*Created by ardyseto@uny.ac.id*
*2023*

**Lab Sheet:**

# Week 2 - Modeling of Dynamic Systems

--------------------------------------------------------------------------------------------------------

**Learning Objectives:**

1. Understand transfer function representation and its operation in Python Control
2. Understand state space representation and converting its into transfer function and vice versa
3. Learn to use Python tools for system identification and parameter estimation
4. Learn about input signal of system

--------------------------------------------------------------------------------------------------------

## Task 1:

Transfer Function Representation

A transfer function is a mathematical representation of a system's input-output relationship. It is a ratio of the system's output to its input, and it is typically expressed in terms of Laplace transforms.

$$G(s) = \frac{num(s)}{den(s)} = \frac{a_0 s^m + a_1 s^{m-1} + ... + a_m}{b_0 s^n + b_1 s^{n-1} + ... + b_n}$$

The Python Control library provides a class called `TransferFunction` that can be used to represent transfer functions. The `TransferFunction` class takes two arguments: the numerator and denominator polynomials of the transfer function. To create a transfer function, use the `tf()` function:

```python
sys = ct.tf(num, den)
```

For example, the following code creates a transfer function object that represents a simple first-order system:

```python
import control

sys = control.TransferFunction([1], [1, 1])
print(sys)
```

The `sys` object can then be used to perform various operations on the transfer function, such as calculating its frequency response or stability.

Here is an example of how to use the `TransferFunction` class to represent a transfer function:

```python
import control

sys = control.TransferFunction([1], [1, 1])
print(sys)
print(sys.num)
print(sys.den)
```

**Task 2:**
First Order System

A first order system is a system that can be represented by a transfer function of the form:

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

where `K` is the gain of the system, $\tau$ is the  time constant value for the first-order system.

For example, the following code creates a first order system with a gain of 2.5 and a time constant of 0.8 second:

```python
K = 2.5  # Gain
tau = 0.8  # Time constant

num = [K]
den = [tau, 1]
tf_first_order = control.TransferFunction(num, den)

print(tf_first_order)
```

**Task 3:**
Second Order System

A second order system is a system that can be represented by a transfer function of the form:

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

where `K` is the gain of the system, $\zeta$ is the damping coefficient, and $\omega$ is the natural frequency of the system.

For example, the following code creates a second order system with a gain of 1.5, a damping coefficient of 0.6, and a natural frequency of 2 rad/s:

```python
K = 1.5  # Gain
zeta = 0.6  # Damping ratio
omega_n = 2  # Natural frequency
```

```
num = [K * omega_n**2]
den = [1, 2 * zeta * omega_n, omega_n**2]
tf_second_order = control.TransferFunction(num, den)

print(tf_second_order)
```

**Task 4:**
Combination of Transfer Function

We will explore the operations of combining transfer functions in series, parallel, and feedback configurations using Python Control. These operations are fundamental in control systems engineering as they allow us to analyze and design complex control systems by combining simpler transfer functions.

Before performing the operations, let's create some transfer function instances representing different subsystems:

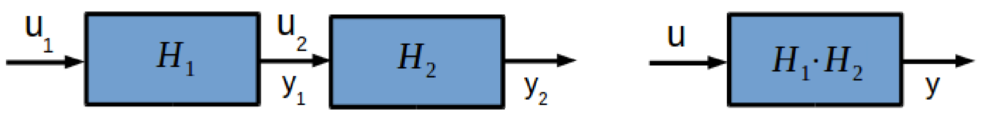
```python
# First-order transfer function
K1 = 2.5
tau1 = 0.8
num1 = [K1]
den1 = [tau1, 1]
H1 = control.TransferFunction(num1, den1)

# Second-order transfer function
K2 = 1.5
zeta2 = 0.6
omega_n2 = 2
num2 = [K2 * omega_n2**2]
den2 = [1, 2 * zeta2 * omega_n2, omega_n2**2]
H2 = control.TransferFunction(num2, den2)
```

1. Combining Transfer Functions in Series
   In a series connection, the outputs of two systems are connected to the inputs of another system. The transfer function of the overall system is the product of the transfer functions of the individual systems.

   For example, consider two systems with transfer functions `H1(s)` and `H2(s)`. The transfer function of the overall system is given by:

$$H(s) = H_1(s) * H_2(s)$$



To combine transfer functions in series, we use the `control.series()` function:

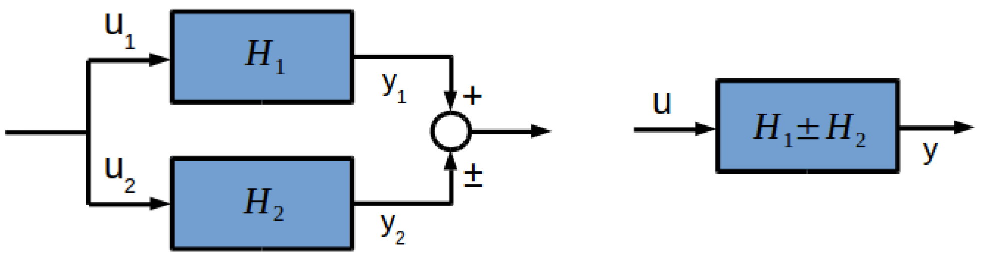
```
Python
tf_series = control.series(H1, H2)

print("Transfer Function in Series:")
print(tf_series)
```

2. Combining Transfer Functions in Parallel
   In a parallel connection, the inputs of two systems are connected to the same output. The transfer function of the overall system is the sum of the transfer functions of the individual systems.

   For example, consider two systems with transfer functions `H1(s)` and `H2(s)`. The transfer function of the overall system is given by:

$$H(s) = H_1(s) + H_2(s)$$



   To combine transfer functions in parallel, we use the `control.parallel()` function:

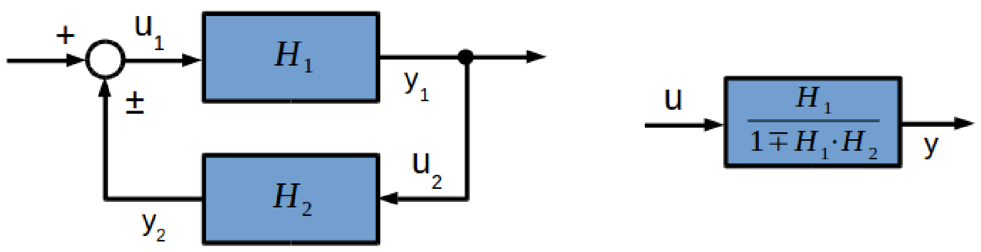
```
Python
tf_parallel = control.parallel(H1, H2)

print("\nTransfer Function in Parallel:")
print(tf_parallel)
```

3. Creating a Feedback Control System
   In a feedback connection, the output of a system is fed back to its input. The transfer function of the overall system is the product of the transfer function of the system and the feedback gain.

   For example, consider a system with transfer functions `H1(s)` and `H2(s)`. The transfer function of the overall system is given by:

$$H(s) = \frac{H_1(s)}{1 + H_1(s) \cdot H_2(s)}$$



To create a feedback control system, we use the `control.feedback()` function:

```
Python
tf_feedback = control.feedback(H1, H2)

print("\nTransfer Function in Feedback:")
print(tf_feedback)
```

**Task 5:**
State space representation

State space representation is a mathematical way of representing a system's input-output relationship. It is a more general representation than transfer function representation, and it can be used to represent both linear and nonlinear systems.

To represent a system in state-space form using the Python Control library, you can use the `control.StateSpace()` class. The state-space representation is a mathematical model that describes a system in terms of its state variables, input, and output equations. Here's how you can create a state-space representation in Python Control:

```
Python
import control

A = [[-2, 0], [1, -1]]
B = [[1], [0]]
C = [[1, 0]]
D = [[0]]

sys_ss = control.StateSpace(A, B, C, D)

print("State-Space Representation:")
print(sys_ss)
```