi yang dilakukan, ditemukan bahwa input yang masuk ke pengendali PID harus difilter lowpass dahulu dengan frekuensi cutoff 5 Hz. Nilai ini diperoleh dari pengamatan spektrum frekuensi output di MATLAB. Pembuatan filter lowpass ini menggunakan transformasi Tustin pada fungsi transfer dari filter lowpass analog yang berbentuk:

$$H(s) = \frac{\omega_0}{s + \omega_0}$$

dengan mensubstitusikan:

$$s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right)$$

Tetapi proses ini agak menyulitkan dikerjakan dengan tangan sehingga proses ini dilakukan dengan menggunakan Python (lihat lampiran) dan diperoleh persamaan diskret sebagai berikut:

$$y[n] = 0.733 \cdot y[n-1] + 0.133 \cdot x[n] + 0.133 \cdot x[n-1]$$

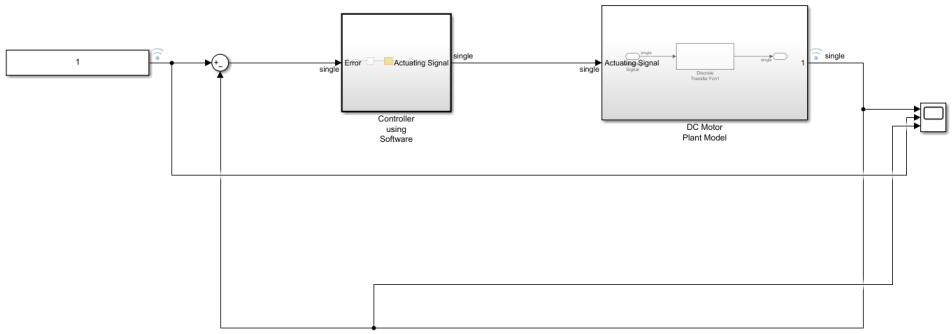
Setelah melalui filter lowpass, kemudian sinyal output dari plant tersebut dipass ke blok PID yang diimplementasikan dengan fungsi berikut:

```
void PIDCompute(){
  if (!AutoKatz) {
    return;
  }
  error = setPoint - input;
  errorI += (ki*error);
  errorD = (filt_input-last_filt_input);

  ////
  if (errorI > outMax) {
    errorI = outMax;
  }
  else if (errorI < outMin) {
    errorI = outMin;
  }
  ////
  if (output > outMax) {
    output = outMax;
  }
  else if (output < outMin) {
    output = outMin;
  }
  else {
    output = (kp*error) + errorI - (kd*errorD);
  }
  lastinput = input;
  last_filt_input = filt_input;
}
```

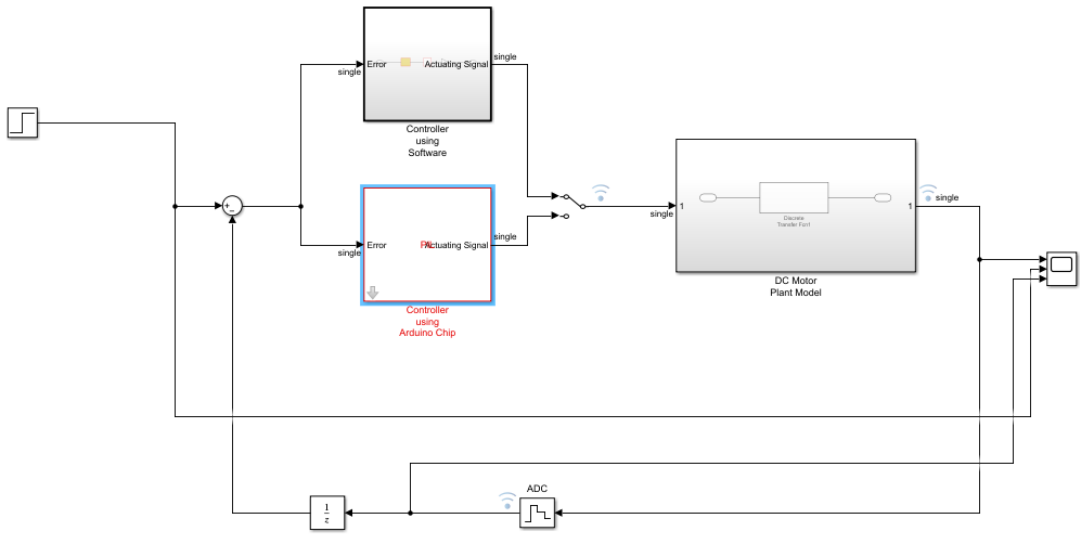
Fungsi PIDCompute() ini dipanggil saat interrupt terjadi dengan periode 5 ms. Selain fungsi PIDCompute(), ada juga beberapa fungsi yang dibuat yang digunakan untuk memperhalus plot dan menghilangkan sebisa mungkin spike-spike yang terjadi akibat perbedaan waktu sampling di arduino dan di simulink serta kenyataan bahwa baudrate memiliki error sesuai datasheet Atmega.

Untuk simulasi software saja, kita hanya perlu menyusun blok di SIMULINK sebagai berikut:



Gambar 10: Simulasi Software-in-the-Loop (SIL)

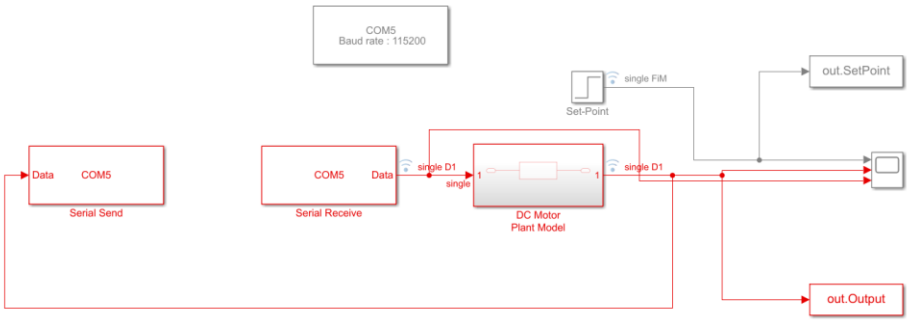
Sementara itu, pengujian dengan metode PIL dilakukan dengan menggunakan toolbox Embedded Coder yang dapat mengubah blok controller pada software SIMULINK menjadi executable code yang dapat dideploy pada chip Arduino Uno. Blok tersebut dapat dilihat pada gambar di bawah (berwarna merah):



Gambar 11: Simulasi Processor-in-the-Loop (PIL)

4.2 Simulator

Untuk simulator pada pengujian dengan HIL di sisi SIMULINK, beberapa blok digunakan dan disusun seperti gambar di bawah:



Gambar 12

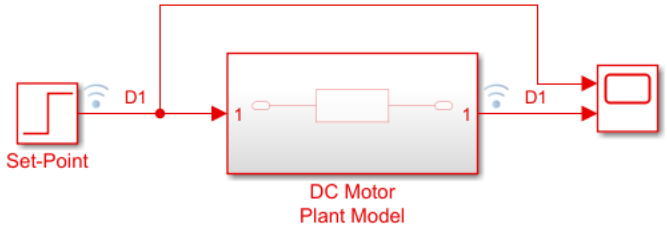
Jadi alurnya adalah sinyal hasil komputasi PID dikirim dari Arduino Uno ke SIMULINK dengan bantuan komunikasi serial pada blok Serial Receive. Dari situ sinyal actuating tersebut dipass ke blok DC Motor Plant Model dimana pada blok ini realisasi plant disimulasikan pada domain Z. Kemudian, sinyal output dari plant dikembalikan dengan feedback unity untuk selanjutnya dikirim kembali ke Arduino Uno dengan bantuan Serial Send. Komunikasi serial dilakukan pada nilai baudrate 115200.

5 Pengujian

5.1 Filter Digital

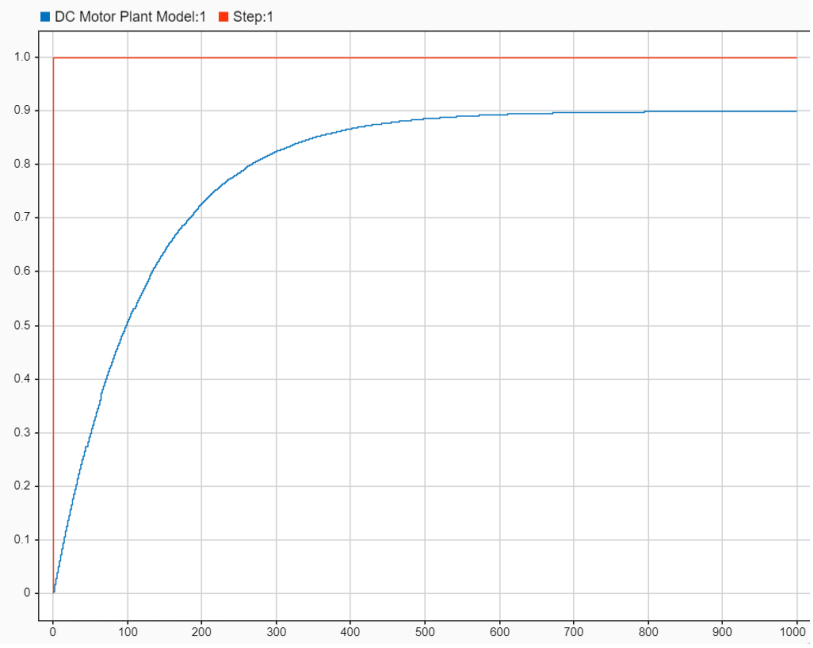
Filter Digital untuk plant direalisasikan pada SIMULINK dalam blok Discrete Transfer Function. Seperti yang sudah disebutkan pada bagian yang lain, persamaan diskret untuk plant ini didapatkan dengan melakukan transformasi Tustin pada fungsi transfer Motor DC yang diperoleh dari praktikum sistem kendali (parameter yang diobservasi merupakan hasil simulasi dengan Motor DC fisik di laboraturium).

Rangkaian Uji pada SIMULINK sebagai berikut:



Gambar 13: Rangkaian Testing untuk Model Plant

Pengujian filter tersebut menghasilkan hasil sebagai berikut:



Gambar 14: Plant Testing: Setpoint vs Output

Perhatikan bahwa sinyal output mempunyai error steady-state kurang lebih 0.1 satuan. Ini yang terjadi apabila sistem Motor DC dilakukan dengan konfigurasi open loop.

5.2 Komunikasi Data

Komunikasi data antara PC dan Arduino dilakukan secara serial menggunakan *port serial* yang dihubungkan dengan kabel *USB to USB B*. Data ditransmisikan dalam format blok data seperti yang telah dijelaskan pada bagian 2.1. Blok data terdiri dari 4 byte data dan dua buah *terminator,* yaitu '\r' dan '\n' (atau 13 dan 10 dalam ASCII) yang masing-masing memiliki panjang 1 *byte*. Oleh karena itu, satu blok data memiliki panjang total 6 *byte*.

Seperti yang telah dijelaskan pada bagian 2.3.2, untuk melakukan transmisi data dari Simulink ke Arduino menggunakan blok *serial send*. Sinyal *output plant* yang merupakan data bertipe *single* (4 *byte*) dan *terminator*-nya dikirim melalui *port serial* yang terhubung dengan Arduino (dalam kasus ini COM4). Data blok tersebut akan diterima oleh Arduino. Arduino sudah di-*upload* kode yang memiliki fungsi untuk menerima data blok dari PC. Namun karena data akan diolah dalam tipe *float* oleh Arduino maka perlu melakukan konversi dari *byte* ke *float*. Untuk melakukan hal tersebut maka didefinisikan sebuah tipe data *union* sebagai berikut.

```
typedef union{
      byte bytes[64];
      float number;
 } FLOATUNION_t;
```

Perlu diperhatikan bahwa *bytes* merupakan *array of byte* yang berukuran 64 untuk menyimpan data yang dikirim dari Simulink. Ukuran *array of byte* tersebut dibuat lebih besar daripada ukuran satu blok data, yaitu 6 *byte*. Hal tersebut bertujuan untuk antisipasi adanya data yang tidak tersimpan, namun sebenarnya ukuran *array of byte* tersebut dapat diperkecil.

Setelah dapat melakukan konversi data dari *byte* ke *float* maka Arduino sudah dapat menerima blok data dari Simulink. Hal tersebut dilakukan oleh fungsi *readFromMatlab* sebagai berikut.

```
float readFromMATLAB() {
      int reln = Serial.readBytesUntil('\r\n', buff, buffer_size);
      for (int i=0; i<buffer_size; i++) {
            myValue.bytes[i] = buff[i];
      }
      float out = myValue.number;
      return out;
}
```

Cara kerja fungsi di atas adalah membaca *byte* data sebelum terminator ('\r\n') dari *port serial* dan menyimpannya pada variabel buff yang merupaka *array of byte*. Selanjutnya, data yang masih dalam bentuk *byte* tersebut dipindahkan ke variabel yang bertipe `FLOATUNION_t` supaya selanjutnya dapat dikonversi menjadi *float*.

Data yang merupakan sinyal keluaran *plant* yang telah diterima dan dikonversi menjadi *float* selanjunya digunakan untuk perhitungan PID. Hasil perhitungan PID tersebut adalah sinyal kontrol yang akan dikirimkan dari Arduino ke Simulink. Oleh karena itu, sinyal kontrol tersebut perlu dikonversi dari tipe data *float* menjadi *byte*. Untuk mengirim *byte* sinyal kontrol tersebut menggunakan fungsi *writeToMATLAB* berikut ini.

```
void writeToMATLAB(float num) {
      byte *b = (byte *) &num;
```

```
        Serial.write(b, 4);
        Serial.write(13);
        Serial.write(10);
}
```

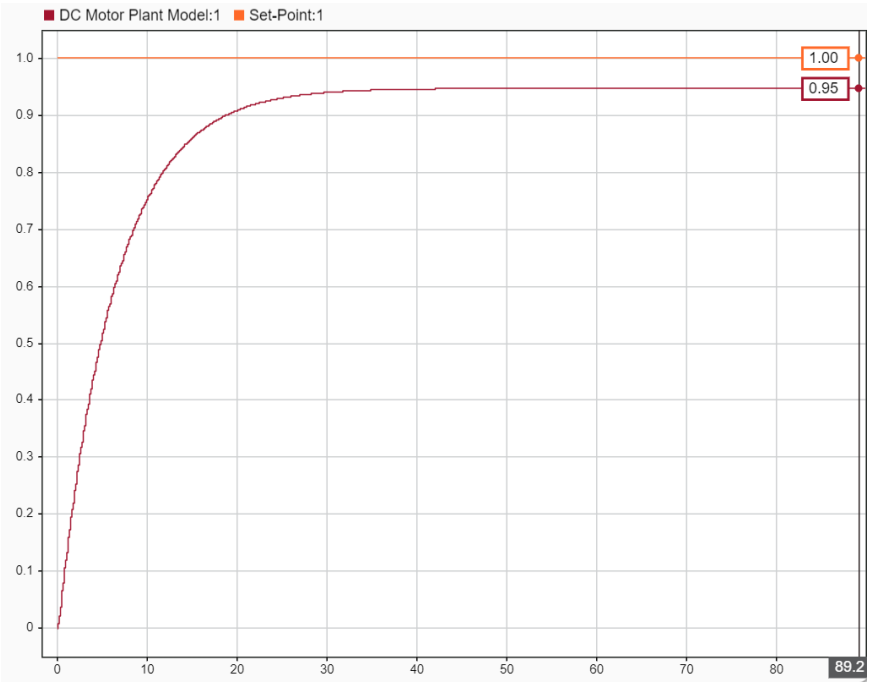
Cara kerja fungsi *writeToMATLAB* di atas adalah pertama mengubah sinyal kontrol dari *float* ke dalam bentuk *byte*. Kemudian mengirim data yang berukuran 4 *byte* tersebut ke Simulink melalui *port serial.* Setelah mengirim data, maka selanjutnya adalah mengirim *terminator*, yaitu '\r' dan '\n' atau dalam *integer* yang merepresenasikan karakter ASCII yaitu 13 dan 10. Data yang dikirim tersebut diterima oleh blok *serial receive* pada Simulink yang parameternya sudah diset sesuai dengan format data yang diinginkan.

5.3 Pengendali

Pada implementasinya, pengendali yang digunakan merupakan pengendali jenis P, I, D, PD, PI, dan PID. Jenis pengendali tersebut umum digunakan pada sistem yang tidak terlalu kompleks. Selain itu juga pengendali lebih mudah diimplementasikan dan dideploy ke hardware contohnya seperti Arduino.

5.3.1 Pengendali P

Pengujian dengan menggunakan metode HIL menghasilkan plot berikut:



Gambar 15: Simulasi HIL dengan pengendali P = 20

Dengan menggunakan pengendali P saja yang bernilai 20, diperoleh hasil seperti gambar di atas. Jika dibandingkan dengan yang diperoleh pada gambar 14 dimana konfigurasinya open loop, terlihat dengan pengendali P, parameter steady-state berubah. Steady-state error berkurang dari yang semula bernilai 0.1 menjadi di angka 0.05 yang mana sudah cukup baik. Observasi menunjukkan penambahan konstanta P lebih dari 20 tidak memberi nilai signifikan lagi. Sebagai tambahan, dapat diamati bahwa pengujian ini

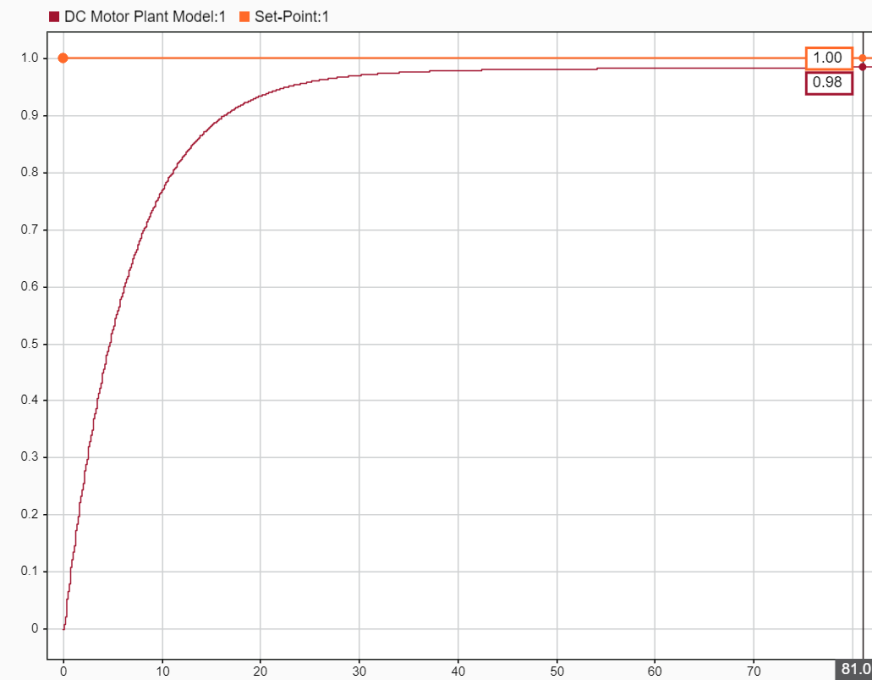
real-time sebab pengujian yang memberi plot pada gambar di atas merupakan hasil uji selama kurang lebih $1\frac{1}{2}$ menit waktu jam dinding.

5.3.2 Pengendali PI

Mari sekarang amati perubahan output yang terjadi akibat tidak hanya penggunaan pengendali proporsional, tetapi juga penambahan pengendali lain: integral. Nilai pengendali yang digunakan adalah:

$$K_P = 20$$
$$K_I = 0.1$$

Hasil yang diperoleh sebagai berikut:



Gambar 16: Simulasi HIL dengan pengendali P = 20 & I = 0.1

Dari kursor dapat disimpulkan bahwa error steady-state juga mengecil menjadi 0.02 tetapi tidak terlalu signifikan perubahannya bila dibandingkan dengan hasil saat pengendali P digunakan. Data parameter transien yang diperoleh dengan matlab menunjukkan hasil berikut:

```
      RiseTime: 15.5491
TransientTime: 42.6859
  SettlingTime: 42.6859
    SettlingMin: 0.9000
    SettlingMax: 0.9975
     Overshoot: 0
    Undershoot: 0
          Peak: 0.9975
      PeakTime: 82.8000
```

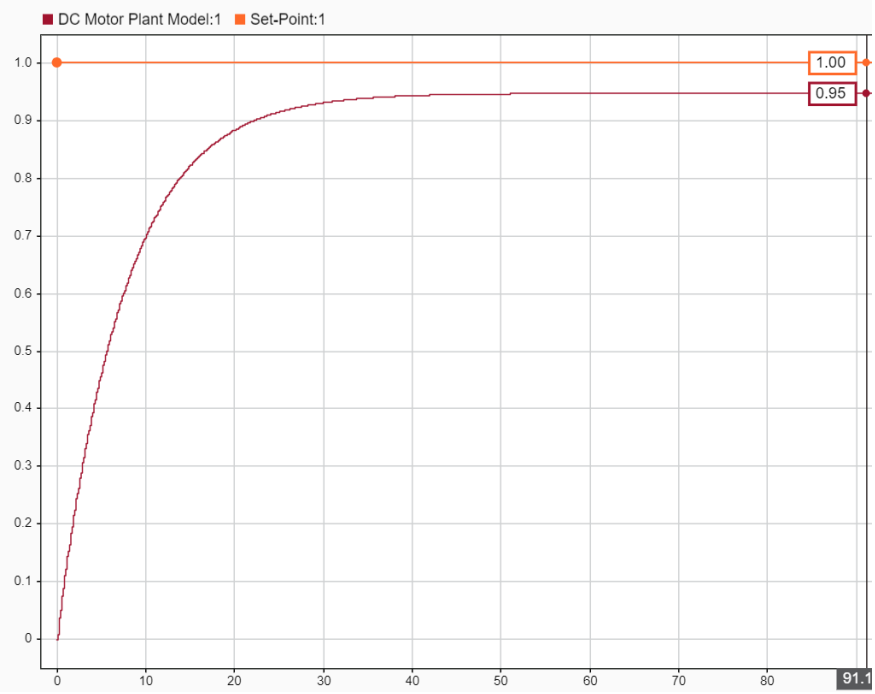
Gambar 17: Data Transien hasil pengendali PI

5.3.3 Pengendali PD

Sekarang pengendali yang digunakan adalah PD dengan nilai-nilai konstanta sebagai berikut:

$$K_P = 20$$
$$K_D = 25$$

Diperoleh hasil sebagai berikut:



Gambar 18: Simulasi HIL dengan pengendali P = 20 & D = 25

Dari kursor kita ketahui bahwa steady-state error tetap sama dengan yang didapat dengan hanya menggunakan kendali proporsional saja. Data transien dapat dilihat di bawah:

```
      RiseTime: 21.3619
TransientTime: NaN
 SettlingTime: NaN
   SettlingMin: 0.9002
   SettlingMax: 0.9473
     Overshoot: 0
    Undershoot: 0
          Peak: 0.9473
     PeakTime: 93.2000
```

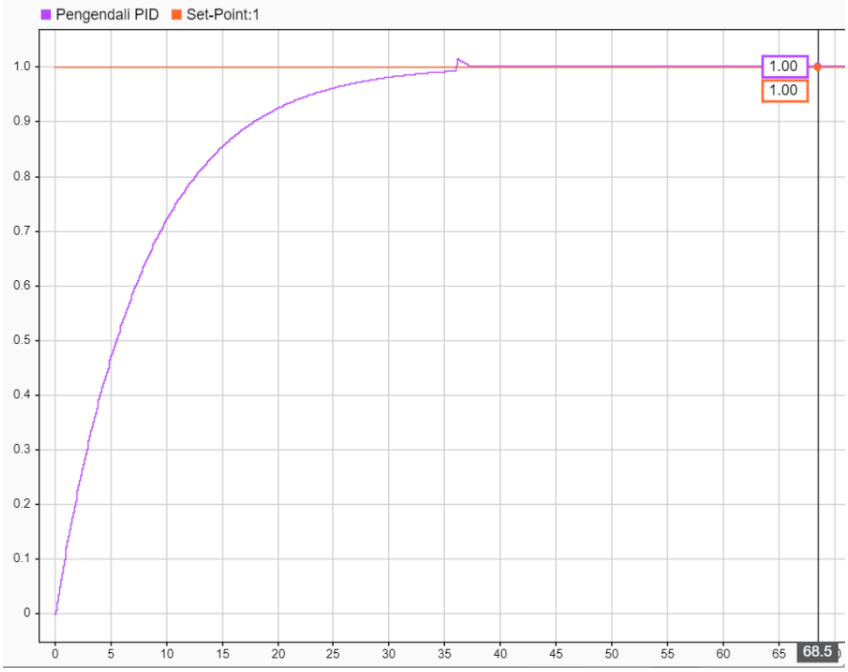
Gambar 19: Data Transien hasil pengendali PD

5.3.4 Pengendali PID

Untuk pengendali PID, konstanta yang digunakan adalah:

$$K_P = 20$$
$$K_D = 25$$
$$K_I = 0.1$$

Hasil yang diperoleh sebagai berikut secara real-time:



Gambar 20: Simulasi HIL dengan pengendali P = 20 & D = 25 & I = 0.15

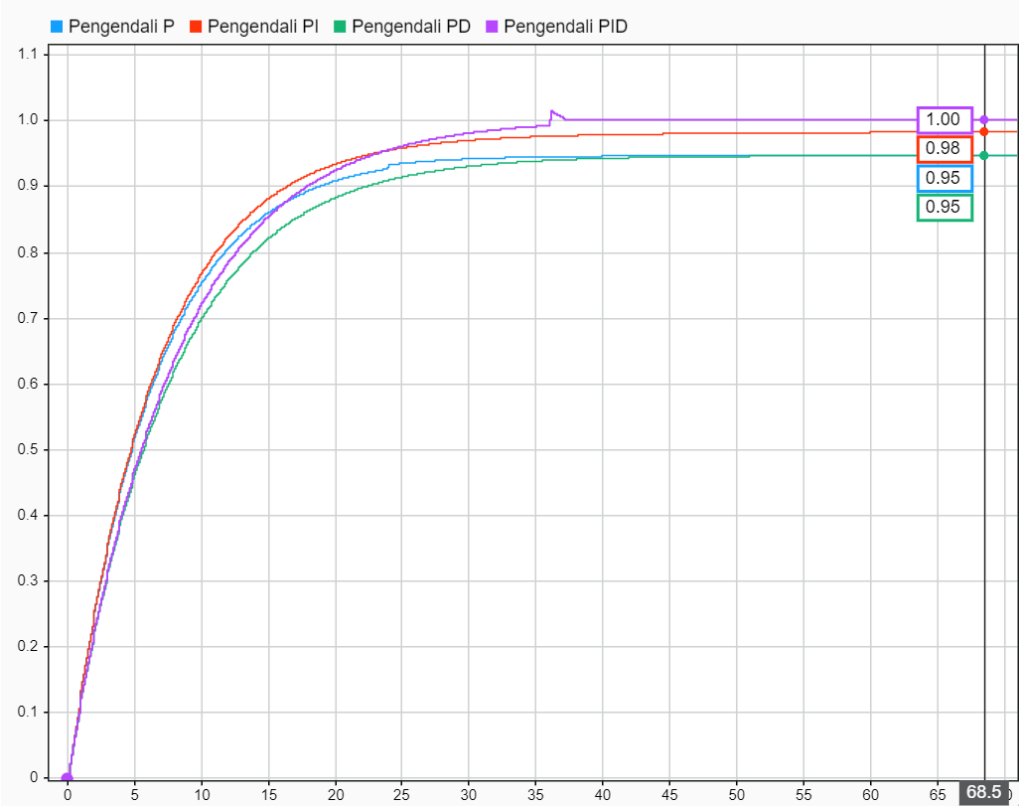
Sedikit spike terlihat bisa jadi karena kurang tepatnya periode sampling yang dipilih atau karena konflik dengan baudrate.

Data transien dari hasil pengendali PID diperoleh sebagai berikut:

```
      RiseTime: 16.9951
TransientTime: 29.3138
  SettlingTime: 29.3138
   SettlingMin: 0.9013
   SettlingMax: 1.0166
     Overshoot: 1.6568
    Undershoot: 0
          Peak: 1.0166
      PeakTime: 36.1000
```

Gambar 21: Data Transien hasil pengendali PID

Sebagai komparasi keempat jenis pengendali tersebut diplot pada gambar selanjutnya



Gambar 22: Komparasi Output dengan Variasi Pengendali P, PI, PD, dan PID

Terlihat bahwa pengendali PID lebih banyak menghasilkan keuntungan bagi sinyal output, baik dari sisi parameter transien maupun steady-state.

6 Kesimpulan

Ada beberapa kesimpulan yang diperoleh:

- Sistem dapat dimodelkan dalam bentuk filter lowpass dengan melakukan transformasi Tustin pada fungsi transfer dalam domain S menjadi dalam domain Z.
- Pengujian dilakukan dengan software saja, dengan metode PIL, dan metode HIL. Dari percobaan ditemukan bahwa proses pengendalian dilakukan secara real-time.
- Dari observasi dan percobaan, diperoleh bahwa dengan melakukan variasi konstanta P, I, dan D, maka parameter transien dan steady-state juga dapat berubah.

7 Referensi

1. Wescott, Tim., *PID Without a PHD,* Wescott Design Services, 2018.
2. https://www.mathworks.com/help/instrument/serialconfiguration.html, diakses pada 27 Mei 2022, pukul 12:30.
3. https://www.mathworks.com/help/instrument/serialsend.html, diakses pada 27 Mei 2022, pukul 12:50.

4. https://www.mathworks.com/help/instrument/serialreceive.html, diakses pada 27 Mei 2022, pukul 13:23.

8 Lampiran

8.1 Source Code

Source code dilampirkan di dokumen dan di github (share ke waskita@gmail.com)

A. Fungsi plotting_and_stepinfo()

```
function [] = plotting_and_stepinfo(out_obj)
%% Fungsi ini digunakan untuk mencari parameter transien dari sinyal output
%  serta melakukan plotting terhadap sinyal set-point dan sinyal output
%%
clc;

% Plotting
t = out_obj.tout;
setpoint = out_obj.SetPoint;
output = out_obj.Output;
% Tes apakah dimensi dari sumbu waktu dan sumbu vertikal sama
try
    tiledlayout(1,2);
    nexttile;
    plot(t,setpoint,'Color',[0,0.7,0.9]);
    xlabel('Time [second]');
    ylabel('Magnitude [Unit]');
    title('Set-Point');
    nexttile;
    plot(t,output,'Color',[0.4,0.5,0.88]);
    xlabel('Time [second]');
    ylabel('Magnitude [Unit]');
    title('Output/Response');

    % Output the transient response parameters
    param = stepinfo(output,t,1,0);
    disp(param);
% Jika tidak, maka masuk ke blok ini
catch ME
    close all;
    tiledlayout(1,2);
    nexttile;
    plot(t,setpoint,'Color',[0.9,0.05,0.3], 'LineWidth', 1.5);
    xlabel('Time [second]');
    ylabel('Magnitude [Unit]');
    title('Set-Point');
    nexttile;
    plot(t(1:(length(t)-1)),output,'Color',[0.4,0.5,0.88], 'LineWidth',
1.5);
    xlabel('Time [second]');
    ylabel('Magnitude [Unit]');
    title('Output/Response');
    % Output the transient response parameters
    param = stepinfo(output,t(1:(length(t)-1)),1,0);
    disp(param);
end
end
```

B. Kode PID di Arduino

```
#define outMax 50.0
#define outMin 0.0

// PID parameters
#define P 20
#define I 0.1
#define D 25
#define setPoint 1.0

bool manual_flag = false;
bool autokatz_flag = true;

//pengali PID 100
unsigned long int SampleTime = 100;

// Create a union to easily convert float to byte
typedef union{
  byte bytes[64];
  float number;
} FLOATUNION_t;

// Create the variable you want to send
FLOATUNION_t myValue;
const int buffer_size = 64;
byte buff[buffer_size];

// Create stored data from MATLAB
float input;
float output;

// Necessary variables
float error, errorI, errorD, prevError;
float kp, ki, kd;

//timer intrerrupt flag
volatile int timer_intr=0;

// filtered signal for reducing spike due to derivative term
float filt_input, last_filt_input, lastinput;

// automatic tuning
bool AutoKatz = false;

void SetTuningsOwn(float Kp, float Ki, float Kd)
{
  float SampleTimeInSec = ((float) SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTimeOwn(unsigned long int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
```

```
      float ratio  = (float) NewSampleTime/(float) SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long int) NewSampleTime;
   }
}

void SetModeOwn(bool Mode)
{
    bool newAutoKatz = (Mode == autokatz_flag);
    if(newAutoKatz && !AutoKatz) {
      InitializeOwn();
    }
    AutoKatz = newAutoKatz;
}

void InitializeOwn()
{
   lastinput = input;
   errorI = output;
   if(errorI > outMax) {
    errorI = outMax;
   }
   else if(errorI < outMin) {
    errorI = outMin;
   }
}

void PIDCompute(){
  if (!AutoKatz) {
    return;
  }
  error = setPoint - input;
  errorI += (ki*error);
  errorD = (filt_input-last_filt_input);

  ////
  if (errorI > outMax) {
    errorI = outMax;
  }
  else if (errorI < outMin) {
    errorI = outMin;
  }
  ////
  if (output > outMax) {
    output = outMax;
  }
  else if (output < outMin) {
    output = outMin;
  }
  else {
    output = (kp*error) + errorI - (kd*errorD);
  }
  lastinput = input;
  last_filt_input = filt_input;
}
```

```
// Register Settings
void RegSet() {
  cli();
  // Timer/Counter 1 initialization
  // Clock source: System Clock
  // Clock value: 15,625 kHz
  // Mode: CTC top=OCR1A
  // OC1A output: Disconnected
  // OC1B output: Disconnected
  // Noise Canceler: Off
  // Input Capture on Falling Edge
  // Timer Period: 5 ms
  // Timer1 Overflow Interrupt: Off
  // Input Capture Interrupt: Off
  // Compare A Match Interrupt: On
  // Compare B Match Interrupt: Off
  TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11)
| (0<<WGM10);
  TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (1<<WGM12) | (1<<CS12) |
(0<<CS11) | (1<<CS10);
  TCNT1H=0x00;
  TCNT1L=0x00;
  ICR1H=0x00;
  ICR1L=0x00;
  OCR1AH=0x00;
  OCR1AL=0x4E;
  OCR1BH=0x00;
  OCR1BL=0x00;

  // Timer/Counter 1 Interrupt(s) initialization
  TIMSK1=(0<<ICIE1) | (0<<OCIE1B) | (1<<OCIE1A) | (0<<TOIE1);

  sei();
}

ISR (TIMER1_COMPA_vect){
  timer_intr=1;
}

float readFromMATLAB() {
  int reln = Serial.readBytesUntil('\r\n', buff, buffer_size);
  for (int i=0; i<buffer_size; i++) {
    myValue.bytes[i] = buff[i];
  }
  float out = myValue.number;
  return out;
}

void writeToMATLAB(float num) {
  byte *b = (byte *) &num;
  Serial.write(b, 4);
  Serial.write(13);
  Serial.write(10);
}

void setup() {
  RegSet();
```

```
  // initialize serial, use the same baudrate in the Simulink Config block
  Serial.begin(115200);
  SetTuningsOwn(P,I,D);
  SetModeOwn(true);
}

void loop(){
  if (Serial.available() > 0) {
    input = readFromMATLAB();
    filt_input = (0.73310279*last_filt_input) + (0.1334486*input) +
(0.1334486*lastinput);

    if(timer_intr == 1){
      PIDCompute();
    }
    timer_intr = 0;
    writeToMATLAB(output);
  }
}
```