SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF MECHATRONICS ENGINEERING

23MCT203- THEORY OF CONTROL ENGINEERING

UNIT 5 – INTRODUCTION TO SCILAB

Ms. J. Swathi, M.E.,

Assistant professor,

Mechatronics Engineering,

SNSCT, Coimbatore.

Arithmetic Operations in Scilab

```
// Define variables
a = 10;
b = 5;

// Arithmetic Operations
addition = a + b;
subtraction = a - b;
multiplication = a * b;
division = a / b;
modulus = mod(a, b); // Modulus operation

// Display Results
disp("Addition: " + string(addition));
disp("Subtraction: " + string(subtraction));
disp("Multiplication: " + string(multiplication));
disp("Division: " + string(division));
disp("Modulus: " + string(modulus));
```

Logical Operations in Scilab

```
// Define variables
p = %t; // True
q = %f; // False

// Logical Operations
and_result = p & q;    // Logical AND
or_result = p | q;     // Logical OR
not_result = ~p;       // Logical NOT

// Display Results
disp("Logical AND: " + string(and_result));
disp("Logical OR: " + string(or_result));
disp("Logical NOT of p: " + string(not_result));
```

Relational Operations in Scilab

```
// Define variables
x = 15;
y = 10;

// Relational Operations
disp("x > y: " + string(x > y));
disp("x < y: " + string(x < y));
disp("x >= y: " + string(x >= y));
disp("x <= y: " + string(x <= y));
disp("x == y: " + string(x == y));
disp("x != y: " + string(x ~= y));
```

Example of Using Logical Operations

Here's how you can use these operations in conditional statements or complex expressions:

```
a = 5;

b = 10;

result1 = (a > 3) & (b < 15);   // result1 will be %T

result2 = (a == 5) | (b == 20);  // result2 will be %T

result3 = ~(a == b);             // result3 will be %T

result4 = (a > 0) ^ (b < 0);     // result4 will be %T
```

Control Structures in Scilab

If-Else

```
// Define a variable
num = 7;

// If-Else structure
if mod(num, 2) == 0 then
    disp("Even number");
else
    disp("Odd number");
end
```

For Loop

```
// For Loop to print numbers from 1 to 5
for i = 1:5
    disp("Value of i: " + string(i));
end
```

While Loop

```
// While Loop to print numbers from 1 to 5
j = 1;
while j <= 5 do
    disp("Value of j: " + string(j));
    j = j + 1;
end
```

Break and Continue

```
// Loop with Break and Continue
for k = 1:10
    if k == 5 then
        disp("Skipping number 5");
        continue;
    elseif k == 8 then
        disp("Breaking the loop at 8");
        break;
    end
    disp("Value of k: " + string(k));
end
```

Develop PID Controller

Steps to Implement a PID Controller in Scilab

To implement a PID controller, we'll:

1.Define the transfer function for the system.

2.Set PID parameters (proportional, integral, and derivative gains).

3.Use Scilab's control functions to simulate the closed-loop system.

Example Code for a PID Controller in Scilab

Let's assume a simple system where the plant (system being controlled) is represented by a transfer function. We'll design the PID controller to control this plant.

Step 1: Define the Plant Transfer Function

For demonstration, let's assume a first-order plant transfer function:

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

Develop PID Controller

Step 2: Define the PID Controller Parameters

The PID controller transfer function in the s domain is given by:

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

where:

- Kp: Proportional gain
- Ki : Integral gain
- Kd : Derivative gain

Step 3: Implement the PID Controller in Scilab

Here's the Scilab code for implementing a PID controller:

Develop PID Controller

Step 3: Implement the PID Controller in Scilab

Here's the Scilab code for implementing a PID controller:

```
// Load the control package
// --> atomsInstall("scicos"); --> use if Scilab Control Package not already
installed
// Load Control Package
// --> atomsLoad("scicos");

// Define system parameters
s = poly(0, 's'); // Define the Laplace variable 's'

// Define the Plant Transfer Function (Example: G(s) = 1 / (s + 1))
G = 1 / (s + 1);

// Define PID controller gains
Kp = 2;  // Proportional gain
Ki = 1;  // Integral gain
Kd = 0.5; // Derivative gain

// Define PID Controller transfer function
PID = Kp + Ki / s + Kd * s;

// Open-loop system (PID * Plant)
OL_sys = PID * G;

// Closed-loop system with unity feedback
CL_sys = OL_sys / (1 + OL_sys);

// Simulate step response of the closed-loop system
t = 0:0.1:10; // Time vector from 0 to 10 seconds
[y, x] = csim('step', CL_sys, t); // Step response of closed-loop system

// Plot the results
clf;
plot(t, y);
xlabel("Time (s)");
ylabel("Output Response");
title("PID Controller Step Response");
```

Develop PID Controller

Explanation of the Code

1.Define the Plant: The plant's transfer function, **G(s)=1/ s+1** , is defined using the Laplace variable s.

2.PID Controller: The PID controller is created by combining proportional (Kp), integral (Ki), and derivative (Kd) components.

3.Open-Loop System: The open-loop transfer function is calculated as **OL_sys = PID * G.**

4.Closed-Loop System: The closed-loop transfer function with unity feedback is **CL_sys = OL_sys / (1 + OL_sys).**

5.Step Response: The csim function simulates the step response of the closed-loop system over time t, and the results are plotted.

PID Tuning

You can adjust Kp, Ki, and Kd to tune the controller's performance:

- **Increase Kp** for quicker response but too high can cause overshoot.
- **Increase Ki** for eliminating steady-state error, but high values may lead to instability.
- **Increase Kd** for improving stability and reducing overshoot, but high values may introduce noise sensitivity.

Simulation to control a DC Motor

To simulate and control a DC motor in Scilab Cloud, the setup will be similar to using the desktop version, though Scilab Cloud primarily runs code without an interactive graphical interface like XCOS. Here’s how you can simulate a DC motor with control directly using Scilab Cloud's code interface:

Step 1: Define Motor and Control Parameters

First, define your DC motor's parameters, control gains, and simulation time settings.

Step 2: Implement the Motor Model and Control Logic

Use Scilab’s built-in functions to set up the motor equations and control algorithm, such as PID control, to manage motor speed.

Simulation to control a DC Motor

Example Code for Scilab Cloud Simulation with Control

Here's a sample code snippet that simulates a DC motor with simple proportional control to maintain a target speed:

```
// DC Motor parameters
Ra = 1; // Armature resistance in ohms
La = 0.5; // Armature inductance in henries
Kb = 0.1; // Back EMF constant
Kt = 0.1; // Torque constant
J = 0.01; // Moment of inertia
B = 0.01; // Damping coefficient

// Control parameters
Kp = 10; // Proportional gain
target_speed = 10; // Target speed in rad/s

// Time vector for simulation
t = 0:0.01:5;

// Initial conditions [initial current, initial speed]
x0 = [0; 0];

// Differential equations for motor with control
function dx/dt = motor_control_eq(t, x)
    Ia = x(1); // Armature current
    omega = x(2); // Angular velocity
```

Simulation to control a DC Motor

Example Code for Scilab Cloud Simulation with Control

Here's a sample code snippet that simulates a DC motor with simple proportional control to maintain a target speed:

```
// Control law (Proportional control for simplicity)
    error = target_speed - omega;
    Va = Kp * error;

    // Motor dynamics equations
    dx/dt(1) = (Va - Ra*Ia - Kb*omega) / La;
    dx/dt(2) = (Kt*Ia - B*omega) / J;
endfunction

// Solve differential equations
x = ode([0, 5], x0, motor_control_eq);
```

```
// Plot results
clf();
subplot(2,1,1);
plot(t, x(:,1)); // Plot current
xlabel('Time (s)');
ylabel('Current (A)');
title('Armature Current with Control');

subplot(2,1,2);
plot(t, x(:,2)); // Plot angular velocity
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
title('Motor Speed with Control');
```

Simulation to control a DC Motor

Explanation

1.Control Law: The proportional controller (with gain Kp) generates the armature voltage Va based on the error between target and actual speed.

2.ODE Solver: ode() function solves the differential equations with the control applied.

3.Plotting: You get two plots—one for armature current and one for motor speed, showing the control effect.

Step 3: Run the Simulation

1.Copy the code into the Scilab Cloud editor.

2.Run the script and observe the plots generated for current and speed response over time.

This setup can be expanded to a full PID controller if you need more precise control.

Simulation of Object Detection for Robot

To simulate object detection for a robot in Scilab Cloud, you can set up a basic model to simulate an environment where a robot detects objects. Although Scilab isn't specifically designed for computer vision, you can use matrix and logical operations to simulate a simplified environment for object detection.

Steps to Simulate Object Detection in Scilab Cloud

1.Define the Environment Grid:

1. Create a 2D grid representing the robot's environment.
2. Each cell in the grid can represent free space, an obstacle, or an object.

2.Initialize the Robot's Position and Range:

1. Define the robot's starting position and its detection range.
2. The detection range can be simulated by checking cells around the robot's position in the grid.

3.Simulate Object Detection:

1. Use a scanning algorithm to simulate the robot's "sensing" area.
2. If the robot's sensing area overlaps with a grid cell marked as an object, mark it as "detected."

4.Display the Results:

1. Plot the grid with different markers to show the robot, objects, and detected objects.

Simulation of Object Detection for Robot

Example Code in Scilab Cloud
Here is a basic example that simulates a grid-based environment with object detection for a robot in Scilab Cloud:

```
// Define the grid dimensions
grid_size = 10; // 10x10 grid
env = zeros(grid_size, grid_size); // Initialize empty grid (0 = empty)

// Place objects in the environment
env(3, 3) = 1; // Object 1 at position (3,3)
env(7, 8) = 1; // Object 2 at position (7,8)
env(6, 5) = 1; // Object 3 at position (6,5)

// Robot's starting position
robot_pos = [5, 5];

// Detection range (e.g., can detect within a 1-cell radius)
detection_range = 1;
```

Simulation of Object Detection for Robot

Example Code in Scilab Cloud
Here is a basic example that simulates a grid-based environment with object detection for a robot in Scilab Cloud:

```
// Function to check for objects within detection range
function detected_objects = detect_objects(env, robot_pos, detection_range)
    detected_objects = []; // Initialize empty list for detected objects
    [rows, cols] = size(env);

    // Loop over the detection area
    for i = max(1, robot_pos(1) - detection_range) : min(rows, robot_pos(1) + detection_range)
        for j = max(1, robot_pos(2) - detection_range) : min(cols, robot_pos(2) + detection_range)
            if env(i, j) == 1 then
                detected_objects = [detected_objects; i, j]; // Add detected object position
            end
        end
    end
endfunction

// Get detected objects
detected = detect_objects(env, robot_pos, detection_range);
```

Simulation of Object Detection for Robot

Example Code in Scilab Cloud

Here is a basic example that simulates a grid-based environment with object detection for a robot in Scilab Cloud:

```
// Plot the environment and detections
clf();
for i = 1:grid_size
    for j = 1:grid_size
        if env(i, j) == 1 then
            plot(i, j, 'ro'); // Plot objects as red circles
        end
    end
end

// Plot robot position
plot(robot_pos(1), robot_pos(2), 'bo'); // Robot as blue circle

// Plot detected objects
for k = 1:size(detected, 1)
    plot(detected(k, 1), detected(k, 2), 'gx'); // Detected objects as green
crosses
end

// Add labels and grid
xlabel("X-axis");
ylabel("Y-axis");
title("Object Detection Simulation for Robot");
xgrid();
```

Simulation of Object Detection for Robot

Explanation of the Code

1.Environment Grid: The env matrix represents the environment, with cells set to 1 for objects and 0 for empty spaces.

2.Detection Range: The robot checks cells within a defined range around its position to simulate "seeing" nearby objects.

3.Detection Logic: detect_objects() checks the specified area around the robot for any cells marked as objects and returns their coordinates.

4.Plotting: The plot shows the robot's position, object locations, and detected objects with different markers.

Running and Testing

1.Copy this code into the Scilab Cloud editor.

2.Adjust the robot_pos and detection_range values to simulate different scenarios.

3.Run the code, and you'll see a simple representation of the environment, robot, and detected objects.