Introduction to Programming for Control & Application

IK6016 Control and Optimization Technology

Table of contents

02

Control Application

Complex numbers, polynomials, transfer functions, simulating system responses, and control system design

Complex Numbers

- Complex numbers are used to represent signals and system responses.
- Poles and zeros in control systems are often complex.

$$z = a + bi$$

Where:

- a = real part
- b = imaginary part
- i = imaginary unit $\sqrt{-1}$

Basic operations on complex numbers:

- **Addition**

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

- **Multiplication**

$$(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$$

- **Conjugate**

$$(a + bi) = (a - bi)$$

Complex Numbers

Example in Python:

```
# Two ways of defining complex number in
Python
z1 = 3 + 4j
z2 = complex(1, -2)

# Operations
z_sum = z1 + z2
z_product = z1 * z2
z_conjugate = z1.conjugate()

print("Sum:", z_sum)
print("Product:", z_product)
print("Conjugate:", z_conjugate)
```

Example in MATLAB:

```
% Defining complex numbers in MATLAB
z1 = 3 + 4i;
z2 = 1 - 2i;

% Operations
z_sum = z1 + z2;
z_product = z1 * z2;
z_conjugate = conj(z1);

disp(['Sum: ', num2str(z_sum)])
disp(['Product: ', num2str(z_product)])
disp(['Conjugate: ', num2str(z_conjugate)])
```

Polynomials

Used to represent the numerator and denominator of transfer functions.

$$P(s) = a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0$$

Polynomials operations:

- Addition
- Multiplication
- Roots of polynomials

Example in Python:

```
import numpy as np

# Define a polynomial P(s) = s^2 + 5s + 6
coefficients = [1, 5, 6]

# Roots of the polynomial
roots = np.roots(coefficients)
print("Roots of the polynomial:", roots)
```

Example in MATLAB:

```
% Define a polynomial P(s) = s^2 + 5s + 6
coefficients = [1 5 6];

% Find the roots of the polynomial
roots = roots(coefficients);
disp('Roots of the polynomial:')
disp(roots)
```

Transfer Functions

What is a transfer function?

- Represents the relationship between the input and output of a system in the Laplace domain

$$H(s) = \frac{Y(s)}{U(s)} = \frac{b_n s^n + b_{n-1} s^{n-1} + \cdots + b_0}{a_m s^m + a_{m-1} s^{m-1} + \cdots + a_0}$$

Example:

$$H(s) = \frac{1}{s^2 + 5s + 6}$$

Why use transfer functions?

- Simplifies analysis
- Provides insight into stability, resonance, and system dynamics

Properties of transfer functions:

- **Poles** → Values of s that make the denominator zero
- **Zeros** →Values of s that make the numerator zero
- **System stability** → Stable if all poles lie in the left half of the complex plane

Transfer Functions

Example in Python:

```
import scipy.signal as signal

# Define the transfer function: H(s) = 1 /
(s^2 + 5s + 6)
num = [1]  # Numerator coefficients
den = [1, 5, 6]  # Denominator coefficients

# Create the transfer function
system = signal.TransferFunction(num, den)

# Display the transfer function
print("Transfer function:", system)

# Analyze poles and zeros
poles, zeros, gain = signal.tf2zpk(num, den)
print("Poles:", poles)
print("Zeros:", zeros)
```

Example in MATLAB:

```
% Define the transfer function: H(s) = 1 /
(s^2 + 5s + 6)
num = [1];  % Numerator coefficients
den = [1 5 6];  % Denominator coefficients

% Create the transfer function
H = tf(num, den);

% Display the transfer function
disp('Transfer function:')
H

% Analyze poles and zeros
pzmap(H)  % Plot poles and zeros
```

Example: Multiplication

Example in Python:

```
import scipy.signal as signal

num1 = [1]
den1 = [1, 5, 6]
num2 = [2]
den2 = [1, 3]

# Multiply two transfer functions
system1 = signal.TransferFunction(num1, den1)
system2 = signal.TransferFunction(num2, den2)
system_product =
signal.TransferFunction(np.polymul(num1,
num2), np.polymul(den1, den2))

print("Product of transfer functions:",
system_product)
```

Example in MATLAB:

```
num1 = [1];
den1 = [1 5 6];
num2 = [2];
den2 = [1 3];

% Multiply two transfer functions
H1 = tf(num1, den1);
H2 = tf(num2, den2);
H_product = H1 * H2;

disp('Product of transfer functions:')
H_product
```

System Responses

What is system response?

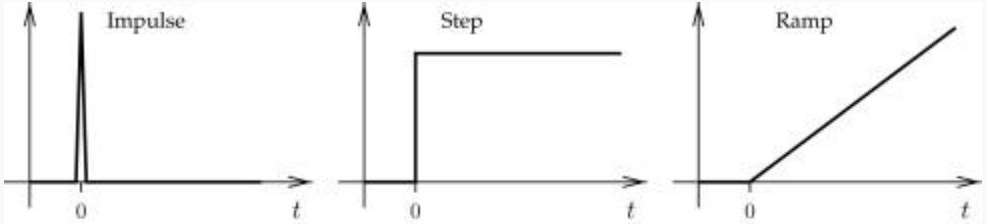
- How a system reacts over time to a given input
- Used to determine system behaviour (stability, speed, and accuracy)

Types of responses:

- Impulse response
- Step response
- Others

Why simulate system responses?

- To predict system behavior before physical implementation
- To analyze:
  - Stability
  - Settling time
  - Overshoot
  - Steady-state error



Example: Step Response

Example in Python:

```python
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define the transfer function
num = [1]  # Numerator coefficients
den = [1, 5, 6]  # Denominator coefficients
system = signal.TransferFunction(num, den)
# Simulate the step response
time, response = signal.step(system)

# Plot the step response
plt.plot(time, response)
plt.title('Step Response')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```

Example in MATLAB:

```matlab
% Define the transfer function
num = [1];
den = [1 5 6];

% Create transfer function
H = tf(num, den);

% Simulate and plot the step response
step(H)
title('Step Response')
grid on
```

Example: Impulse Response

Example in Python:

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define the transfer function
num = [1]  # Numerator coefficients
den = [1, 5, 6]  # Denominator coefficients
system = signal.TransferFunction(num, den)
# Simulate the impulse response
time, response = signal.impulse(system)

# Plot the impulse response
plt.plot(time, response)
plt.title('Impulse Response')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```

Example in MATLAB:

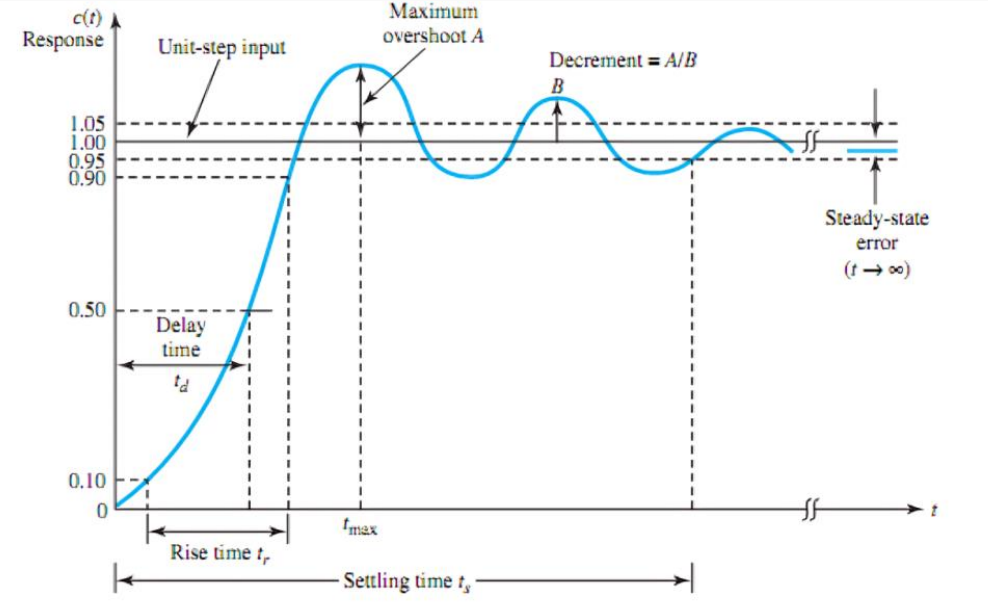
```
% Define the transfer function
num = [1];
den = [1 5 6];

% Create transfer function
H = tf(num, den);

% Simulate and plot the impulse response
impulse(H)
title('Impulse Response')
grid on
```

Control System Design

What is control system design?

- Selecting and tuning controller to meet the desired performance criteria for a system

Goal: Modify the system's behavior

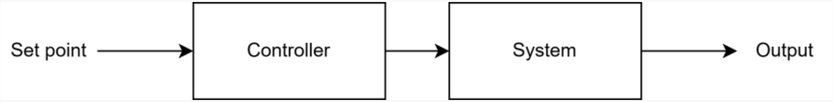


Control System Design

Open-loop control:

- No feedback from the system
- Can be used for simple tasks where disturbances are minimal

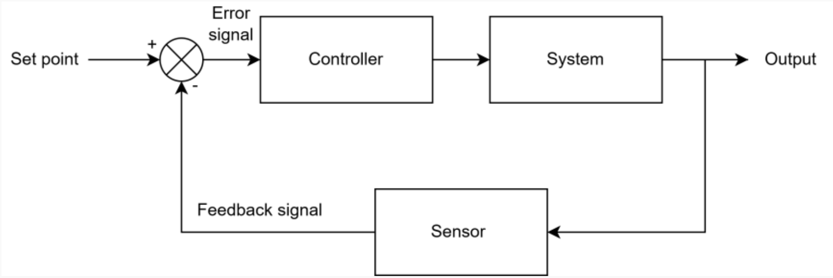
Example: A heater that maintains a fixed temperature without adjusting based on actual room temperature.



Closed-loop control:

- Feedback is used to adjust the system in real-time
- More robust and compensates for disturbances

Example: A thermostat that measures the actual room temperature and adjusts the heater accordingly



Control System Design

PID Controller:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

where:

- K_p = proportional gain
- K_i = integral gain
- K_p = derivative gain
- $e(t)$ = error signal (difference between set point and actual output)

Effect of each term:

- **Proportional (P):** Increase response speed but may introduce steady-state error
- **Integral (I):** Eliminates steady-state error but can slow down the response
- **Derivative (D):** Improves system stability and reduces overshoot but can amplify noise

Control System Design

Example in Python:

```
import control as ctrl  # pip install control
import matplotlib.pyplot as plt

# Define the transfer function of a simple
system G(s) = 1 / (s^2 + 5s + 6)
num = [1]
den = [1, 5, 6]
system = ctrl.TransferFunction(num, den)

# Define a PID controller with Kp, Ki, and Kd
values
Kp = 10
Ki = 1
Kd = 1
pid_controller = ctrl.TransferFunction([Kd,
Kp, Ki], [1, 0])
```

```
...
# Closed-loop system
closed_loop = ctrl.feedback(pid_controller *
system)

# Simulate step response
time, response =
ctrl.step_response(closed_loop)

# Plot the response
plt.plot(time, response)
plt.title('Step Response with PID')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```

Control System Design

Example in MATLAB:

```
% Define transfer function G(s) = 1 / (s^2 +
5s + 6)
num = [1];
den = [1 5 6];
G = tf(num, den);

% Define PID controller with Kp, Ki, and Kd
values
Kp = 10;
Ki = 1;
Kd = 1;
C = pid(Kp, Ki, Kd);

% Closed-loop system
T = feedback(C*G, 1);
```

```
...

% Simulate and plot step response
step(T)
title('Step Response with PID')
grid on
```

Assignments

1. **Transfer Functions:**
   Create a Python/MATLAB program that connects two transfer functions in cascade:

$$H_1(s) = \frac{5s+1}{s^2+3s+2}, \qquad H_2 = \frac{2s+4}{s+5}$$

   Find the poles and zeros of the combined system. Plot them on the complex plane and discuss the stability of the system.

2. **System Response:**
   Simulate and compare the step responses of two systems:

$$G_1(s) = \frac{5}{s^2+2s+5}, \qquad G_2(s) = \frac{5}{s^2+5s+6}$$

   Analyze the difference in response characteristics (settling time, overshoot, etc.) and discuss the implication for control design.

Assignments

3. **System Response:**

   Create a Python/MATLAB program that simulate the response of the system $G(s) = \frac{10}{s^2+5s+10}$ to the following inputs:

   a. a unit step

   b. a sinusoidal input

   c. a ramp input

   Plot and compare the system responses for each type of input.

4. **System Response:**

   Model a simple real-world system (you can choose any system, e.g., spring-mass-damper, RLC circuit, etc.) and simulate its response to different inputs using Python/MATLAB program.

Assignments

5. PID Controller:

Implement a PID controller to maintain the temperature of a room at a desired setpoint $T_{\text{setpoint}} = 25°C$. The current room temperature $T(t)$ is affected by the heating power $P(t)$ applied at time t. The rate of temperature change is modeled as:

$$\frac{dT(t)}{dt} = -K(T(t) - T_{\text{ambient}}) + \frac{P(t)}{C}$$

where K is a heat loss constant ($K = 0.1$), C is the thermal capacity of the room ($C = 5$), and T_{ambient} is the ambient temperature ($T_{\text{ambient}} = 20°C$).

Create a Python/MATLAB program that implement the PID controller and simulate the system for a total of 100 time steps and plot the temperature over time.

Bonus challenge: Tune the PID parameters for the best system response, minimizing overshoot and settling time

Resources to Study

Course materials: irinamrdhtllh/ik6016-lecture-notes (github.com)

Python

- Python: 3.12.6 Documentation (python.org)
- Numpy: NumPy documentation — NumPy v2.1 Manual
- Matplotlib: Matplotlib documentation — Matplotlib 3.9.2 documentation
- Scipy: SciPy documentation — SciPy v1.14.1 Manual
- Control: Python Control Systems Library — Python Control Systems Library 0.10.1 documentation (python-control.readthedocs.io)

MATLAB: Programming with MATLAB - MATLAB & Simulink (mathworks.com)

There's only one way to master programming:
Read the documentation, implement, and keep experimenting.