# Dasar Sistem Kendali dan Implementasinya dengan Python

Simulasi dan Aplikasi Praktis dengan Jupyter Notebook, python-control dan Webots untuk Sistem Kendali PID

Ardy Seto Priambodo, Aris Nasuha, Oktaf Agni Dhewa, Ahmad Taufiq Musaddid

# Daftar Isi

# 1 Tutorial

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



## 1.1 Referensi gambar dan tabel

Gambar 1.1 merupakan contoh gambar. Tabel 1.1 adalah contoh tabel.

## 1.2 Memasukkan gambar

**Gambar 1.1:** Contoh ini Caption Gambar

## 1.3 Memasukkan tabel

**Tabel 1.1:** Contoh ini Caption Tabel

| Komponen | Deskripsi Singkat |
|---|---|
| Sensor | Mengukur proses |
| Kontrol | Mengolah error dan menghasilkan aksi kendali |
| Aktuator | Menggerakkan plant sesuai sinyal kendali |



## 1.4 Membuat `note`, `tip`, `warning`, `caution`, dan `important`

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam aliquet libero quis lectus elementum fermentum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam aliquet libero quis lectus elementum fermentum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam aliquet libero quis lectus elementum fermentum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam aliquet libero quis lectus elementum fermentum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam aliquet libero quis lectus elementum fermentum.

## 1.5 Rumus matematika

Penulisan rumus inline $e(t) = \mathrm{SP} - \mathrm{PV}$. Penulisan rumus dalam baris baru:

$$e(t) = \mathrm{SP} - \mathrm{PV}$$

## 1.6 Kode program

Penulisan kode program dalam baris `print("Hello, World!")`. Penulisan kode program dalam blok:

```python
def prime_numbers(n):
    """Mengembalikan daftar bilangan prima hingga n."""
    primes = []
    for num in range(2, n + 1):
        is_prime = True
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            primes.append(num)
    return primes
print(prime_numbers(20))
```

## 1.7 Teks, link, dan daftar

Penulisan bold **bold** dan italic *italic*. Pembuatan link Google. penulisan list dengan tanda −:

- Item pertama
- Item kedua

- Item ketiga

Penulisan list dengan angka:

1. Item pertama
2. Item kedua
3. Item ketiga

## 1.8 Contoh hasil perplexity ai

## 1.9 Cara Kerja PID Control

**Pengertian Dasar** Kontrol PID (Proportional-Integral-Derivative) adalah algoritma kontrol umpan balik yang sangat umum digunakan di industri untuk menjaga suatu variabel proses (seperti suhu, tekanan, kecepatan) agar tetap pada nilai yang diinginkan (setpoint). Kontroler PID bekerja dengan membandingkan nilai aktual (feedback) dengan nilai setpoint, kemudian menghitung selisihnya (error), dan menghasilkan sinyal kontrol berdasarkan tiga komponen utama: Proporsional (P), Integral (I), dan Derivatif (D)[1][3][2].

## 1.10 Komponen Utama PID

**1. Proporsional (P)** - Komponen proporsional menghasilkan output yang sebanding dengan besarnya error saat ini. - Semakin besar error, semakin besar pula respon kontrol yang diberikan. - Fungsi utamanya adalah mempercepat respon sistem terhadap perubahan setpoint, tetapi tidak selalu menghilangkan error sepenuhnya[1][2][3][7].

**2. Integral (I)** - Komponen integral mengakumulasi error dari waktu ke waktu. - Berfungsi untuk mengatasi error yang tersisa (offset) yang tidak bisa dihilangkan oleh komponen proporsional saja. - Integral akan terus menambah output hingga error benar-benar nol, sehingga sistem menjadi lebih akurat[1][2][3][7].

**3. Derivatif (D)** - Komponen derivatif merespons terhadap laju perubahan error. - Membantu mengantisipasi perubahan cepat dan meredam osilasi atau overshoot, sehingga sistem menjadi lebih stabil. - Derivatif memberikan aksi koreksi sebelum error menjadi terlalu besar[1][2][3][7].

## 1.11 Cara Kerja Secara Umum

- Kontroler PID secara terus-menerus membaca nilai proses aktual dari sensor.
- Menghitung error sebagai selisih antara setpoint dan nilai aktual.
- Setiap komponen (P, I, D) memberikan kontribusi pada sinyal kontrol berdasarkan error dan karakteristiknya masing-masing.
- Sinyal kontrol ini kemudian digunakan untuk mengatur aktuator (misal, membuka katup, mengatur kecepatan motor, dll) agar variabel proses kembali ke setpoint[1][2][3][6].

**Rumus PID:**

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- $u(t)$: Output kontroler
- $e(t)$: Error (setpoint - nilai aktual)
- $K_p$: Konstanta proporsional
- $K_i$: Konstanta integral
- $K_d$: Konstanta derivatif[3]



## 1.12 Ilustrasi Sederhana

Misal pada sistem pemanas ruangan: - Setpoint suhu: 25°C - Sensor membaca suhu aktual: 23°C (error = 2°C) - Komponen P akan langsung menaikkan daya pemanas sesuai besarnya error. - Komponen I akan terus menambah daya jika error bertahan lama, hingga suhu benar-benar mencapai 25°C. - Komponen D akan mengurangi daya pemanas jika suhu naik terlalu cepat, untuk mencegah suhu melewati 25°C (overshoot)[1][2][3].

## 1.13 Penyesuaian (Tuning) PID

Agar kontrol PID bekerja optimal, nilai parameter

$$K_p$$

,

$$K_i$$

, dan

$$K_d$$

harus disesuaikan (tuning) sesuai karakteristik sistem yang dikendalikan (plant). Penyetelan ini penting agar respon sistem cepat, akurat, dan stabil[3][5].

## 1.14 Aplikasi

PID banyak digunakan di berbagai bidang, seperti: - Pengendalian suhu - Otomatisasi industri - Kontrol kecepatan motor - Proses kimia dan ilmiah[1][2][6]

## 1.15 Kesimpulan

PID control bekerja dengan mengoreksi error antara nilai aktual dan setpoint secara terus-menerus melalui tiga aksi utama: proporsional untuk respon cepat, integral untuk menghilangkan error sisa, dan derivatif untuk menstabilkan sistem dari perubahan cepat. Kombinasi ketiganya memungkinkan pengendalian yang presisi dan stabil pada berbagai sistem otomasi industri[1][2][3][7].

# 2 Pengenalan Sistem Kendali

## 2.1 Dasar Sistem Kendali

Sistem kendali adalah cabang ilmu teknik yang mempelajari bagaimana mengatur perilaku suatu sistem agar dapat beroperasi sesuai dengan tujuan yang diinginkan. Dalam kehidupan sehari-hari, sistem kendali banyak ditemukan pada perangkat elektronik, mesin industri, kendaraan, hingga sistem otomasi rumah tangga.

Pada dasarnya, sistem kendali bertujuan untuk menjaga variabel keluaran (output) suatu proses tetap pada nilai yang diharapkan meskipun terjadi gangguan atau perubahan pada lingkungan sekitarnya. Untuk mencapai tujuan ini, sistem kendali memanfaatkan prinsip umpan balik (feedback), di mana hasil keluaran diukur dan dibandingkan dengan nilai referensi, kemudian digunakan untuk memperbaiki aksi sistem.

Komponen utama dalam sistem kendali meliputi:

- Sensor: Mengukur variabel proses, seperti suhu, tekanan, kecepatan, atau posisi.
- Kontroller: Mengolah data dari sensor dan menentukan aksi yang diperlukan.
- Aktuator: Melaksanakan aksi berdasarkan perintah dari kontroller, misalnya menggerakkan motor atau membuka katup.
- Plant: Objek atau proses yang dikendalikan, seperti mesin, kendaraan, atau sistem pemanas.

Sistem kendali dapat diklasifikasikan menjadi dua jenis utama:

- Sistem Kendali Terbuka (Open-Loop): Tidak menggunakan umpan balik. Kontroller memberikan aksi berdasarkan input tanpa memperhatikan hasil keluaran. Contoh: mesin cuci yang beroperasi dengan waktu tetap tanpa memperhatikan kebersihan cucian.
- Sistem Kendali Tertutup (Closed-Loop): Menggunakan umpan balik. Kontroller memperbaiki aksi berdasarkan perbandingan antara keluaran aktual dan refe-

rensi. Contoh: AC otomatis yang mengatur suhu ruangan berdasarkan sensor suhu.

Salah satu metode pengendalian tertutup yang paling banyak digunakan adalah kendali PID (Proportional-Integral-Derivative). Kendali PID bekerja dengan mengombinasikan tiga aksi koreksi, yaitu proporsional (P), integral (I), dan derivatif (D), untuk menghasilkan respons sistem yang stabil dan sesuai harapan. PID digunakan luas dalam pengaturan suhu, kecepatan motor, posisi lengan robot, hingga sistem otomasi industri.

## 2.2 Simulasi dan Implementasi Praktis Sistem Kendali

Perkembangan teknologi digital memungkinkan sistem kendali tidak hanya dirancang secara teoritis, tetapi juga dapat disimulasikan dan diuji secara virtual sebelum diterapkan pada perangkat nyata. Simulasi sistem kendali sangat penting dalam proses pembelajaran dan pengembangan karena beberapa alasan berikut:

- Memudahkan pemahaman konsep dengan visualisasi perilaku sistem.
- Mengurangi risiko kerusakan perangkat fisik akibat kesalahan desain atau pengaturan parameter.
- Mempercepat proses iterasi desain dan optimasi sistem kendali.

Beberapa platform dan perangkat lunak yang banyak digunakan untuk simulasi dan implementasi sistem kendali antara lain:

- Jupyter Notebook: Lingkungan pemrograman interaktif yang memudahkan penulisan kode, dokumentasi, dan visualisasi hasil simulasi dalam satu tempat.
- python-control: Pustaka Python khusus untuk analisis dan desain sistem kendali linier, menyediakan berbagai fungsi untuk pemodelan, simulasi, serta analisis respons sistem.
- Webots: Simulator robotika yang memungkinkan pengguna menguji algoritma kendali pada model robot secara virtual, termasuk integrasi dengan Python untuk pengembangan kendali cerdas dan aplikasi robotika.

Dengan simulasi, mahasiswa dan dosen dapat melakukan eksperimen virtual, seperti mengatur parameter PID, mengamati respons sistem terhadap gangguan, atau menguji strategi kendali pada robot dan sistem fisik lain tanpa perlu perangkat keras mahal.

## 2.3 Bahasa Pemrograman Python untuk Sistem Kendali

Python menjadi bahasa pemrograman pilihan utama dalam pengembangan dan simulasi sistem kendali karena beberapa keunggulan:

- Sintaks yang sederhana dan mudah dipelajari, cocok untuk mahasiswa dan pemula.
- Dukungan pustaka yang sangat luas, mulai dari analisis numerik (NumPy, SciPy), visualisasi (Matplotlib), hingga pustaka khusus sistem kendali seperti python-control.
- Integrasi yang baik dengan berbagai platform simulasi, seperti Jupyter Notebook dan Webots.
- Komunitas pengguna yang besar sehingga mudah menemukan referensi, tutorial, dan bantuan.

Dalam konteks pembelajaran sistem kendali, Python memungkinkan mahasiswa untuk:

- Memodelkan sistem fisik dalam bentuk persamaan matematis atau diagram blok.
- Melakukan simulasi respons sistem terhadap berbagai masukan dan gangguan.
- Mengimplementasikan algoritma kendali, seperti PID, dan menguji performanya secara langsung.
- Mengembangkan aplikasi kendali pada robot virtual maupun perangkat nyata melalui integrasi dengan sensor dan aktuator.

Penggunaan Python juga memudahkan dosen dalam menyusun materi ajar, membuat contoh kasus, serta memberikan tugas praktikum yang dapat diakses dan dijalankan oleh mahasiswa di berbagai perangkat.



## 2.4 Ruang Lingkup Buku

Buku ini disusun untuk membantu dosen dan mahasiswa memahami konsep dasar sistem kendali serta menguasai keterampilan praktis dalam simulasi dan implementasi menggunakan Python. Materi disusun secara sistematis, mulai dari teori dasar, pemodelan matematis, analisis sistem, hingga penerapan algoritma kendali PID pada berbagai kasus nyata.

Ruang lingkup utama buku ini meliputi:

- Pengenalan konsep dan komponen sistem kendali.
- Pemodelan dan analisis sistem linier.
- Desain dan tuning kendali PID.
- Simulasi sistem kendali menggunakan python-control dan Jupyter Notebook.
- Implementasi algoritma kendali pada robot dan sistem fisik menggunakan Webots.
- Studi kasus dan latihan praktis untuk memperkuat pemahaman dan keterampilan.

Setiap bab dilengkapi dengan penjelasan teoritis, contoh kode, visualisasi hasil simulasi, serta tugas mandiri yang dirancang untuk mengasah kemampuan analisis dan pemrograman mahasiswa. Dengan pendekatan ini, diharapkan pembaca tidak hanya memahami konsep, tetapi juga mampu menerapkan pengetahuan dalam menyelesaikan masalah nyata di bidang sistem kendali dan otomasi.

# 3 Persiapan Simulasi Berbasis Python

Sebelum memulai simulasi dan implementasi sistem kendali, sangat penting untuk menyiapkan lingkungan pemrograman Python yang stabil, terstruktur, dan mudah digunakan. Pada bab ini, Anda akan belajar cara menginstal Python menggunakan Anaconda (conda), mengelola virtual environment, menginstal library utama, serta penggunaan dasar library seperti NumPy, Matplotlib, Jupyter Notebook, dan python-control. Penjelasan disertai contoh kode yang sering digunakan dalam simulasi dan analisis sistem kendali.



## 3.1 Instalasi dan Pengelolaan Lingkungan Python

### 3.1.1 Instalasi Anaconda

1. Unduh Anaconda dari https://www.anaconda.com/products/distribution sesuai sistem operasi Anda.
2. Jalankan installer dan ikuti instruksi pemasangan hingga selesai.
3. Buka terminal (Anaconda Prompt di Windows) dan ketik:

```
conda --version
```

Perintah di atas digunakan untuk memastikan conda sudah terinstal dengan benar. Jika muncul versi conda, maka instalasi berhasil.

### 3.1.2 Pengelolaan Virtual Environment

- Membuat environment baru bernama `kendali` dengan Python 3.10:

```
conda create -n kendali python=3.10
```

Perintah ini membuat environment terpisah dengan nama `kendali` dan Python versi 3.10.

- Mengaktifkan environment:

```
conda activate kendali
```

Setelah diaktifkan, semua instalasi library akan masuk ke environment ini.

- Menonaktifkan environment:

```
conda deactivate
```

Environment yang aktif akan dinonaktifkan dan kembali ke base environment.

- Melihat daftar environment:

```
conda env list
```

Menampilkan seluruh environment yang tersedia di sistem.

- Menghapus environment:

```
conda remove -n kendali --all
```

Menghapus environment `kendali` beserta seluruh isinya.

## 3.2 Instalasi Library

Instalasi library utama untuk simulasi sistem kendali berbasis Python dapat dilakukan menggunakan conda dan pip. Berikut langkah-langkah instalasinya:

```
conda install numpy matplotlib jupyter
```

Perintah di atas digunakan untuk menginstal NumPy, Matplotlib, dan Jupyter Notebook secara langsung dalam environment aktif menggunakan conda. Ketiga library ini sangat penting untuk komputasi numerik, visualisasi, dan pemrograman interaktif.

```
pip install control
```

Perintah ini digunakan untuk menginstal library python-control menggunakan pip, karena library ini belum tersedia di conda default channel.

## 3.3 Penggunaan Dasar Jupyter Notebook

Jupyter Notebook adalah aplikasi web yang memungkinkan Anda menulis kode, dokumentasi, dan visualisasi secara interaktif.

```
jupyter notebook
```

Perintah ini akan membuka browser pada alamat `http://localhost:8888/`. Anda dapat membuat file notebook baru (.ipynb) untuk eksperimen, simulasi, dan dokumentasi kode.

## 3.4 Penggunaan Dasar NumPy

```
import numpy as np

a = np.array([1, 2,[3])
b = np.array([4,[5][6])
c = a + b
d = a * b
e = np.dot(a, b)

m1 = np.array([[1,[2], [3][4]])
m2 = np.array([[5,[6], [7][8]])
m3 = m1 + m2
m4 = np.matmul(m1, m2)
m5 = m1.T
det = np.linalg.det(m1)
inv = np.linalg.inv(m1)

zeros = np.zeros((3, 3))
ones = np.ones((2, 2))
identity = np.eye(4)
linspace = np.linspace(0, 10, 100)
```

Penjelasan:

- `a` dan `b` adalah array satu dimensi.
- `c` adalah hasil penjumlahan elemen array.
- `d` adalah hasil perkalian elemen array.
- `e` adalah hasil perkalian dot product antara `a` dan `b`.
- `m1` dan `m2` adalah matriks 2x2.
- `m3` adalah hasil penjumlahan dua matriks.
- `m4` adalah hasil perkalian matriks.

- `m5` adalah transpose dari matriks `m1`.
- `det` adalah determinan dari matriks `m1`.
- `inv` adalah invers dari matriks `m1`.
- `zeros` adalah matriks nol berukuran 3x3.
- `ones` adalah matriks satu berukuran 2x2.
- `identity` adalah matriks identitas 4x4.
- `linspace` adalah array berisi 100 titik dari 0 sampai 10.

## 3.5 Penggunaan Dasar Matplotlib

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.linspace(0, 10, 100)
5  y = np.sin(x)
6
7  plt.plot(x, y)
8  plt.xlabel('x')
9  plt.ylabel('sin(x)')
10 plt.title('Grafik Sinus')
11 plt.grid(True)
12 plt.show()
13
14 y2 = np.cos(x)
15 plt.plot(x, y, label='sin(x)')
16 plt.plot(x, y2, label='cos(x)')
17 plt.legend()
18 plt.show()
19
20 plt.subplot(2, 1, 1)
21 plt.plot(x, y)
22 plt.title('sin(x)')
23
24 plt.subplot(2, 1, 2)
25 plt.plot(x, y2)
26 plt.title('cos(x)')
27 plt.tight_layout()
28 plt.show()
29
30 data = np.random.randn(1000)
31 plt.hist(data, bins=30)
32 plt.title('Histogram Data Acak')
33 plt.show()
34
35 plt.scatter(x, y)
36 plt.title('Scatter Plot')
```

```
37  plt.show()
```

Penjelasan:

- Membuat grafik fungsi sinus dengan label sumbu dan judul.
- Membuat dua grafik (sinus dan cosinus) dalam satu plot dengan legenda.
- Membuat dua subplot (sin dan cos) dalam satu gambar.
- Membuat histogram dari data acak.
- Membuat scatter plot dari data `x` dan `y`.

## 3.6 Penggunaan Dasar python-control

```
1  import control as ctrl
2
3  num = [1]
4  den = [1][1]
5  sistem = ctrl.TransferFunction(num, den)
```

Penjelasan:

- Membuat sistem transfer function $G(s) = \frac{1}{s+1}$ dengan pembilang `num` dan penyebut `den`.
- Objek `sistem` menyimpan representasi sistem linier dalam bentuk transfer function.

Untuk analisis dan simulasi lebih lanjut (respons step, impuls, dsb.) akan dibahas pada bab berikutnya.

# 4 Pemodelan Sistem

Pemodelan sistem merupakan tahap fundamental dalam perancangan dan analisis sistem kendali. Dengan pemodelan yang tepat, karakteristik sistem dapat dipahami dan strategi kendali yang sesuai dapat dirancang untuk mencapai performa yang diinginkan. Bab ini membahas konsep dasar pemodelan sistem, pendekatan matematis yang umum digunakan, serta contoh implementasi pemodelan untuk berbagai aplikasi teknik.



## 4.1 Konsep Dasar Pemodelan Sistem

Pemodelan sistem adalah proses merepresentasikan perilaku dinamis suatu sistem fisik ke dalam bentuk matematis, sehingga dapat dianalisis dan disimulasikan. Model sistem biasanya dinyatakan dalam bentuk persamaan diferensial, fungsi alih (transfer function), atau model ruang keadaan (state-space).

Model yang baik harus mampu menggambarkan karakteristik utama sistem, namun tetap cukup sederhana untuk dianalisis dan digunakan dalam perancangan kendali. Pemodelan yang akurat sangat penting agar sistem kendali yang dirancang dapat bekerja sesuai harapan, baik pada simulasi maupun implementasi nyata.

## 4.2 Langkah-langkah Pemodelan Sistem

1. Identifikasi Sistem Fisik
   Menentukan komponen utama, variabel masukan (input), dan keluaran (output) sistem yang akan dimodelkan.
2. Penentuan Asumsi dan Penyederhanaan
   Membuat asumsi untuk menyederhanakan model, misal mengabaikan gesekan kecil, mengasumsikan sistem linier, atau menganggap struktur kaku.

3. Penurunan Persamaan Dinamis
   Menggunakan hukum fisika (misal Hukum Newton, Hukum Kirchoff, atau Hukum Euler-Lagrange) untuk menurunkan persamaan diferensial yang menggambarkan hubungan antara input dan output sistem.

4. Representasi Model Matematis
   Mengubah persamaan dinamis ke dalam bentuk fungsi alih (transfer function) atau model ruang keadaan (state-space).

5. Validasi Model
   Membandingkan hasil simulasi model dengan data eksperimen atau sistem nyata untuk memastikan keakuratan model.

## 4.3 Bentuk Representasi Model Sistem

### 4.3.1 Fungsi Alih (Transfer Function)

Fungsi alih adalah representasi hubungan antara input dan output sistem linier waktu invarian (LTI) dalam domain Laplace. Bentuk umum fungsi alih:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + ... + b_1 s + b_0}{a_n s^n + ... + a_1 s + a_0}$$

di mana:

- $Y(s)$: Transformasi Laplace dari output
- $U(s)$: Transformasi Laplace dari input
- $s$: Variabel kompleks Laplace

Fungsi alih banyak digunakan untuk analisis sistem linier, seperti sistem mekanik, elektrik, dan proses industri.

### 4.3.2 Model Ruang Keadaan (State-Space)

Model ruang keadaan merepresentasikan sistem dalam bentuk himpunan persamaan diferensial orde satu:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

di mana:

- $x(t)$: Vektor status (state)
- $u(t)$: Vektor input
- $y(t)$: Vektor output
- $A, B, C, D$: Matriks sistem

Model ini sangat fleksibel dan dapat digunakan untuk sistem multi-input multi-output (MIMO), serta sistem yang tidak mudah direpresentasikan dengan fungsi alih.

## 4.4 Contoh Pemodelan Sistem

### 4.4.1 Sistem Mekanik (Massa-Pegas-Redam)

Misal sistem terdiri dari massa $m$, pegas $k$, dan peredam $b$, dengan gaya input $F(t)$ dan posisi output $x(t)$:

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = F(t)$$

Transformasi Laplace:

$$G(s) = \frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k}$$

### 4.4.2 Sistem Elektrik (Rangkaian RC)

Untuk rangkaian seri resistor $R$ dan kapasitor $C$ dengan tegangan input $V\_in(t)$ dan output $V\_out(t)$:

$$RC\frac{dV\_out}{dt} + V\_out = V\_in$$

Fungsi alih:

$$G(s) = \frac{V\_out(s)}{V\_in(s)} = \frac{1}{RCs + 1}$$

### 4.4.3 Sistem Proses Industri (Tangki Air)

Sistem tangki air dengan luas permukaan $A$, laju aliran masuk $q\_in$, dan laju aliran keluar $q\_out$:

$$A\frac{dh}{dt} = q\_in - q\_out$$

Jika $q\_out = kh$:

$$A\frac{dh}{dt} + kh = q\_in$$

Fungsi alih:

$$G(s) = \frac{H(s)}{Q\_in(s)} = \frac{1}{As + k}$$

### 4.4.4 Sistem Motor DC

Persamaan dinamis motor DC dapat dinyatakan sebagai berikut:

$$J\frac{d\omega}{dt} + b\omega = K_i i$$

$$L\frac{di}{dt} + Ri = V - K_b\omega$$

Setelah transformasi Laplace dan eliminasi variabel, diperoleh fungsi alih kecepatan sudut terhadap tegangan input:

$$G(s) = \frac{\Omega(s)}{V(s)} = \frac{K_i}{(Js + b)(Ls + R) + K_i K_b}$$

## 4.5 Implementasi Pemodelan Fungsi Alih di Python

Berikut contoh pembuatan fungsi alih sistem massa-pegas-redam menggunakan python-control:

```
import control as ctrl

m = 1.0
b = 2.0
k = 5.0

num = [1]
den = [m, b, k]
G = ctrl.TransferFunction(num, den)
print(G)
```

Penjelasan:

- `num` adalah pembilang fungsi alih.
- `den` adalah penyebut fungsi alih.
- `G` adalah objek transfer function yang merepresentasikan sistem.

## 4.6 Studi Kasus Singkat: Pemodelan Sistem Nyata

- Sistem Kendali Volume Air Pada plant industri, pemodelan volume air dalam tangki dilakukan dengan mengukur ketinggian air menggunakan sensor ultrasonik dan laju aliran dengan flow sensor. Model matematis diperoleh dari persamaan neraca massa dan diubah menjadi fungsi alih orde dua untuk analisis dan perancangan kendali.
- Pemodelan Motor Induksi dan Motor DC Motor induksi dan motor DC dimodelkan menggunakan persamaan state-space atau fungsi alih, hasil identifikasi sistem, serta validasi dengan data eksperimen. Pendekatan ini digunakan untuk tuning kendali PID atau fuzzy agar performa sistem optimal.
- Robot Self-Balancing dan Quadcopter Sistem robotik seperti robot self-balancing dan quadcopter dimodelkan dengan pendekatan dinamika nonlinier, kemudian dilinierkan dan direpresentasikan dalam bentuk state-space untuk memudahkan analisis dan desain kendali.

## 4.7 Pentingnya Validasi Model

Validasi model dilakukan dengan membandingkan respons model terhadap data eksperimen atau sistem nyata. Jika terdapat perbedaan signifikan, model perlu direvisi atau dilakukan identifikasi ulang. Validasi yang baik memastikan model dapat digunakan untuk simulasi dan perancangan kendali secara andal.

Bab ini memberikan dasar penting bagi mahasiswa dan dosen dalam memahami bagaimana sistem fisik direpresentasikan secara matematis, sehingga dapat dianalisis, disimulasikan, dan dikendalikan secara efektif menggunakan pendekatan modern berbasis Python.

# 5 Diagram Blok

Diagram blok adalah alat visual utama dalam pemodelan sistem kendali, digunakan untuk menggambarkan hubungan antar subsistem atau komponen. Hubungan antar blok dapat berupa rangkaian seri, paralel, maupun umpan balik (feedback). Pemahaman struktur ini sangat penting karena menentukan perilaku dan karakteristik sistem secara keseluruhan, baik dalam analisis matematis maupun implementasi simulasi[4].

## 5.1 Hubungan Seri

Pada hubungan seri, keluaran dari satu blok menjadi masukan bagi blok berikutnya. Secara matematis, dua sistem dengan transfer function $G_1(s)$ dan $G_2(s)$ yang dihubungkan seri akan menghasilkan transfer function total:

$$G_{seri}(s) = G_1(s) \times G_2(s)$$

Diagram Blok Seri:

```
[Input] ---> [G1(s)] ---> [G2(s)] ---> [Output]
```

Implementasi Python (python-control):

```python
import control as ctrl

G1 = ctrl.TransferFunction([1], [1,2])   # Contoh G1(s) = 1/(s+2)
G2 = ctrl.TransferFunction([2], [1,3])   # Contoh G2(s) = 2/(s+3)

G_seri = ctrl.series(G1, G2)
print(G_seri)
```

Penjelasan:

- Membentuk dua transfer function, lalu menggabungkannya secara seri menggunakan `ctrl.series`.
- Transfer function total adalah hasil perkalian G1 dan G2.

## 5.2 Hubungan Paralel

Pada hubungan paralel, dua atau lebih blok menerima masukan yang sama dan hasil keluarannya dijumlahkan. Transfer function total dari dua sistem paralel $G_1(s)$ dan $G_2(s)$:

$$G_{paralel}(s) = G_1(s) + G_2(s)$$

Diagram Blok Paralel:

```
                +--> [G1(s)] --+
[Input] ----->|                +--> [Jumlah] ---> [Output]
                +--> [G2(s)] --+
```

Implementasi Python (python-control):

```python
import control as ctrl

G1 = ctrl.TransferFunction([1], [1,2])   # Contoh G1(s) = 1/(s+2)
G2 = ctrl.TransferFunction([2], [1,3])   # Contoh G2(s) = 2/(s+3)

G_paralel = ctrl.parallel(G1, G2)
print(G_paralel)
```

Penjelasan:

- Dua transfer function dijumlahkan menggunakan `ctrl.parallel`.
- Transfer function total adalah penjumlahan G1 dan G2.

## 5.3 Hubungan Umpan Balik (Feedback)

Pada sistem umpan balik, sebagian atau seluruh keluaran sistem dikembalikan ke masukan melalui jalur umpan balik. Sistem ini sangat umum digunakan untuk meningkatkan kestabilan dan akurasi sistem kendali.

Transfer function sistem umpan balik negatif (paling umum):

$$G_{feedback}(s) = \frac{G_1(s)}{1 + G_1(s)H(s)}$$

di mana $H(s)$ adalah transfer function jalur umpan balik.

Diagram Blok Feedback:

```
           +----------------------+
           |                      |
[Input] --+--> [G1(s)] -->[Output]
           |         ^
           |         |
           +----[H(s)]---+
```

Implementasi Python (python-control):

```python
import control as ctrl

G1 = ctrl.TransferFunction([1], [1,2])   # G1(s) = 1/(s+2)
H = ctrl.TransferFunction([1], [1,1])    # H(s) = 1/(s+1)

G_feedback = ctrl.feedback(G1, H)
print(G_feedback)
```

Penjelasan:

- `ctrl.feedback` digunakan untuk membentuk sistem umpan balik antara G1 dan H.
- Hasilnya adalah transfer function tertutup sesuai rumus di atas.

## 5.4 Contoh Diagram Blok Sistem Kendali Praktis

### 5.4.1 Sistem Kendali Motor DC (Seri dan Feedback)

- G1(s): Controller (misal PID)
- G2(s): Plant (Motor DC)
- H(s): Sensor feedback

```
[Setpoint] ---> [Controller] ---> [Motor DC] ---> [Output Kecepatan]
                  ^                                  |
```

```
3       |                                       |
4       +-----------[Sensor Feedback]----------+
```

Implementasi Python:

```python
import control as ctrl

# Controller PID: Kp = 2, Ki = 1, Kd = 0.5
Kp, Ki, Kd = 2, 1, 0.5
PID = ctrl.TransferFunction([Kd, Kp, Ki], [1])

# Plant Motor DC: G2(s) = 1/(s^2 + 3s + 2)
Plant = ctrl.TransferFunction([1], [1,3,2])

# Sistem terbuka (seri)
OpenLoop = ctrl.series(PID, Plant)

# Feedback dengan sensor unity (H(s) = 1)
ClosedLoop = ctrl.feedback(OpenLoop, 1)
print(ClosedLoop)
```

Penjelasan:

- Membentuk controller PID dan plant motor DC dalam bentuk transfer function.
- Menggabungkan secara seri, lalu membentuk sistem tertutup dengan feedback unity.

## 5.5 Manfaat Analisis Diagram Blok

- Menyederhanakan Analisis: Memudahkan pemecahan sistem kompleks menjadi blok-blok sederhana.
- Mendukung Simulasi Modular: Setiap blok dapat diuji dan dimodifikasi secara terpisah.
- Memudahkan Implementasi: Hubungan antar blok dapat langsung diimplementasikan dalam kode Python, seperti dengan `ctrl.series`, `ctrl.parallel`, dan `ctrl.feedback`.

Diagram blok dan implementasinya dalam python-control menjadi fondasi penting dalam analisis, simulasi, dan desain sistem kendali modern. Dengan memahami hubungan seri, paralel, dan feedback, mahasiswa dapat membangun dan menguji sistem kendali secara modular dan terstruktur.

# 6 Representasi State-Space

Representasi state-space adalah metode pemodelan sistem dinamis yang sangat umum digunakan dalam teknik kontrol modern. Pendekatan ini sangat bermanfaat untuk sistem multi-input multi-output (MIMO), sistem dengan orde tinggi, serta sistem yang tidak mudah direpresentasikan dengan fungsi alih. Model state-space juga sangat fleksibel untuk implementasi algoritma kendali modern seperti state feedback, LQR, observer, dan lain-lain.



## 6.1 Konsep Dasar State-Space

Model state-space merepresentasikan sistem dinamis menggunakan himpunan persamaan diferensial orde satu yang menghubungkan status internal (state), input, dan output sistem:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

di mana:

- $x(t)$: vektor status (state), menggambarkan kondisi internal sistem
- $u(t)$: vektor input (masukan)
- $y(t)$: vektor output (keluaran)
- $A$: matriks sistem (dinamika state)
- $B$: matriks input
- $C$: matriks output
- $D$: matriks feedthrough (langsung dari input ke output)

Pendekatan ini sangat berguna untuk sistem yang kompleks, seperti motor induksi, robot self-balancing, sistem tenaga listrik, dan aplikasi industri lainnya.

Keunggulan Representasi State-Space:

- Dapat digunakan untuk sistem MIMO (multi input, multi output)
- Mudah diimplementasikan pada sistem digital dan embedded system (persamaan difference/discrete)
- Mendukung perancangan kendali modern seperti state feedback, LQR, observer, dan MPC
- Cocok untuk sistem nonlinier setelah dilinierkan di sekitar titik operasi

## 6.2 Contoh Penurunan Model State-Space

### 6.2.1 Sistem Massa-Pegas-Redam

Persamaan dinamis:

$$m\ddot{x} + b\dot{x} + kx = u$$

Didefinisikan:

- $x_1 = x$ (posisi)
- $x_2 = \dot{x}$ (kecepatan)

Maka:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(u - bx_2 - kx_1)$$

Bentuk matriks:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

Output:

$$y = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

### 6.2.2 Sistem Elektrik (Rangkaian RC)

Persamaan dinamis:

$$RC\frac{dV * out}{dt} + V * out = V_{in}$$

Definisikan state $x = V_{out}$:

$$\dot{x} = -\frac{1}{RC}x + \frac{1}{RC}u$$

$$y = x$$

## 6.3 Implementasi State-Space di Python dengan python-control

Berikut contoh implementasi model state-space pada sistem massa-pegas-redam menggunakan python-control:

```
import control as ctrl
import numpy as np

m = 1.0
b = 2.0
k = 5.0

A = np.array([[0, [-k/m, -b/m]])
B = np.array([, [1/m]])
C = np.array([])
D = np.array([])

sistem_ss = ctrl.ss(A, B, C, D)
print(sistem_ss)
```

Penjelasan:

- Matriks A, B, C, D dibentuk sesuai hasil penurunan model matematis.
- Fungsi `ctrl.ss()` digunakan untuk membuat objek state-space.
- Objek `sistem_ss` dapat digunakan untuk analisis lebih lanjut, simulasi respons, dan perancangan kendali berbasis state-space.

## 6.4 Konversi antara State-Space dan Transfer Function

Pada banyak kasus, kita perlu mengonversi model dari transfer function ke state-space atau sebaliknya. Python-control menyediakan fungsi untuk melakukan konversi ini secara mudah.

### 6.4.1 Transfer Function ke State-Space

```
import control as ctrl

# Misal G(s) = 1/(s^2 + 2s + 5)
num =
den = [1,= ctrl.TransferFunction(num, den)

ss_model = ctrl.tf2ss(G)
print(ss_model)
```

Penjelasan:

- `ctrl.tf2ss()` mengubah transfer function menjadi model state-space.

### 6.4.2 State-Space ke Transfer Function

```
import control as ctrl
import numpy as np

A = np.array([,
              [-5, -2]])
B = np.array([,
              ])
C = np.array([])
D = np.array([])

ss_model = ctrl.ss(A, B, C, D)
tf_model = ctrl.ss2tf(ss_model)
print(tf_model)
```

Penjelasan:

- `ctrl.ss2tf()` mengubah model state-space menjadi transfer function.

## 6.5 Kontrolabilitas dan Observabilitas

Kontrolabilitas (controllability) dan observabilitas (observability) adalah dua konsep fundamental dalam analisis sistem dinamis berbasis state-space. Kedua konsep ini menentukan apakah suatu sistem dapat dikendalikan dan diamati secara penuh melalui input dan output yang tersedia. Memastikan sistem yang dirancang bersifat kontrolabel dan observabel sangat penting sebelum melakukan desain kendali lanjutan seperti state feedback, observer, atau LQR.

### 6.5.1 Kontrolabilitas (Controllability)

Definisi:
Sistem dikatakan kontrolabel jika dimungkinkan untuk membawa keadaan (state) sistem dari kondisi awal apa pun ke kondisi akhir apa pun dalam waktu hingga tertentu, hanya dengan memberikan input yang sesuai.

Secara matematis:
Untuk sistem linier kontinu:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

Sistem kontrolabel jika matriks kontrolabilitas $\mathcal{C}$ memiliki rank penuh (full rank):

$$\mathcal{C} = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B]$$

dengan $n$ adalah orde sistem (jumlah state).

Implementasi di Python:

```
1  import numpy as np
2  import control as ctrl
3
4  A = np.array([,[-5, -2]])
```

```
B = np.array([,])
# Matriks kontrolabilitas
from numpy.linalg import matrix_rank

n = A.shape
controllability_matrix = B
for i in range(1, n):
    controllability_matrix = np.hstack((controllability_matrix
        , np.linalg.matrix_power(A, i) @ B))

rank_c = matrix_rank(controllability_matrix)
print("Rank kontrolabilitas:", rank_c)
```

Penjelasan:

- Matriks kontrolabilitas dibentuk dengan menggabungkan B, AB, $A^2B$, dst.
- Jika rank matriks sama dengan jumlah state (orde sistem), maka sistem kontrolabel.

### 6.5.2 Observabilitas (Observability)

Definisi:
Sistem dikatakan observabel jika seluruh keadaan (state) sistem dapat ditentukan secara unik dari output dan input selama interval waktu tertentu.

Secara matematis:
Untuk sistem linier kontinu:

$$\dot{x}(t) = Ax(t) + Bu(t) y(t) = Cx(t) + Du(t)$$

Sistem observabel jika matriks observabilitas $\mathcal{O}$ memiliki rank penuh:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

Implementasi di Python:

```python
import numpy as np

A = np.array([,[-5, -2]])
C = np.array([])

n = A.shape
observability_matrix = C
for i in range(1, n):
    observability_matrix = np.vstack((observability_matrix, C
        @ np.linalg.matrix_power(A, i)))

rank_o = np.linalg.matrix_rank(observability_matrix)
print("Rank observabilitas:", rank_o)
```

Penjelasan:

- Matriks observabilitas dibentuk dengan menumpuk C, CA, $CA^2$, dst.
- Jika rank matriks sama dengan jumlah state (orde sistem), maka sistem observabel.

### 6.5.3 Penjelasan Tambahan

- Sistem yang tidak kontrolabel: Tidak semua state dapat dikendalikan dengan input yang tersedia. Desain kendali state feedback tidak akan efektif.
- Sistem yang tidak observabel: Tidak semua state dapat diestimasi dari output yang tersedia. Desain observer atau estimator tidak akan akurat.
- Sistem ideal: Sistem yang kontrolabel dan observabel, sehingga seluruh state dapat dikendalikan dan diamati.

Sebelum melakukan desain kendali berbasis state-space (misal: pole placement, LQR, observer), selalu periksa kontrolabilitas dan observabilitas. Jika sistem tidak memenuhi salah satu syarat, perlu dilakukan modifikasi model, penambahan sensor/aktuator, atau pemilihan strategi kendali lain.

Analisis kontrolabilitas dan observabilitas adalah langkah penting dalam memastikan sistem state-space dapat dikendalikan dan dimonitor secara penuh, sehingga desain kendali modern dapat diterapkan secara efektif dan optimal.

## 6.6 Aplikasi dan Studi Kasus State-Space

- Motor Induksi dan Motor DC:
  Model state-space digunakan untuk analisis dan desain kendali modern seperti LQR, state feedback, dan observer pada motor listrik, baik untuk efisiensi energi maupun peningkatan performa dinamis.
- Robot Self-Balancing:
  Sistem robot beroda dua dengan dinamika kompleks direpresentasikan dalam bentuk state-space untuk memudahkan perancangan kendali fuzzy dan LQR, serta evaluasi performa sistem.
- Sistem Suspensi Aktif:
  Model state-space digunakan pada sistem suspensi aktif kendaraan untuk membandingkan performa kendali PID dan LQR, serta mengoptimalkan kenyamanan dan stabilitas kendaraan.

Untuk implementasi pada sistem digital atau embedded, model state-space dapat didiskretkan menggunakan metode seperti backward difference, sehingga dapat diimplementasikan pada mikrokontroler atau komputer industri.

Representasi state-space dan kemampuan konversi ke/dari transfer function adalah fondasi utama dalam perancangan dan analisis sistem kendali modern. Dengan pendekatan ini, sistem yang kompleks dapat dimodelkan, dianalisis, dan dikendalikan secara efisien baik dalam simulasi maupun implementasi nyata.

# 7 Analisis Domain Waktu

Analisis domain waktu merupakan salah satu pendekatan utama dalam mengevaluasi kinerja dan karakteristik sistem kendali. Dengan menganalisis respons sistem terhadap masukan tertentu dalam domain waktu, kita dapat memahami bagaimana sistem bereaksi terhadap gangguan, perubahan setpoint, serta menilai kestabilan dan performa sistem secara menyeluruh. Analisis ini sangat penting, baik untuk sistem linier maupun nonlinier, dan menjadi dasar dalam desain serta tuning parameter kendali.



## 7.1 Konsep Dasar Analisis Domain Waktu

Analisis domain waktu mempelajari bagaimana keluaran (output) sistem berubah terhadap waktu sebagai respons terhadap masukan tertentu, seperti step, impuls, ramp, atau sinyal acak. Melalui respons waktu, karakteristik utama sistem seperti waktu naik (rise time), waktu tunda (delay time), waktu pemulihan (settling time), overshoot, dan error steady-state dapat dievaluasi.

Parameter-parameter ini sangat penting dalam aplikasi nyata, misalnya:

- Sistem kendali suhu harus mampu mencapai suhu target dengan cepat dan tanpa overshoot berlebih.
- Sistem kendali posisi motor harus stabil dan minim error steady-state.

## 7.2 Jenis Respons Domain Waktu

### 7.2.1 Respons Step

Respons sistem terhadap masukan loncatan (step) digunakan untuk menilai seberapa cepat dan stabil sistem mencapai kondisi baru setelah perubahan mendadak pada input.

### 7.2.2 Respons Impuls

Respons terhadap masukan impuls digunakan untuk melihat reaksi awal sistem terhadap gangguan singkat.

### 7.2.3 Respons Ramp

Respons terhadap masukan ramp (kenaikan linier) digunakan untuk menilai kemampuan sistem mengikuti perubahan input yang bertahap.

### 7.2.4 Respons Terhadap Sinyal Arbitrer

Sistem juga dapat dianalisis terhadap masukan acak atau periodik untuk menguji performa dalam kondisi nyata.

## 7.3 Parameter Kinerja Domain Waktu

Beberapa parameter penting yang dievaluasi dari respons waktu sistem antara lain:

- Waktu Naik (Rise Time):Waktu yang dibutuhkan output untuk naik dari 10% ke 90% nilai akhir.
- Waktu Puncak (Peak Time):Waktu saat output mencapai nilai maksimum.
- Overshoot:Persentase maksimum output yang melebihi nilai akhir.
- Waktu Pemulihan (Settling Time):Waktu yang dibutuhkan output untuk tetap berada dalam rentang tertentu (misal ±2%) dari nilai akhir.
- Error Steady-State:Selisih antara nilai output akhir dengan nilai input referensi setelah waktu yang cukup lama.

Parameter-parameter ini digunakan untuk membandingkan dan mengoptimalkan desain sistem kendali.

Kestabilan sistem dapat dievaluasi dengan mengamati respons waktu terhadap gangguan atau perubahan input. Sistem yang stabil akan selalu kembali ke titik keseimbangan setelah terjadi gangguan, sedangkan sistem tidak stabil akan menghasilkan respons yang terus membesar atau berosilasi tanpa redaman.

## 7.4 Simulasi dan Implementasi Analisis Domain Waktu di Python

Python dan library python-control menyediakan berbagai fungsi untuk menganalisis respons waktu sistem. Berikut contoh implementasi dasar:

### 7.4.1 Respons Step

```
import control as ctrl
import matplotlib.pyplot as plt

num =
den =
G = ctrl.TransferFunction(num, den)

t, y = ctrl.step_response(G)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Respons Step')
plt.title('Respons Step Sistem')
plt.grid(True)
plt.show()
```

Penjelasan:

- `ctrl.step_response()` menghasilkan waktu dan output respons step dari sistem.
- Grafik menunjukkan bagaimana sistem mencapai nilai akhirnya.

### 7.4.2 Respons Impuls

```
import control as ctrl
import matplotlib.pyplot as plt

num =
den =
G = ctrl.TransferFunction(num, den)

t, y = ctrl.impulse_response(G)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Respons Impuls')
plt.title('Respons Impuls Sistem')
plt.grid(True)
plt.show()
```

Penjelasan:

- `ctrl.impulse_response()` digunakan untuk menganalisis respons sistem terhadap masukan impuls.

### 7.4.3 Respons terhadap Sinyal Arbitrer (Forced Response)

```
import control as ctrl
import numpy as np
import matplotlib.pyplot as plt

num =
den =
G = ctrl.TransferFunction(num, den)

t = np.linspace(0, 10, 100)
u = np.sin(t)
t, y, x = ctrl.forced_response(G, T=t, U=u)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Respons terhadap Masukan Sinus')
plt.title('Forced Response Sistem')
plt.grid(True)
plt.show()
```

Penjelasan:

- `ctrl.forced_response()` digunakan untuk mensimulasikan respons sistem terhadap masukan yang ditentukan sendiri, misalnya sinyal sinus.



### 7.4.4 Analisis Parameter Kinerja Otomatis

Beberapa parameter kinerja seperti rise time, settling time, dan overshoot dapat dihitung secara otomatis menggunakan library tambahan seperti `control.timeresp` atau dengan mengolah data hasil simulasi.

## 7.5 Analisis Sistem Orde 1 dan Orde 2

Analisis sistem orde 1 dan orde 2 sangat penting karena kedua jenis sistem ini sering dijumpai dalam berbagai aplikasi teknik, baik sebagai model utama maupun sebagai blok

pembentuk sistem yang lebih kompleks. Karakteristik respons waktu sistem-sistem ini dapat dihitung secara analitik dan menjadi dasar untuk memahami perilaku sistem kendali sebelum masuk ke sistem dengan orde lebih tinggi.

### 7.5.1 Sistem Orde 1

Bentuk umum fungsi alih sistem orde 1:

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

di mana:

- $K$ = gain (penguatan sistem)
- $\tau$ = konstanta waktu (time constant)

Ciri utama respons step sistem orde 1:

- Respons naik secara eksponensial menuju nilai akhir tanpa overshoot.
- Waktu menuju steady-state ditentukan oleh $\tau$: semakin besar $\tau$, semakin lambat respons sistem.
- Waktu pemulihan (settling time) biasanya sekitar $4\tau$.

Persamaan respons step:

$$y(t) = K\left(1 - e^{-t/\tau}\right)$$

Contoh Implementasi di Python:

```
import control as ctrl
import matplotlib.pyplot as plt

K = 2
tau = 3
num = [K]
den = [tau, 1]
G1 = ctrl.TransferFunction(num, den)

t, y = ctrl.step_response(G1)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Respons Step')
```

```
plt.title('Respons Step Sistem Orde 1')
plt.grid(True)
plt.show()
```

Penjelasan:

- Sistem orde 1 dengan gain 2 dan konstanta waktu 3 detik.
- Grafik menunjukkan respons naik eksponensial menuju nilai akhir 2.

### 7.5.2 Sistem Orde 2

Bentuk umum fungsi alih sistem orde 2:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

di mana:

- $\omega_n$ = frekuensi natural (natural frequency)
- $\zeta$ = rasio redaman (damping ratio)

Karakteristik respons step sistem orde 2:

- Underdamped $(01)$:Respons lambat, tanpa osilasi, waktu menuju steady-state lebih lama.
- Un-damped ($\zeta = 0$):Respons berosilasi terus-menerus tanpa redaman.

Parameter penting:

- Overshoot ($M_p$):Persentase puncak maksimum di atas nilai akhir.
- Waktu naik (rise time, $t_r$)
- Waktu puncak (peak time, $t_p$)
- Waktu pemulihan (settling time, $t_s$)

Contoh Implementasi di Python:

```
import control as ctrl
import matplotlib.pyplot as plt

omega_n = 2       # Frekuensi natural
zeta = 0.5        # Rasio redaman (underdamped)
num = [omega_n2]
```

```
den = [1, 2*zeta*omega_n, omega_n2]
G2 = ctrl.TransferFunction(num, den)

t, y = ctrl.step_response(G2)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Respons Step')
plt.title('Respons Step Sistem Orde 2 (Underdamped)')
plt.grid(True)
plt.show()
```

Penjelasan:

- Sistem orde 2 dengan $\omega_n = 2$ dan $\zeta = 0.5$ (underdamped).
- Respons menunjukkan adanya overshoot dan osilasi sebelum mencapai steady-state.

### 7.5.3 Perbandingan Karakteristik Sistem Orde 1 dan Orde 2

| Karakteristik | Orde 1 | Orde 2 (Underdamped) |
|---|---|---|
| Bentuk respons | Eksponensial | Osilasi teredam |
| Overshoot | Tidak ada | Ada (tergantung $\zeta$) |
| Settling time | ~$4\tau$ | Tergantung $\zeta, \omega_n$ |
| Rise time | Tergantung $\tau$ | Tergantung $\zeta, \omega_n$ |
| Parameter utama | $\tau$ | $\omega_n, \zeta$ |



### 7.5.4 Aplikasi Sistem Orde 1 dan Orde 2

- Sistem Orde 1: Banyak ditemukan pada proses termal, sistem pengisian kapasitor, sensor suhu, dan sistem hidrolik sederhana. Contoh pada penelitian ketersediaan air bersih di IKN, model matematika menggunakan persamaan diferensial orde satu untuk menggambarkan perubahan kuantitas air akibat input dan output sistem.
- Sistem Orde 2: Umum pada sistem mekanik massa-pegas-redam, sistem suspensi kendaraan, dan respons transien motor listrik. Dalam sistem tenaga listrik, ana-

lisis kestabilan sudut dan waktu pemutus kritis juga sering menggunakan model orde dua untuk mendeskripsikan dinamika generator dan beban.

Analisis sistem orde 1 dan orde 2 memberikan pemahaman mendalam tentang perilaku dasar sistem dinamis, menjadi fondasi penting sebelum menganalisis atau merancang sistem kendali yang lebih kompleks.

## 7.6 Studi Kasus Analisis Domain Waktu

- Sistem Kendali Suhu Industri: Analisis domain waktu digunakan untuk memastikan suhu proses ozonasi limbah tetap stabil dalam rentang waktu tertentu dan tidak mengalami overshoot berlebih, sehingga efisiensi proses meningkat.
- Sistem Kendali Pitch Pesawat: Optimasi parameter PID dilakukan dengan meminimalkan overshoot, rise time, dan settling time berdasarkan respons step, sehingga sistem mampu menjaga sudut pitch dengan cepat dan presisi.
- Sistem Tenaga Listrik: Analisis domain waktu digunakan untuk mengevaluasi kestabilan tegangan dan frekuensi setelah terjadi gangguan, memastikan sistem kembali ke kondisi normal dalam waktu singkat.

Analisis domain waktu sangat penting dalam:

- Menentukan performa awal sistem sebelum tuning kendali.
- Membandingkan efektivitas berbagai strategi kendali (on-off, PID, fuzzy, dsb.).
- Menjamin sistem memenuhi spesifikasi teknis seperti waktu pemulihan, overshoot, dan error steady-state.
- Menjadi dasar dalam optimasi parameter kendali berbasis simulasi maupun eksperimen.

Analisis domain waktu adalah fondasi utama dalam evaluasi dan desain sistem kendali modern. Dengan memahami respons waktu, parameter kinerja, dan kestabilan sistem, perancang dapat mengembangkan sistem kendali yang responsif, stabil, dan andal untuk berbagai aplikasi teknik.

# 8 Analisis Domain Frekuensi

Analisis domain frekuensi adalah metode penting dalam evaluasi dan desain sistem kendali, terutama untuk sistem linier waktu invarian (LTI). Dengan pendekatan ini, perilaku sistem terhadap masukan sinusoidal pada berbagai frekuensi dapat dipelajari, sehingga karakteristik seperti kestabilan, respons frekuensi, margin gain dan phase, serta sensitivitas terhadap gangguan dapat dianalisis secara komprehensif. Analisis domain frekuensi juga sangat relevan dalam aplikasi industri, tenaga listrik, dan sistem elektronik, di mana kestabilan dan performa sistem sering diuji menggunakan sinyal periodik dan gangguan harmonik.



## 8.1 Konsep Dasar Analisis Domain Frekuensi

Pada analisis domain frekuensi, sistem diuji dengan masukan sinusoidal berbagai frekuensi. Output sistem diamati untuk setiap frekuensi, sehingga diperoleh informasi tentang gain (penguatan) dan phase shift (pergeseran fasa) sistem terhadap masukan tersebut.

Karakteristik utama yang diamati:

- Magnitude (Gain): Perbandingan amplitudo output terhadap input pada setiap frekuensi.
- Phase: Pergeseran fasa antara input dan output pada setiap frekuensi.

Analisis ini sangat penting untuk menilai kestabilan sistem, kemampuan sistem menolak gangguan, serta menentukan parameter tuning kendali yang optimal.

## 8.2 Representasi Analisis Domain Frekuensi

### 8.2.1 Bode Plot

Bode plot adalah grafik yang menampilkan magnitude (dalam dB) dan phase (dalam derajat) terhadap skala logaritmik frekuensi. Bode plot sangat populer karena mudah digunakan untuk menganalisis sistem orde tinggi dan memberikan informasi margin gain serta margin phase.

### 8.2.2 Nyquist Plot

Nyquist plot menggambarkan respons frekuensi sistem dalam bidang kompleks (real-imaginer). Plot ini sangat berguna untuk analisis kestabilan sistem tertutup dan penentuan margin kestabilan.

### 8.2.3 Nichols Chart

Nichols chart menggabungkan magnitude dan phase dalam satu grafik, sering digunakan untuk desain kendali berbasis frekuensi.

## 8.3 Parameter Kinerja Domain Frekuensi

Beberapa parameter penting yang dievaluasi dalam domain frekuensi antara lain:

- Bandwidth: Rentang frekuensi di mana sistem dapat merespons masukan dengan baik.
- Gain Margin: Margin penguatan maksimum sebelum sistem menjadi tidak stabil.
- Phase Margin: Margin pergeseran fasa maksimum sebelum sistem menjadi tidak stabil.
- Resonant Peak: Titik di mana magnitude respons maksimum, berhubungan dengan kemungkinan overshoot di domain waktu.
- Frekuensi Crossover: Frekuensi di mana magnitude sistem sama dengan 0 dB.

Parameter-parameter ini sangat penting dalam desain sistem kendali yang stabil dan robust[4].

Analisis domain frekuensi memungkinkan evaluasi kestabilan sistem tanpa harus mensimulasikan respons waktu secara langsung. Sistem dikatakan stabil jika tidak ada osilasi berkelanjutan atau amplifikasi sinyal pada frekuensi tertentu. Margin gain dan phase menjadi indikator utama kestabilan sistem tertutup, terutama pada sistem industri dan tenaga listrik yang sering mengalami gangguan frekuensi[1][4].

## 8.4 Simulasi Analisis Domain Frekuensi di Python

Python-control menyediakan berbagai fungsi untuk analisis domain frekuensi, seperti Bode plot, Nyquist plot, dan margin analisis.

### 8.4.1 Bode Plot

```
import control as ctrl
import matplotlib.pyplot as plt

num =
den =
G = ctrl.TransferFunction(num, den)

mag, phase, omega = ctrl.bode_plot(G, dB=True, Hz=False,
    omega_limits=(0.1, 100), Plot=True)
plt.show()
```

Penjelasan:

- `ctrl.bode_plot()` menghasilkan grafik magnitude dan phase terhadap frekuensi.
- `omega_limits` menentukan rentang frekuensi yang dianalisis.

### 8.4.2 Nyquist Plot

```
import control as ctrl
import matplotlib.pyplot as plt

num =
den =
G = ctrl.TransferFunction(num, den)

ctrl.nyquist_plot(G)
```

```
plt.show()
```

Penjelasan:

- `ctrl.nyquist_plot()` menggambarkan plot Nyquist dari sistem.

### 8.4.3 Margin Analisis

```
import control as ctrl

num =
den =
G = ctrl.TransferFunction(num, den)

gm, pm, sm, wg, wp, ws = ctrl.margin(G)
print("Gain Margin:", gm)
print("Phase Margin:", pm)
print("Frekuensi Gain Margin:", wg)
print("Frekuensi Phase Margin:", wp)
```

Penjelasan:

- `ctrl.margin()` menghitung gain margin, phase margin, dan frekuensi crossover.

## 8.5 Studi Kasus Analisis Domain Frekuensi

- Sistem Tenaga Listrik:
  Analisis domain frekuensi digunakan untuk menilai kestabilan sistem tenaga listrik terhadap gangguan beban dan variasi parameter, serta untuk merancang sistem kendali frekuensi agar sistem tetap stabil setelah gangguan.
- Motor Induksi dan VFD:
  Monitoring frekuensi pada motor induksi tiga fasa dengan variable frequency drive (VFD) digunakan untuk mengoptimalkan konsumsi daya dan memperpanjang umur motor, serta menghindari resonansi berbahaya.
- Sistem Elektronik dan PLTS:
  Analisis harmonisa dan supraharmonics pada sistem photovoltaic menggunakan Fast Fourier Transform (FFT) untuk mendeteksi frekuensi gangguan dominan yang dapat mempengaruhi performa perangkat elektronik dan jaringan distribusi listrik.

Karakteristik domain frekuensi seperti bandwidth, resonant peak, dan margin phase sangat berhubungan dengan parameter domain waktu seperti waktu naik, overshoot, dan settling time. Sistem dengan bandwidth tinggi umumnya memiliki respons waktu yang cepat, sedangkan margin phase yang kecil dapat menyebabkan overshoot dan osilasi di domain waktu.

Aplikasi Analisis Domain Frekuensi dalam Desain Kendali:

- Menentukan parameter tuning PID dan compensator berbasis frekuensi.
- Menganalisis dan mengoptimalkan kestabilan serta robustness sistem industri.
- Mendesain filter dan sistem penolak gangguan harmonik pada aplikasi tenaga listrik dan elektronik.

Analisis domain frekuensi adalah landasan penting dalam desain, evaluasi, dan optimasi sistem kendali modern. Dengan pemahaman yang baik terhadap respons frekuensi, margin kestabilan, dan sensitivitas sistem, perancang dapat memastikan sistem bekerja optimal dan aman pada berbagai kondisi operasi nyata.

# 9 Kontroler PID

Kontroler PID (Proportional-Integral-Derivative) adalah salah satu algoritma kendali paling luas digunakan dalam industri, robotika, otomasi, dan berbagai aplikasi teknik lainnya. Fleksibilitas, kemudahan implementasi, serta kemampuannya dalam mengatasi berbagai karakteristik sistem membuat PID menjadi standar utama dalam pengendalian proses baik linier maupun nonlinier.

## 9.1 Konsep Dasar Kontroler PID

Kontroler PID bekerja dengan menghitung sinyal kendali berdasarkan tiga komponen utama:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

di mana:

- $u(t)$: sinyal kendali ke plant
- $e(t)$: error (selisih antara setpoint dan output aktual)
- $K_p$: gain proporsional
- $K_i$: gain integral
- $K_d$: gain derivatif

Setiap komponen memiliki peran spesifik:

- Proportional (P): Menghasilkan aksi kendali proporsional terhadap besar error.
- Integral (I): Menghilangkan error steady-state dengan mengakumulasi error.
- Derivative (D): Mengantisipasi perubahan error, meredam osilasi dan meningkatkan stabilitas.

## 9.2 Struktur PID dalam Domain Laplace

Fungsi alih kontroler PID dalam domain Laplace:

$$C(s) = K_p + \frac{K_i}{s} + K_d s$$

atau dalam bentuk transfer function:

$$C(s) = \frac{K_d s^2 + K_p s + K_i}{s}$$

Fungsi dan Karakteristik Masing-masing Komponen:

- P (Proportional):
  - Mempercepat respons sistem.
  - Terlalu besar menyebabkan overshoot dan osilasi.
- I (Integral):
  - Menghilangkan error steady-state.
  - Terlalu besar menyebabkan sistem lambat dan bisa menimbulkan osilasi.
- D (Derivative):
  - Meredam osilasi dan memperbaiki stabilitas.
  - Terlalu besar menyebabkan sistem sensitif terhadap noise.

Jenis-jenis Kontroler PID:

- P (Proportional) Controller
- PI (Proportional-Integral) Controller
- PD (Proportional-Derivative) Controller
- PID (Proportional-Integral-Derivative) Controller

Pemilihan jenis tergantung pada kebutuhan sistem dan spesifikasi performa yang diinginkan.

## 9.3 Implementasi PID di Python (python-control)

### 9.3.1 Membuat Kontroler PID

```
import control as ctrl

Kp = 2.0
Ki = 1.0
Kd = 0.5

# PID: (Kd*s^2 + Kp*s + Ki)/s

num = [Kd, Kp, Ki]
den = [1]
PID = ctrl.TransferFunction(num, den)
print(PID)
```

Penjelasan:

- Membuat transfer function PID sesuai rumus Laplace.
- `ctrl.TransferFunction` digunakan untuk mendefinisikan blok PID.

### 9.3.2 Simulasi Sistem dengan PID

```
import control as ctrl
import matplotlib.pyplot as plt

# Plant: G(s) = 1/(s^2 + 3s + 2)

num_plant = [1]
den_plant = [1][3][2]
Plant = ctrl.TransferFunction(num_plant, den_plant)

# PID Controller

Kp = 2.0
Ki = 1.0
Kd = 0.5
num_pid = [Kd, Kp, Ki]
den_pid = [1]
PID = ctrl.TransferFunction(num_pid, den_pid)

# Open loop: PID \* Plant

OpenLoop = ctrl.series(PID, Plant)

```

```
# Closed loop dengan feedback unity

ClosedLoop = ctrl.feedback(OpenLoop, 1)

# Simulasi respons step

t, y = ctrl.step_response(ClosedLoop)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Output')
plt.title('Respons Step Sistem dengan PID')
plt.grid(True)
plt.show()
```

Penjelasan:

- Sistem plant dan PID controller digabung secara seri lalu ditutup dengan feedback unity.
- Simulasi respons step untuk melihat performa kendali PID.

## 9.4 Tuning Parameter PID

Tuning parameter PID adalah proses menentukan nilai optimal dari tiga parameter kendali utama yaitu $K_p$ (proportional), $K_i$ (integral), dan $K_d$ (derivative) agar sistem kendali dapat bekerja dengan performa terbaik sesuai kebutuhan aplikasi. Proses tuning ini sangat krusial karena parameter yang tidak tepat dapat menyebabkan sistem menjadi lambat, overshoot berlebihan, osilasi, atau bahkan tidak stabil.

Tujuan utama tuning PID adalah:

- Mempercepat waktu respons sistem.
- Meminimalkan overshoot dan osilasi.
- Menghilangkan error steady-state.
- Menjaga kestabilan dan robustness sistem terhadap gangguan dan perubahan parameter.

Tantangan tuning PID meliputi:

- Karakteristik sistem yang tidak linier dan berubah-ubah.
- Keterbatasan informasi model sistem yang lengkap.
- Interaksi antar parameter PID yang kompleks.
- Trade-off antara kecepatan respons dan kestabilan.

### 9.4.1 Metode Tuning Klasik

#### 9.4.1.1 Metode Ziegler-Nichols

Metode ini menggunakan eksperimen untuk mendapatkan ultimate gain $K_u$ dan ultimate period $T_u$ dari sistem pada kondisi osilasi berkelanjutan, kemudian parameter PID dihitung berdasarkan rumus empiris.

| Tipe Kontroler | $K_p$ | $K_i$ | $K_d$ |
|---|---|---|---|
| P | $0.5K_u$ | - | - |
| PI | $0.45K_u$ | $1.2K_p/T_u$ | - |
| PID | $0.6K_u$ | $2K_p/T_u$ | $K_pT_u/8$ |

Metode ini mudah dan cepat, namun kurang akurat untuk sistem dengan delay besar atau nonlinier.

#### 9.4.1.2 Metode Cohen-Coon

Metode ini cocok untuk sistem dengan delay waktu signifikan dan memberikan formula tuning berdasarkan parameter model proses. Lebih akurat dibanding Ziegler-Nichols untuk proses industri tertentu.

### 9.4.2 Metode Tuning Modern dan Otomatis

#### 9.4.2.1 Metaheuristik dan Algoritma Optimasi

Metode ini menggunakan algoritma seperti Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Firefly Algorithm, dan lain-lain untuk mencari parameter PID optimal berdasarkan fungsi objektif (misal minimisasi error kuadrat, overshoot, settling time).

Keunggulan:

- Dapat menangani sistem nonlinier dan multivariable.

- Tidak memerlukan model matematis yang lengkap.
- Mampu menemukan solusi optimal global.

#### 9.4.2.2 Pembelajaran Mesin dan Reinforcement Learning

Pendekatan ini menggunakan teknik pembelajaran untuk menyesuaikan parameter PID secara adaptif berdasarkan performa sistem selama operasi, termasuk metode diferensiasi otomatis dan pembelajaran berbasis model.

### 9.4.3 Prosedur Umum Tuning PID

1. Identifikasi Model Sistem (jika memungkinkan).
2. Pilih metode tuning sesuai karakteristik sistem dan kebutuhan.
3. Lakukan tuning awal (manual atau otomatis).
4. Simulasikan dan evaluasi respons sistem (waktu naik, overshoot, settling time, error steady-state).
5. Lakukan fine-tuning berdasarkan hasil simulasi atau eksperimen nyata.
6. Validasi performa pada kondisi operasi nyata dan lakukan penyesuaian bila diperlukan.

### 9.4.4 Contoh Tuning PID dengan Metode Ziegler-Nichols

Misal dari eksperimen diperoleh:

- Ultimate gain $K_u = 6.0$
- Ultimate period $T_u = 2.5$ detik

Parameter PID dihitung untuk tipe PID:

$$K_p = 0.6 \times 6.0 = 3.6$$

$$K_i = \frac{2 \times 3.6}{2.5} = 2.88$$

$$K_d = \frac{3.6 \times 2.5}{8} = 1.125$$

### 9.4.5 Implementasi Tuning PID di Python

```python
import control as ctrl
import matplotlib.pyplot as plt

# Parameter PID hasil tuning Ziegler-Nichols
Kp = 3.6
Ki = 2.88
Kd = 1.125

num_pid = [Kd, Kp, Ki]
den_pid = [1]
PID = ctrl.TransferFunction(num_pid, den_pid)

# Contoh plant: G(s) = 1/(s^2 + 3s + 2)
num_plant = [1]
den_plant = [1][3][2]
Plant = ctrl.TransferFunction(num_plant, den_plant)

# Sistem terbuka
OpenLoop = ctrl.series(PID, Plant)

# Sistem tertutup dengan feedback unity
ClosedLoop = ctrl.feedback(OpenLoop, 1)

# Simulasi respons step
t, y = ctrl.step_response(ClosedLoop)
plt.plot(t, y)
plt.xlabel('Waktu (detik)')
plt.ylabel('Output')
plt.title('Respons Step Sistem dengan PID Tuning Ziegler-
    Nichols')
plt.grid(True)
plt.show()
```

### 9.4.6 Kelebihan dan Kekurangan Metode Tuning

| Metode | Kelebihan | Kekurangan |
|---|---|---|
| Manual Trial and Error | Mudah dan cepat | Tidak sistematis, memakan waktu dan kurang optimal |
| Ziegler-Nichols | Cepat dan populer | Kurang akurat untuk sistem dengan delay besar atau nonlinier |

| Metode | Kelebihan | Kekurangan |
|---|---|---|
| Cohen-Coon | Lebih akurat untuk proses dengan delay | Rumus kompleks, memerlukan model proses |
| Metaheuristik | Optimal, fleksibel, cocok sistem nonlinier | Memerlukan komputasi tinggi dan waktu tuning |
| Pembelajaran Mesin | Adaptif dan otomatis | Kompleks, memerlukan data dan pemahaman mendalam |

Tren dan Riset Terkini dalam Tuning PID:

- Penggunaan algoritma metaheuristik terus berkembang untuk mengatasi keterbatasan metode klasik.
- Integrasi pembelajaran mesin dan reinforcement learning untuk tuning adaptif dan real-time.
- Pengembangan tuning PID untuk sistem nonlinier dan multivariabel kompleks.
- Penelitian pada tuning PID fractional order (FOPID) untuk performa lebih baik pada sistem dinamis nyata.

Tuning parameter PID adalah aspek vital dalam desain sistem kendali yang menentukan performa dan kestabilan sistem. Metode tuning klasik seperti Ziegler-Nichols masih banyak digunakan karena kemudahan dan kecepatan, namun metode modern berbasis optimasi dan pembelajaran mesin menawarkan solusi lebih optimal dan adaptif untuk sistem yang lebih kompleks dan dinamis.

Pemilihan metode tuning harus disesuaikan dengan karakteristik sistem, tujuan kendali, dan sumber daya yang tersedia agar hasil kendali optimal dan handal.



## 9.5 Aplikasi PID dalam Industri dan Riset

- Industri Kimia: Kontrol level dan aliran cairan pada tangki terkopel yang bersifat nonlinier.
- Robotika Miniatur: Implementasi PID multi-channel pada chip tunggal untuk mengendalikan banyak aktuator sekaligus.
- Sistem Otomasi: Pengaturan suhu, tekanan, kecepatan motor, dan posisi robot.

- Motor Listrik dan Servo: PID digunakan untuk mengoptimalkan kecepatan dan posisi motor DC, PMSM, dan motor induksi.

Variasi dan Pengembangan PID:

- PI/PD Controller: Digunakan jika hanya diperlukan penghilangan error steady-state (PI) atau peredaman osilasi (PD).
- Fractional Order PID (FOPID): Menambahkan derajat kebebasan pada orde integral dan derivatif untuk meningkatkan performa pada sistem kompleks dan nonlinier.
- Decentralized PID: Untuk sistem multi-input multi-output (MIMO) seperti tangki terkopel dua input dua output (TITO), digunakan PID terdesentralisasi untuk setiap channel.
- Self-Tuning PID: PID dengan algoritma penyesuaian parameter otomatis berbasis optimasi atau kecerdasan buatan.

Kelebihan dan Keterbatasan PID:

Kelebihan:

- Mudah diimplementasikan dan dipahami.
- Cocok untuk banyak aplikasi industri.
- Efektif untuk sistem linier dan beberapa sistem nonlinier sederhana.

Keterbatasan:

- Kurang optimal untuk sistem dengan delay besar atau dinamika nonlinier kompleks.
- Tuning parameter bisa sulit pada sistem multivariabel atau sistem dengan banyak gangguan.
- Sensitif terhadap noise pada sinyal derivatif.



## 9.6 Studi Kasus dan Perbandingan

- Sistem Tangki Terkopel: Studi komparatif menunjukkan PID lebih unggul dibanding PI dalam mengendalikan level dan aliran pada sistem tangki terkopel nonlinier. PID mampu mengurangi error steady-state dan mempercepat waktu pemulihan, namun tuning parameter sangat penting untuk menghindari osilasi berlebih.

- Kontrol Pitch pada Sistem Rotasi: Berbagai metode tuning (Ziegler–Nichols, Aström–Hägglund, Kaiser–Chaira) diuji pada sistem pitch, menunjukkan bahwa pemilihan metode tuning mempengaruhi overshoot dan waktu pemulihan secara signifikan.
- Robotika dan Miniaturisasi: Implementasi PID multi-channel pada chip tunggal memungkinkan pengendalian banyak aktuator secara efisien untuk aplikasi robotika miniatur dan sistem embedded.

Kontroler PID tetap menjadi solusi utama dalam dunia kendali karena kesederhanaan, fleksibilitas, dan efektivitasnya dalam berbagai aplikasi. Dengan pemahaman mendalam tentang prinsip kerja, tuning, serta variasi dan keterbatasannya, engineer dapat mengoptimalkan performa sistem kendali baik di industri maupun riset modern.

# 10 Simulasi Wall-Following di Webots dengan PID Control

Bab ini membahas implementasi algoritma wall-following pada robot menggunakan Webots, dengan fokus pada kontrol PID dan perbandingannya dengan bang-bang controller. Simulasi ini menggunakan Robotino 3 (Festo Didactic) yang dilengkapi sensor jarak dan motor diferensial.



## 10.1 Instalasi dan Setup Awal

### 10.1.1 Instalasi Webots

1. Unduh Webots versi terbaru dari https://cyberbotics.com
2. Ikuti panduan instalasi sesuai sistem operasi
3. Pastikan Python 3.x terinstal dan path Python ditambahkan ke environment variable

### 10.1.2 Setup Proyek Webots

1. Buat proyek baru di Webots
2. Tambahkan robot:
   - Pilih `Robotino 3` dari library Webots
   - Tambahkan 8 sensor jarak (infrared/distance sensor) di posisi melingkar
3. Konfigurasi sensor:

```python
sensors = []
for i in range(8):
    sensor = robot.getDevice(f"distance*sensor*{i}")
    sensor.enable(TIME_STEP)
    sensors.append(sensor)
```

## 10.2 Pemrograman Dasar

### 10.2.1 Pembacaan Sensor

```
def read_sensors():
    values = []
    for sensor in sensors:
        values.append(sensor.getValue())
        return values
```

Sensor 0-7 memberikan nilai 0-1000 (semakin kecil = semakin dekat ke dinding).

### 10.2.2 Kontrol Motor Diferensial

```
left_motor = robot.getDevice("left_wheel_motor")
right_motor = robot.getDevice("right_wheel_motor")
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))

def set_speed(left, right):
    left_motor.setVelocity(left)
    right_motor.setVelocity(right)
```

## 10.3 Implementasi Bang-Bang Controller

### 10.3.1 Prinsip Kerja:

- Jika jarak ke dinding threshold: belok kiri

### 10.3.2 Kode Implementasi:

```

THRESHOLD = 500
BASE_SPEED = 2.0

while robot.step(TIME_STEP) != -1:
sensor_values = read_sensors()
front_left = sensor_values[1]

    if front_left 30%
```

### 10.3.3 Studi Kasus:

Pada sudut 90°, PID mampu menjaga jarak konstan dengan error ±2cm, sementara Bang-Bang mengalami overshoot 15cm dan membutuhkan 3 detik untuk stabil.

## 10.4 Implementasi PID Controller

Pada bagian ini dibahas secara komprehensif implementasi algoritma PID untuk wall-following pada robot di Webots, mulai dari konsep, pemrograman, tuning, hingga analisis performa. PID controller digunakan untuk menjaga jarak robot terhadap dinding pada nilai setpoint yang diinginkan dengan menghasilkan pergerakan yang halus, responsif, dan minim error steady-state.



### 10.4.1 Konsep dan Prinsip Kerja PID pada Wall-Following

PID controller bekerja dengan menghitung error antara jarak sensor ke dinding dan setpoint (jarak ideal). Sinyal kendali yang dihasilkan merupakan kombinasi dari tiga komponen:

- Proportional (P): Koreksi berdasarkan error saat ini.
- Integral (I): Koreksi akumulasi error masa lalu untuk menghilangkan error steady-state.
- Derivative (D): Koreksi berdasarkan perubahan error untuk meredam osilasi dan mempercepat respons.

Output PID digunakan untuk mengatur kecepatan relatif roda kiri dan kanan sehingga robot tetap sejajar dan pada jarak konstan dari dinding.

### 10.4.2 Struktur Dasar Algoritma PID Wall-Following

1. Baca nilai sensor jarak (misal sensor kiri untuk left wall-following)
2. Hitung error:
   $\text{error} = \text{setpoint} - \text{nilai sensor}$
3. Update integral error:
   $\text{integral} += \text{error}$

4. Hitung derivative error:
   derivative $=$ error $-$ error sebelumnya
5. Hitung output PID:
   output $= K_p \times$ error $+ K_i \times$ integral $+ K_d \times$ derivative
6. Atur kecepatan motor:
   - Roda kiri: base $-$ output
   - Roda kanan: base $+$ output

### 10.4.3 Contoh Kode PID Controller di Webots

```
# Parameter PID
Kp = 0.8
Ki = 0.001
Kd = 0.3
setpoint = 600  # Jarak ideal dari dinding (misal 600 satuan sensor)
integral = 0
last_error = 0

def pid_control(sensor_value):
    global integral, last_error
    error = setpoint - sensor_value
    integral += error
    derivative = error - last_error
    last_error = error
    output = Kp * error + Ki * integral + Kd * derivative
    return output

BASE_SPEED = 3.0

while robot.step(TIME_STEP) != -1:
    sensor_values = read_sensors()
    left_sensor = sensor_values[1]  # Misal sensor kiri untuk left wall-following

    pid_output = pid_control(left_sensor)
    left_speed = BASE_SPEED - pid_output
    right_speed = BASE_SPEED + pid_output

    set_speed(left_speed, right_speed)
```

Penjelasan kode:

- Setpoint diatur sesuai jarak ideal yang ingin dipertahankan dari dinding.

- Fungsi `pid_control` menghitung output PID berdasarkan error, integral, dan derivative.
- Output PID digunakan untuk menyeimbangkan kecepatan roda kiri dan kanan.
- Robot akan bergerak lurus jika error nol, berbelok jika error positif/negatif.

### 10.4.4 Tuning Parameter PID

- Tuning dapat dilakukan secara manual (trial and error) atau otomatis (misal dengan PSO, GA, Bat Algorithm).
- Nilai parameter optimal sangat bergantung pada karakteristik robot, sensor, dan lingkungan.
- Parameter tuning yang baik menghasilkan error rata-rata sangat kecil (misal MAE < 2 cm).

Analisis Performa PID Controller:

- Respons lebih halus dan stabil dibandingkan bang-bang controller. Robot mampu mengikuti dinding dengan deviasi kecil dan minim osilasi.
- Error steady-state sangat kecil jika tuning tepat, bahkan pada lintasan dengan banyak belokan atau permukaan tidak rata.
- Waktu tempuh lebih cepat karena robot tidak perlu sering memperbaiki posisi secara ekstrem.
- Robust terhadap gangguan dan perubahan lingkungan jika parameter PID dioptimalkan dengan baik.

Contoh hasil pengujian:

- Nilai MAE (Mean Absolute Error) < 2 cm pada lintasan lurus dan berbelok[1].
- Waktu puncak PID lebih cepat (misal 2.24 detik) dibandingkan kontrol fuzzy (3.71 detik), menunjukkan respons yang lebih cepat.
- Error rata-rata PID lebih kecil dibandingkan bang-bang controller dan lebih mudah dikontrol pada lintasan kompleks.

Catatan Praktis:

- Sampling time (interval pembacaan sensor dan update PID) harus konsisten, misal 100–300 ms[1].

- Filter sederhana seperti moving average dapat digunakan untuk mengurangi noise sensor sebelum masuk ke PID.
- Pada tikungan tajam, override mode atau switching ke strategi lain (misal bang-bang) bisa digunakan untuk menghindari stuck.

Implementasi PID controller untuk wall-following di Webots menghasilkan pergerakan robot yang lebih presisi, halus, dan efisien dibandingkan metode bang-bang. Dengan tuning parameter yang tepat, PID mampu menjaga jarak robot terhadap dinding secara konsisten dan responsif, bahkan pada lintasan dengan banyak belokan atau permukaan tidak rata.

## 10.5 Troubleshooting Umum

1. Osilasi Berlebihan:
    - Kurangi $K_p$
    - Tingkatkan $K_d$
2. Respon Lambat:
    - Tingkatkan $K_p$
    - Kurangi $K_i$
3. Sensor Noise:
    - Tambahkan filter moving average:

```
SMOOTHING_FACTOR = 0.3
smoothed_value = SMOOTHING_FACTOR * new_value + (1 -
    SMOOTHING_FACTOR) * previous_value
```

Optimasi Lanjutan:

1. Adaptive PID: Sesuaikan parameter secara real-time berdasarkan kondisi lingkungan
2. Fuzzy-PID Hybrid: Gabungkan logika fuzzy untuk handling kondisi ekstrem
3. PID dengan Feedforward: Tambahkan kompensasi untuk perubahan lintasan tiba-tiba

Dengan implementasi yang tepat, PID controller menawarkan solusi optimal untuk wall-following dibandingkan bang-bang controller, terutama dalam hal stabilitas dan efisiensi energi. Bab selanjutnya akan membahas integrasi algoritma ini dengan SLAM dan navigasi otonom tingkat lanjut.