**MODUL PRAKTIKUM**

**MEKATRONIKA**



Oleh

Tim Dosen Teknik Komputer

**PROGRAM STUDI S1 TEKNIK KOMPUTER**

**UNIVERSITAS PENDIDIKAN INDONESIA**

**KAMPUS UPI DI CIBIRU**

**2025**

# **MODUL 3. SISTEM KENDALI RADIAL**

## **A. PENDAHULUAN**

Sistem radial merupakan salah satu prinsip dasar dalam distribusi energi atau pengaturan mekanis yang melibatkan gerakan berbentuk lingkaran atau sudut tertentu dengan acuan pusat (*radial point*). Dalam konteks teknik kontrol dan mekanika, istilah "radial" merujuk pada gerakan atau sistem yang mengacu pada sumbu pusat sebagai titik referensi. Sistem ini sering diaplikasikan pada pengendalian posisi, orientasi, atau distribusi energi dalam lingkup sudut tertentu.

Prinsip radial banyak digunakan dalam bidang elektronik, mekanik, dan otomasi. Pada mekanisme kontrol, sistem radial memanfaatkan aktuator seperti motor untuk menghasilkan gerakan melingkar, serta sensor untuk mendeteksi posisi atau sudut tertentu. Perangkat pengendali (seperti mikrokontroler) akan memproses data yang diterima dari sensor guna mengatur sistem secara presisi. Aplikasi sistem ini meliputi mesin CNC, robotika, radar antenna, hingga simulasi pengaturan posisi pada pendidikan dan pelatihan teknis.

Dalam sistem radial, tantangan utama adalah mencapai presisi, stabilitas, dan respons cepat. Saat suatu sistem diminta bergerak ke sudut tertentu, sistem harus:

1. Bergerak cepat menuju target (*minim delay*),
2. Berhenti tepat di sudut yang diinginkan (*tanpa overshoot*),
3. Tetap stabil meskipun ada gangguan, seperti gesekan mekanis atau perubahan beban.

PID (*Proportional-Integral-Derivative*) adalah algoritma kendali yang dapat menggabungkan tiga aksi dasar untuk memenuhi ketiga kriteria di atas:

1. *Proportional* (P): Menghasilkan respons sesuai kesalahan saat ini (misal: semakin besar selisih sudut motor dari target, semakin tinggi torsi yang diberikan).
2. *Integral* (I): Mengoreksi kesalahan akumulasi dari waktu ke waktu (misal: menghilangkan *steady-state error* pada posisi sudut akhir).
3. *Derivative* (D): Memprediksi kesalahan di masa depan berdasarkan laju perubahan sudut, seperti mengurangi kecepatan saat mendekati target untuk menghindari *overshoot*.

*Trainer Radial System Control* adalah alat peraga yang dirancang untuk simulasi dan pembelajaran sistem kontrol radial menggunakan prinsip PID. Alat ini menggunakan Motor DC/Servo 24 Volt, Arduino Mega 2560 sebagai pengendali utama, dan *Driver* Motor untuk mengatur gerakan. Dilengkapi LCD 16x2 untuk menampilkan data, sensor putaran untuk deteksi kecepatan, serta modul radial untuk simulasi sudut.



**Gambar 1.** Tampak atas *Trainer Radial System Control.*

## **B. STANDAR KOMPETENSI**

1. Mahasiswa mampu memahami PID pada sistem gerak radial.
2. Mahasiswa mampu mengoperasikan sistem gerak radial.

**C. KOMPETENSI DASAR**

1. Mahasiswa mampu memahami konsep dasar PID.
2. Mahasiswa mampu melakukan *tuning* sederhana PID untuk memahami pengaruh tiap komponen terhadap respons dinamis motor dari sistem gerak radial.

## **D. DASAR TEORI**

**1.1 Gerak Radial**

Sistem gerak radial adalah jenis gerak yang melibatkan pergerakan objek menjauh atau mendekat dari titik pusat sepanjang garis lurus yang dikenal sebagai radius. Dengan memanfaatkan kontrol Arduino, gerakan ini dapat diatur secara otomatis menggunakan berbagai komponen, seperti motor servo atau motor DC, serta sensor untuk mendeteksi posisi dan kecepatan objek. Contoh penerapan sistem ini adalah mekanisme pada lengan robotik yang bergerak radial untuk mengambil dan meletakkan objek pada jarak tertentu dari titik pusatnya.

Dalam implementasi kontrol Arduino, sensor seperti *rotary encoder* atau potensiometer digunakan untuk mendeteksi posisi radial, sementara motor digunakan untuk menghasilkan gerakan. Arduino berfungsi sebagai pengontrol utama, membaca data dari sensor, kemudian mengatur kecepatan dan arah motor bedasarkan algoritma tertentu. Sebagai contoh, dengan menggunakan motor servo, Arduino dapat memberikan sinyal PWM (*Pulse Width Modulation*) untuk menentukan sudut gerakan radial. Selain itu, kode program di Arduino IDE dapat dirancang untuk mengintegrasikan umpan balik (*feedback*) sehingga gerakan menjadi lebih presisi dan responsif. Sistem ini umum digunakan dalam teknologi robotik, otomasi, atau perangkat elektronik lain yang membutuhkan kontrol gerak berbasis radius.

**1.2 Konsep PID**

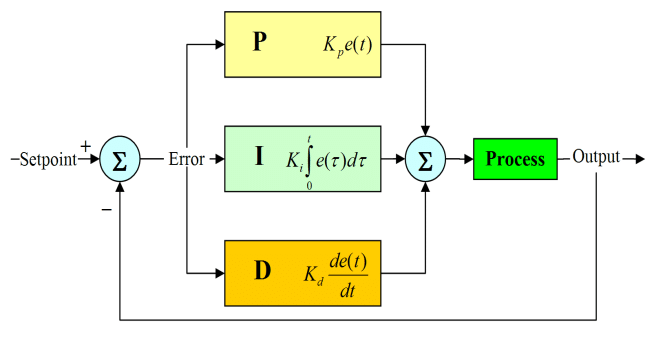
PID adalah pengendali umpan balik (*feedback controller*) metode yang banyak digunakan dalam *closed-loop system* (sistem lingkaran tertutup) untuk mengendalikan suatu proses, seperti suhu ruangan, kecepatan motor, atau posisi robot. Dalam sistem ini, keluaran (*output*) diukur terus-menerus dan dibandingkan dengan nilai yang diinginkan (*setpoint*). Selisih antara keduanya (*error*) digunakan untuk menghitung aksi koreksi yang dikirim ke sistem. Tujuan utamanya adalah meminimalkan *error* ini agar sistem bekerja sesuai target. Pada sistem linier, hubungan antara *input* dan *output* dapat dimodelkan dengan persamaan diferensial linier atau fungsi alih (*transfer function*), sehingga desain PID juga dapat dianalisis secara matematis. Secara matematis, sinyal kendali  dari PID dinyatakan sebagai:

di mana:

* ,
* *Kp*​, *Ki*​, dan *Kd*​ adalah gain untuk komponen *Proportional*, *Integral*, dan *Derivative*.

**1.2.1 Respons Proporsional**

Komponen Proportional menghasilkan aksi kendali yang sebandingdengan besarnya *error*saat ini, yaitu selisih antara nilai yang diinginkan (*setpoint*) dan nilai aktual sistem (*process variable*). Semakin besar *error*, semakin kuat respons yang diberikan. *Proportional Gain* (Kp) menentukan rasio antara respons keluaran (*output*) terhadap sinyal *error*. Sebagai contoh, jika *error* memiliki besar 10 dan Kp = 5, maka respons proporsional yang dihasilkan adalah . Semakin besar nilai Kp, semakin cepat respons sistem dalam mengurangi *error*. Namun, jika Kp terlalu besar, variabel proses (seperti posisi motor) akan mulai berosilasi di sekitar *setpoint*. Jika Kp terus dinaikkan, osilasi akan membesar dan sistem menjadi tidak stabil, bahkan mungkin kehilangan kendali.



**Gambar 2.** Diagram blok algoritma kontrol PID dasar.

**1.2.2 Respons Integral**

Komponen Integral menjumlahkan *error* dari waktu ke waktu. Akibatnya, bahkan *error* kecil yang bertahan lama akan menyebabkan komponen integral meningkat secara perlahan. Respons integral akan terus bertambah selama *error* tidak nol, sehingga efeknya adalah menghilangkan *steady-state error* (selisih akhir antara variabel proses dan *setpoint*). Namun, fenomena *integral windup* dapat terjadi ketika aksi integral menyebabkan saturasi kontroler (keluaran maksimum) tanpa mengurangi *error* menuju nol. Pada fenomena tersebut, ketika sistem melewati *setpoint* (*overshoot*), *error* menjadi negatif, sehingga integral mulai mengakumulasi nilai dalam arah berlawanan. Proses ini terjadi berulang dan dapat menyebabkan osilasi berkelanjutan (*overshoot* tanpa henti).

**1.2.3 Respons Derivatif**

Komponen Derivatif mengurangi keluaran (*output*) jika variabel proses (misalnya posisi motor) meningkat terlalu cepat. Respons ini sebanding dengan laju perubahan *error* (seberapa cepat *error* berubah). Semakin besar *Derivative Gain* (Kd), semakin kuat sistem bereaksi terhadap perubahan *error*, sehingga mempercepat respons keseluruhan. Namun, sebagian besar sistem kontrol menggunakan nilai Kd yang sangat kecil karena respons derivatif sangat peka terhadap *noise* pada sinyal variabel proses. Jika sinyal umpan balik sensor berisik atau laju *loop* kontrol terlalu lambat, respons derivatif bisa membuat sistem kontrol menjadi tidak stabil.

**1.3 Karakteristik Respons Sistem PID**

Dalam mengevaluasi kinerja sistem kendali, terutama sistem PID, terdapat beberapa karakteristik respons yang menjadi parameter penting untuk dianalisis. Karakteristik ini membantu menilai seberapa baik suatu sistem mencapai target yang diinginkan dan kestabilan sistem secara keseluruhan. Berikut adalah karakteristik respons utama yang perlu diperhatikan:

**1.3.1 Waktu Naik (*Rise Time*)**

Waktu naik adalah waktu yang dibutuhkan output sistem untuk naik dari 10% ke 90% dari nilai akhir yang diinginkan (*setpoint*). Parameter ini menunjukkan kecepatan respons sistem terhadap perubahan *input*. Sistem dengan waktu naik yang singkat menunjukkan respons yang cepat, namun kadang dapat menyebabkan *overshoot* yang besar jika tidak dikontrol dengan baik. Komponen Proporsional (P) dalam PID memiliki pengaruh signifikan terhadap waktu naik, di mana nilai Kp yang lebih tinggi cenderung mempercepat waktu naik.

**1.3.2 Lewatan Maksimum (*Overshoot*)**

*Overshoot* adalah fenomena ketika output sistem melampaui nilai setpoint sebelum akhirnya mencapai keadaan stabil. *Overshoot* dinyatakan dalam persentase dari nilai final dan dihitung dengan rumus:

di mana *>* adalah nilai maksimum output dan adalah nilai akhir pada keadaan stabil. Overshoot yang tinggi dapat menyebabkan stres pada komponen sistem dan perlu diminimalisir pada sistem yang memerlukan presisi tinggi. Komponen Derivatif (D) dalam PID berfungsi untuk meredam *overshoot*, sementara komponen Proporsional (P) dan Integral (I) yang terlalu tinggi dapat meningkatkan *overshoot*.

**1.3.3 Waktu Penetapan (*Settling Time*)**

Waktu penetapan adalah waktu yang diperlukan sistem untuk mencapai dan tetap berada dalam rentang tertentu (biasanya ±2% atau ±5%) dari nilai akhir yang diinginkan. Parameter ini mengindikasikan seberapa cepat sistem mencapai stabilitas. Sistem dengan waktu penetapan yang pendek lebih cepat mencapai keadaan stabil. Semua komponen PID mempengaruhi waktu penetapan, namun komponen Integral (I) dan Derivatif (D) memiliki pengaruh yang lebih signifikan dalam menentukan seberapa cepat osilasi sistem meredam.

**1.3.4 Kesalahan Keadaan Tunak (*Steady-State Error*)**

*Steady-state error* adalah perbedaan antara nilai *input* (*setpoint*) dengan nilai *output* sistem setelah respons transien hilang dan sistem mencapai keadaan stabil. Kesalahan ini dihitung dengan rumus:

di mana *r* adalah nilai referensi (*setpoint*) dan adalah nilai *output* pada keadaan stabil. Komponen Integral (I) dalam PID dirancang untuk mengeliminasi *steady-state error* dengan mengakumulasi dan mengoreksi *error* seiring waktu.

**1.3.5 Hubungan Karakteristik Respons dengan Parameter PID**

Tabel di bawah ini menunjukkan pengaruh umum dari masing-masing parameter PID terhadap karakteristik respons sistem:

**Tabel 1.** Pengaruh Umum Parameter PID dengan Karakteristik Respons Sistem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | ***Rise Time*** | ***Overshoot*** | ***Settling Time*** | ***Steady-State Error*** |
| Kp naik | Menurun | Meningkat | Sedikit Berubah | Menurun |
| Ki naik | Sedikit Menurun | Meningkat | Meningkat | Eleminasi |
| Kd naik | Sedikit Berubah | Menurun | Menurun | Tidak berpengaruh langsung |

Perlu dicatat bahwa hubungan ini bersifat umum dan dapat bervariasi tergantung pada karakteristik spesifik sistem yang dikendalikan. Dalam praktiknya, penalaan (*tuning*) parameter PID memerlukan penyesuaian berdasarkan respons aktual sistem dan persyaratan kinerja yang diinginkan.

**1.4 Tuning Manual Sistem PID**

Tuning manual parameter PID dapat dilakukan dengan menganalisis grafik respons sistem orde dua (*step response*) yang menampilkan perbandingan antara nilai *setpoint* dan nilai aktual sistem. Pada sistem berbasis Arduino, grafik ini dihasilkan melalui *serial plotter* Arduino IDE dengan melakukan *print* nilai *setpoint* dan *output* secara real-time. Dengan mengamati bentuk grafik dapat diidentifikasi kekurangan kinerja sistem dan penyesuaian parameter PID (Kp, Ki, Kd) sesuai kebutuhan. Beberapa contoh kriteria analisis grafik divisualisasikan pada Gambar 3 dan dijabarkan sebagai berikut.

1. ***Rise Time* Lambat**

Kenaikan nilai aktual terlalu landai menuju *setpoint* karena Kp terlalu rendah. Tingkatkan Kp untuk mempercepat respons, tetapi hindari berlebihan agar tidak memicu osilasi.

1. ***Overshoot* Tinggi**

Nilai aktual melewati setpoint akibat Kp terlalu tinggi atau kurangnya peredaman. Kurangi Kp atau tambahkan Kd untuk meredam kecepatan saat mendekati target.

1. **Osilasi Berkepanjangan**

Gelombang berulang di sekitar setpoint muncul jika Ki/Kp terlalu tinggi atau sistem kurang stabil. Kurangi Ki/Kp atau tambahkan Kd untuk meredam osilasi.

1. ***Settling Time* Panjang**

Waktu stabilisasi lama terjadi jika osilasi belum teredam atau ada steady-state error. Optimalkan Kd (mengurangi osilasi) atau Ki (menghilangkan error residual).

1. ***Steady-State Error***

Selisih tetap antara *setpoint* dan nilai akhir menunjukkan Ki terlalu rendah. Naikkan Ki untuk mengoreksi akumulasi *error*, tetapi jangan berlebihan agar tidak memicu osilasi.



**Gambar 3.** Respons Sistem PID pada Beberapa Kondisi.

**E. Perangkat Praktikum**

**1.1 Alat dan Bahan Percobaan**

1. *Trainer Radial System Control*
2. Kabel Power 220v
3. Kabel USB 2.0
4. Laptop

**1.2 Kebutuhan Percobaan dan Analisis**

1. *Script* MATLAB untuk *tuning* secara manual yang bisa diambil melalui *link* berikut.

<https://github.com/andyathsid/mechatronic-lab-module/blob/main/code/matlab/manual-tuning/manual_tuning_radial.m>

1. Program MATLAB.

**Catatan**: *script* sudah diuji pada versi MATLAB r2023b dan r2024a dengan *toolbox* lengkap.

**F. Keselamatan Kerja**

1. Gunakanlah pengaman saat melakukan praktik.
2. Gunakanlah alat dan bahan sesuai dengan fungsinya.
3. Jangan makan, minum ataupun bercanda saat praktik.
4. Jangan sembarang menyalakan alat yang ada tanpa mengetahui cara mengoperasikannya.
5. Apabila ada kesulitan konsultasikan dengan instruktur.

**G. Percobaan**

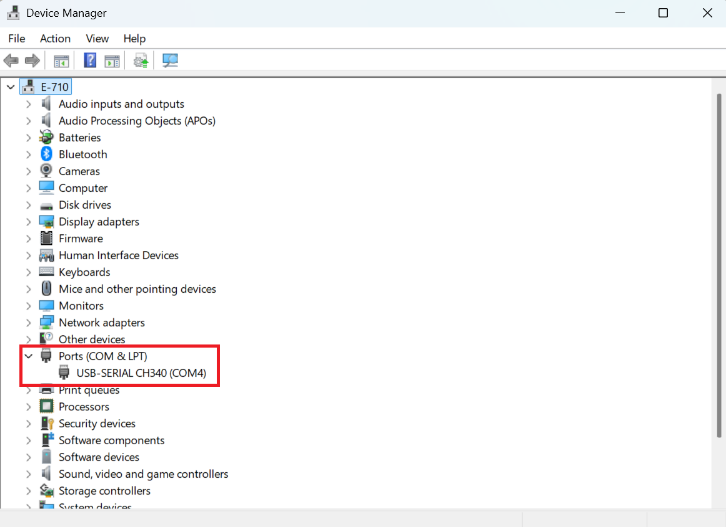
1. **Persiapan:**

* Pastikan *Trainer* sudah menyala dan LCD sudah menampilkan “*Trainer Radial System Control*”.
* *Setting setpoint* yang akan dituju dengan memutar potensiometer. Atur *setpoint* sesuai dengan urutan kelompok dan kelas pada Tabel 2.

**Tabel 2.** Nilai Setpoint Sesuai Urutan Kelompok dan Kelas

|  |  |  |
| --- | --- | --- |
| **Kelas** | **Urutan** | ***Setpoint*** |
| A | 1 | 100 |
| 2 | 50 |
| 3 | 110 |
| 4 | 65 |
| 5 | 120 |
| 6 | 75 |
| 7 | 130 |
| 8 | 85 |
| 9 | 140 |
| 10 | 95 |
| B | 1 | 45 |
| 2 | 105 |
| 3 | 60 |
| 4 | 115 |
| 5 | 70 |
| 6 | 125 |
| 7 | 80 |
| 8 | 135 |
| 9 | 90 |
| 10 | 145 |

* Lakukan *setting* Kp, Ki, dan Kd dengan menekan tombol “*Start*” untuk mengganti parameter, kemudian ganti nilai dengan menekan tombol “*Count* +” dan “*Count* –“.
* Tekanl tombol “*Reset”* untuk mengembalikan ke posisi semula.
* Hubungkan *Trainer ke* laptopmenggunakan kabel USB 2.0.
* Periksa *port* COM Arduino yang digunakan melalui *Device Manager* (misal: COM1, COM2, dst).



**Gambar 4.** Contoh Arduino yang terhubung ke *port* COM4.

* Sesuaikan nilai variable `arduinoPort` dari *script* MATLAB pada baris ke-6.

1. clear all;

2. close all;

3. clc;

4.

5. % Setup serial connection

6. arduinoPort = 'COM4';  % Change to your Arduino COM port

7. baudRate = 9600;       % Make sure this matches your Arduino setting

8.

* Jalankan *script* menggunakan program MATLAB dan pastikan **Kp**, **Ki**, dan **Kd** bernilai 0*.* Periksa terminal hingga muncul *output* sebagai berikut.

=====================================

PID Performance Measurement - Test #1

=====================================

Instructions:

1. Make sure your system is reset

2. When ready to measure performance metrics, type "ready" and press Enter

3. Then press the START button on your hardware to begin the test

4. The script will collect data for 60 seconds and calculate performance metrics

Type "ready" when you want to start the test:

1. ***Tuning*:**
   1. **Uji Kp:**
      * Pertahankan **Ki** = 0, **Kd** = 0.
      * Naikkan **Kp** secara bertahap (misal: 1 → 2 → 3 atau kelipatan 5).

* Tekan tombol *reset* setelah selesai mengganti parameter PID, pastikan *setpoint* sudah di-*setting*.
* Masukkan *input “*ready” di terminal MATLAB sesuai instruksi yang diberikan.

=====================================

PID Performance Measurement - Test #1

=====================================

Instructions:

1. Make sure your system is reset

2. When ready to measure performance metrics, type "ready" and press Enter

3. Then press the “START” button on your hardware to begin the test

4. The script will collect data for 60 seconds and calculate performance metrics

Type "ready" when you want to start the test: ready

* Tekan tombol “Start” pada *Trainer*. *Script* akan otomatis mendeteksi perubahan posisi dan memulai mengambil data selama 60 detik.

=====================================

PID Performance Measurement - Test #1

=====================================

Instructions:

1. Make sure your system is reset

2. When ready to measure performance metrics, type "ready" and press Enter

3. Then press the START button on your hardware to begin the test

4. The script will collect data for 60 seconds and calculate performance metrics

Type "ready" when you want to start the test: ready

Serial buffer flushed. Previous test data cleared.

Press the START button on your hardware now!

Waiting for position to change...

Movement detected! Starting timer and calculations...

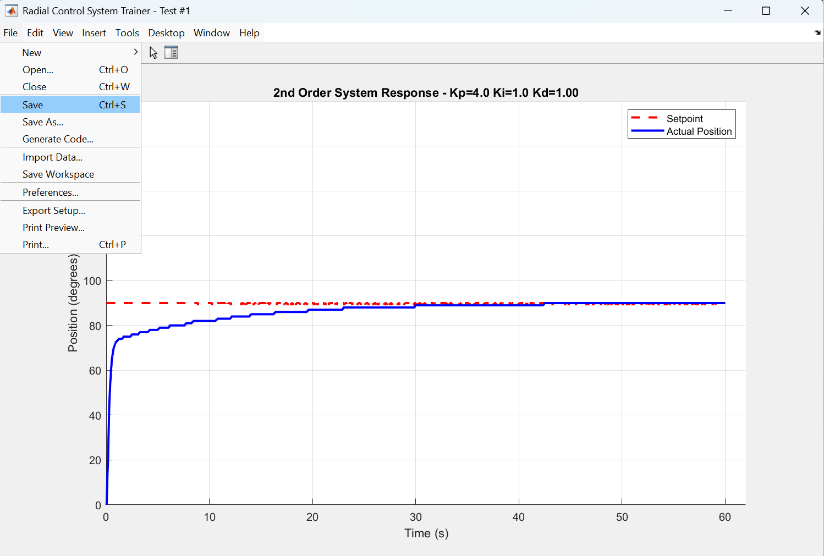
Collecting data...

Test duration complete. Calculating metrics...

* Catat nilai *Rise Time*, *Settling Time, Overshoot,* dan *Steady-State Error* pada tabel di Lampiran 1.

|  |
| --- |
| =======================================  PID Controller Performance Metrics:  =======================================  Current PID values: Kp=4.0, Ki=1.0, Kd=1.00  Rise Time (10% to 90%): 7.596 seconds  Maximum Overshoot: 0% (No overshoot detected)  Settling Time (within 2%): 29.930 seconds  Steady State Error: 0.00 degrees (0.00%)  Overall Performance Score: 2.6472 (lower is better)  Test complete.  Do you want to run another test? (y/n): |

* Simpan grafik respon sistem pada parameter yang diuji dengan format PNG atau yang lainnya.



**Gambar 5.** Contoh Grafik Hasil Percobaan

* Ulangi percobaan uji **Kp** hingga mendapatkan hasil untuk total 5 paramater **Kp** yang berbeda dengan masukkan *input* “y” pada terminal sesuai dengan instruksi yang diberikan.
* Tentukan nilai **Kp** yang terbaik.
  1. **Uji Ki:**
     + Pertahankan **Kp** yang menghasilkan respons terbaik dari uji sebelumnya.
     + Tambahkan **Ki** secara bertahap (misal: 0.1 → 0.2 → 0.3 atau kelipatan 0.5).
     + Ulangi dan sesuaikan sisa langkah-langkah serupa yang dilakukan pada percobaan a. Uji Kp untuk mendapatkan nilai **Ki** yang terbaik
  2. **Uji Kd:**
     + Pertahankan **Kp** dan **Ki** yang telah diatur.
     + Tambahkan **Kd** secara bertahap (misal: 0.5 → 1 → 1.5 atau kelipatan 0.5).
     + Ulangi dan sesuaikan sisa langkah-langkah serupa yang dilakukan pada percobaan sebelumnya untuk mendapatkan nilai **Kd** yang terbaik.

**H. Analisis Hasil**

1. Berdasarkan data pengamatan, tentukan kombinasi Kp, Ki, Kd yang paling stabil dan jelaskan alasannya.
2. Jelaskan fenomena yang menghambat (misalnya: *rise time* lambat, osilasi, sering terjadi *overshoot*) dalam mencapai *setpoint.*
3. Diskusikan *trade-off* antara kecepatan respon (Kp) dan stabilitas (Kd/Ki).
4. Analisis dalam bentuk tabel pengaruh *tuning* ketika setiap parameter PID dinaikkan mulai dari Kp hingga Ki dan mulai dari nilai 0 pada seperti pada Tabel 1.

**LAMPIRAN**

Lampiran 1. Data Observasi *Tuning* PID dari Percobaan Praktikum Sistem Kendali Radial

1. **Identitas Kelompok**

Kelas (A/B) :

Kelompok (1, 2, 3, dst) :

Anggota :

|  |  |
| --- | --- |
| **NIM** | **Nama** |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Tabel Observasi Pengaruh Parameter PID pada Respons Sistem**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Kp** | **Ki** | **Kd** | **Rise Time (detik)** | **Settling Time (detik)** | **Overshoot**  **(%)** | **Steady-State Error**  **(%)** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |