

Title

Report on Control Systems

Author: Paladi Swaroop

Branch: CAI

Roll No: 21AT1A3176

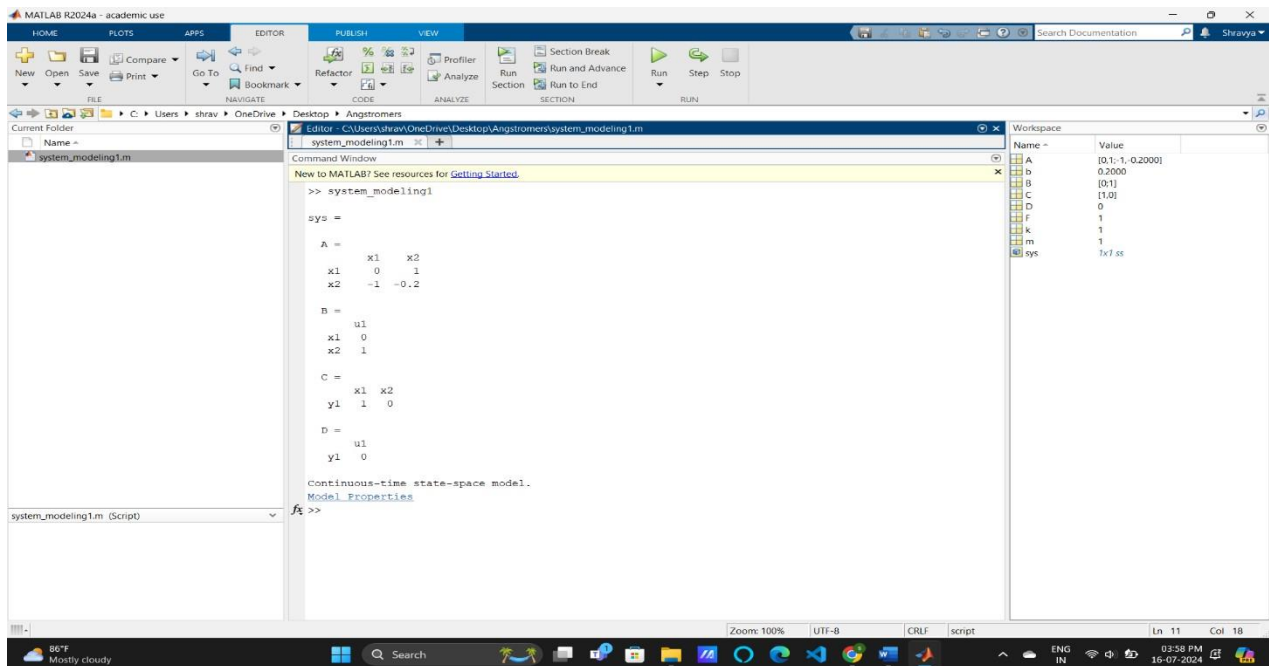
Date of Submission: 22/07/2024

Training Program Name: Angstromers

Introduction: System Modeling

The initial phase in designing a control system involves creating suitable mathematical models of the system to be managed. These models can be obtained from either theoretical principles or empirical data. This part covers the state-space and transfer function representations of dynamic systems. We will also explore fundamental methods for modeling mechanical and electrical systems and demonstrate how to create these models in MATLAB for further examination.

Key MATLAB commands utilized in this tutorial include: ss, tf.



The screenshot shows the MATLAB R2024a interface. The Editor window displays the script `system_modeling1.m` with the following code:

```
New to MATLAB? See resources for Getting Started.
>> system_modeling1

sys =

    A =          x1    x2
        x1      0      1
        x2     -1     -0.2

    B =          u1
        x1      0
        x2      1

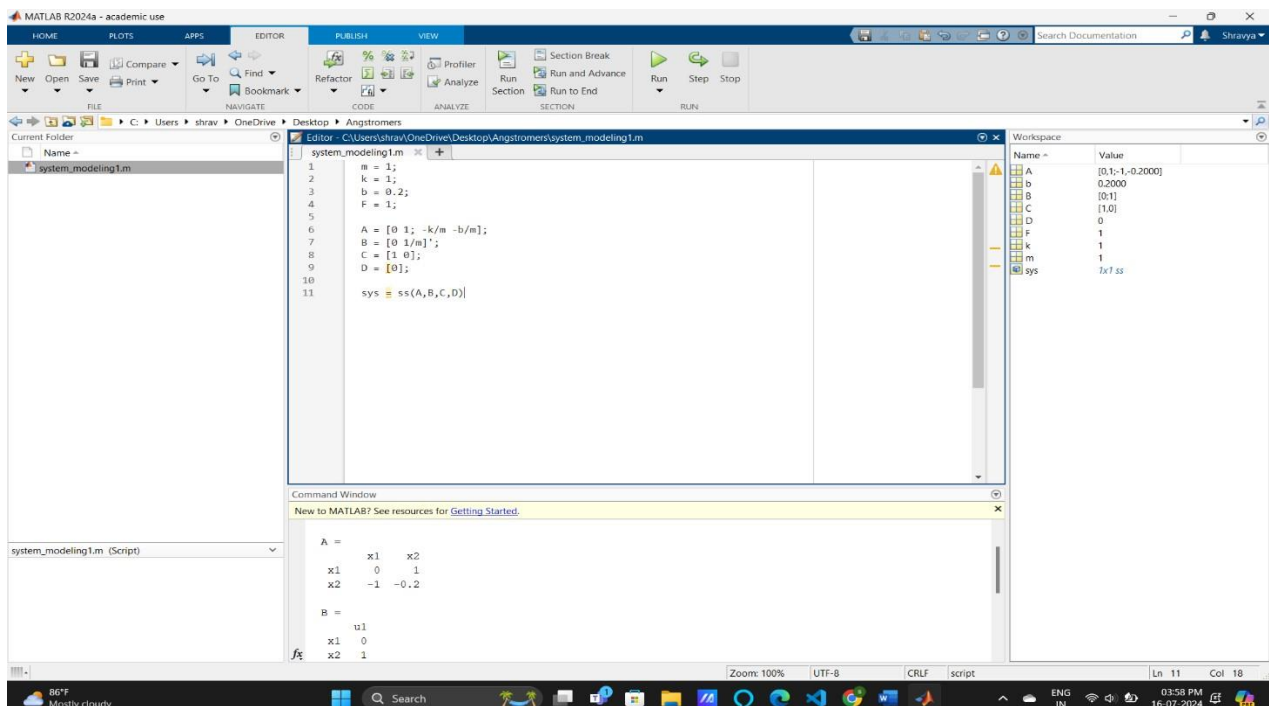
    C =          x1    x2
        y1      1      0

    D =          u1
        y1      0

Continuous-time state-space model.
Model Properties
fx >>
```

The Workspace window on the right shows the variables defined in the script:

Name	Value
A	[0.1 -1, -0.2000]
B	0.2000
C	[0 1]
D	[1 0]
F	0
k	1
m	1
sys	1x1 ss



The screenshot shows the MATLAB R2024a interface with the script `system_modeling1.m` updated to calculate the matrices `m`, `k`, `b`, and `F` before creating the state-space model:

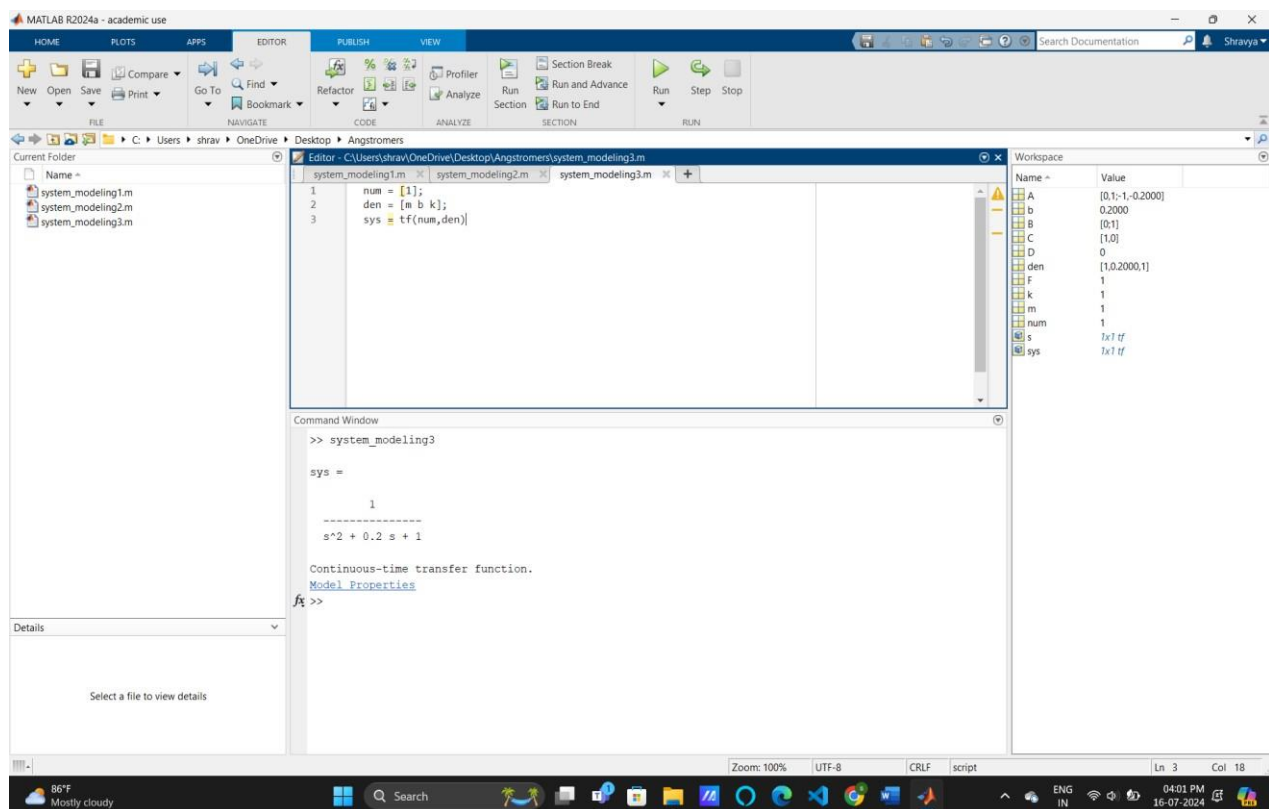
```
1 m = 1;
2 k = 1;
3 b = 0.2;
4 F = 1;
5
6 A = [0 1; -k/m -b/m];
7 B = [0 1/m]';
8 C = [1 0];
9 D = [0];
10
11 sys = ss(A,B,C,D)
```

The Command Window shows the resulting matrices:

```
A =          x1    x2
        x1      0      1
        x2     -1     -0.2

B =          u1
        x1      0
        x2      1
```

The Workspace window remains the same as in the previous screenshot.



Learnings

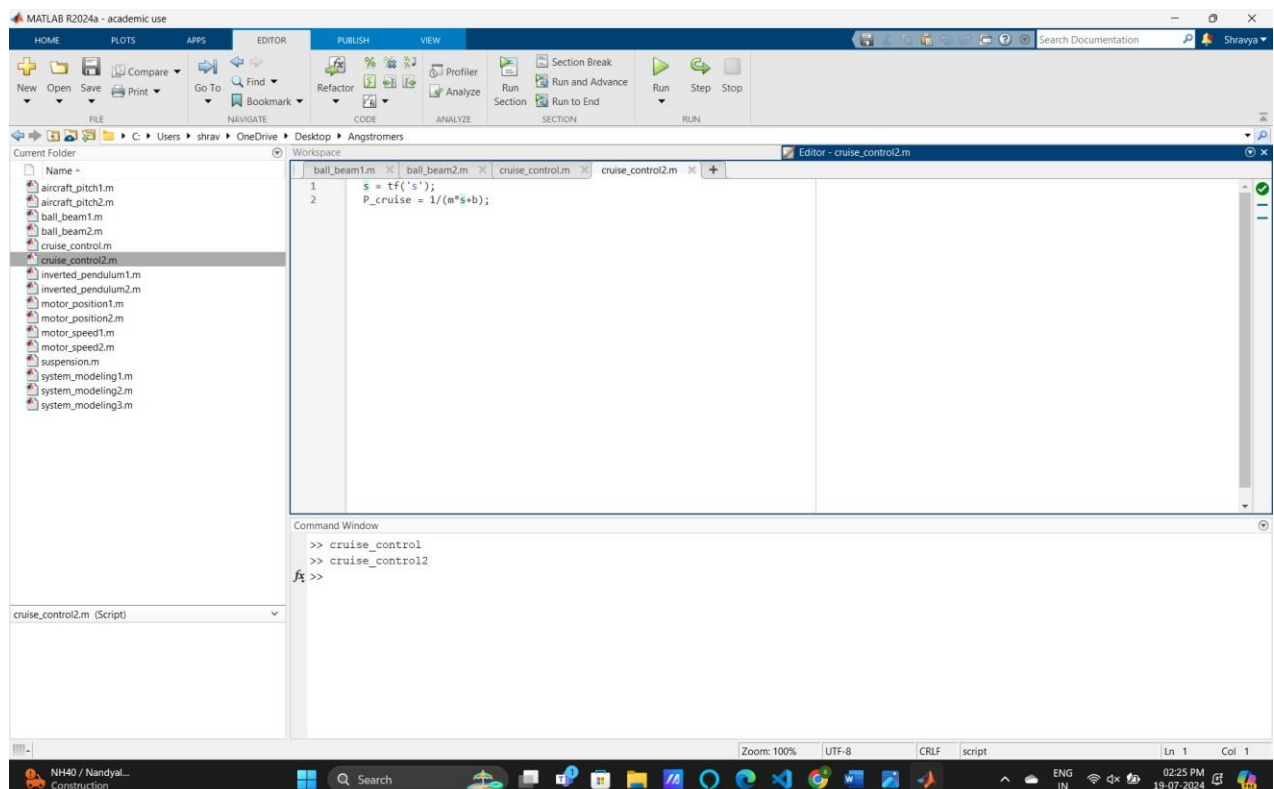
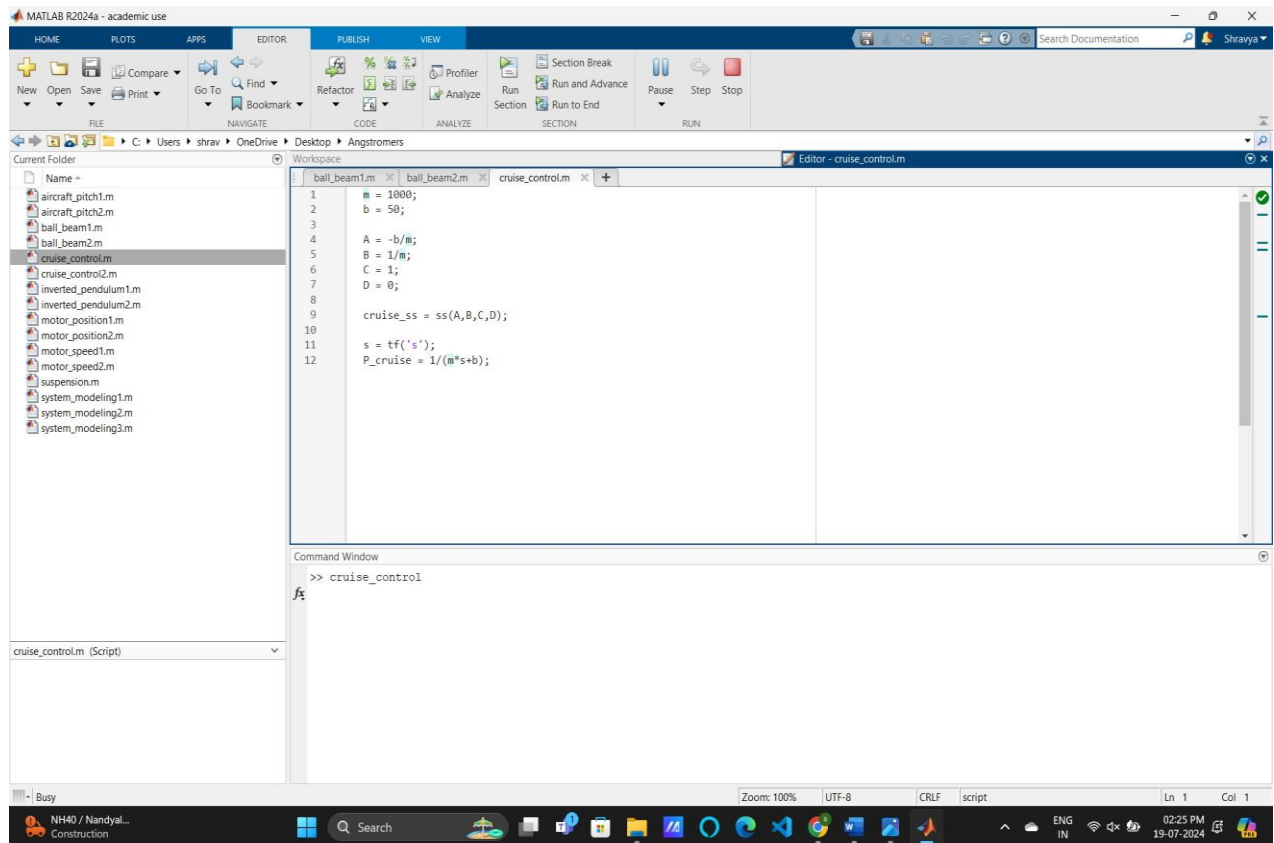
1. **Dynamic Systems:** Grasping that dynamic systems change according to established rules, often depicted by differential equations.
2. **State-Space Representation:** Understanding how to model systems using state-space representations, which use matrices to describe the relationships between inputs, state variables, and outputs.
3. **Transfer Function Representation:** Learning to apply Laplace transforms to convert time-domain models into frequency-domain models, making analysis easier by focusing on input-output relationships.
4. **Mechanical Systems Modeling:** Using Newton's laws to model mechanical systems, such as the mass-spring-damper system, and deriving both state-space and transfer function representations.
5. **Electrical Systems Modeling:** Applying Kirchhoff's laws to model electrical systems, like RLC circuits, and drawing parallels between mechanical and electrical systems for better comprehension.
6. **System Identification:** Acknowledging the necessity of experimental data and statistical methods to model systems with unknown or complex theoretical parameters.
7. **MATLAB Implementation:** Using MATLAB commands to create state-space and transfer function models for further analysis and control design.

Use Cases

1. **Control System Design:** Engineers can use the principles of system modeling to design controllers for a variety of mechanical and electrical systems, ensuring desired performance and stability.
2. **Simulation and Analysis:** Using state-space and transfer function models in MATLAB for simulating the behavior of dynamic systems and analyzing their response to different inputs.
3. **Predictive Maintenance:** Applying system identification techniques to develop models based on experimental data, which can predict system failures and schedule maintenance.
4. **Educational Purposes:** Teaching students and professionals about dynamic systems, modeling techniques, and MATLAB usage in control systems and engineering courses.
5. **Industrial Applications:** Implementing control systems in manufacturing, automotive, aerospace, and robotics industries to optimize performance and ensure safety.
6. **Research and Development:** Conduct research in advanced control strategies, system identification methods, and new modeling techniques to innovate and improve existing systems.
7. **Prototyping and Testing:** Creating prototype models of new systems to test their behavior under various conditions and refine designs before full-scale production.
8. **Complex System Analysis:** Breaking down complex systems into manageable models for analysis, allowing for better understanding and control of intricate processes like chemical reactions or power grids.

Cruise Control: System Modeling

Cruise control is an automatic feedback control system in modern vehicles designed to maintain a constant speed despite external disturbances such as wind changes or road inclines. By measuring the vehicle's speed, comparing it to a desired reference speed, and automatically adjusting the throttle, the system can keep a steady speed. This tutorial introduces a simplified model of vehicle dynamics for cruise control, covering the physical setup, system equations, parameters, state-space model, and transfer function model.



Learnings

1. **Physical Setup Understanding:** Grasping the basic setup of a cruise control system and the principles behind maintaining constant vehicle speed despite external disturbances.
2. **System Equations Derivation:** Learning how to derive the system equations using Newton's second law and understanding the relationship between force, mass, and damping in the context of vehicle dynamics.
3. **Parameter Identification:** Understanding the importance of system parameters like vehicle mass and damping coefficient and their impact on system behavior.
4. **State-Space Model Representation:** Learning how to represent a system using statespace models, identifying state variables, and writing the state-space equations.
5. **Transfer Function Representation:** Gaining knowledge on converting differential equations to transfer functions using Laplace transforms, and understanding the relationship between input and output in the frequency domain.
6. **MATLAB Implementation:** Becoming proficient in using MATLAB commands such as `ss` for state-space models and `tf` for transfer function models to represent and analyze dynamic systems.

Use Cases

1. **Vehicle Control Systems:** Engineers can apply these principles to design and implement cruise control systems in vehicles, ensuring they maintain a steady speed under varying conditions.
2. **Simulation and Testing:** Using the state-space and transfer function models in MATLAB for simulating the cruise control system's behavior and testing different control strategies.
3. **Educational Demonstrations:** Teaching students about dynamic systems, feedback control, and the use of MATLAB for system modeling and analysis.
4. **Prototyping:** Developing prototype models for new vehicle control systems and testing their performance in a simulated environment before actual implementation.
5. **Research:** Conducting research in the field of automotive control systems, exploring new methods for improving the performance and reliability of cruise control systems.
6. **Performance Optimization:** Using these models to optimize the performance of existing cruise control systems, enhancing fuel efficiency and ride comfort.
7. **Comparative Analysis:** Comparing different control strategies and their effectiveness in maintaining vehicle speed under various disturbances.

DC Motor Speed: System Modeling

DC motors are frequently used as actuators in control systems, offering rotary motion that can be transformed into translational motion using wheels or drums. This tutorial addresses the modeling of a DC motor's speed control system, detailing the physical setup, system equations, design requirements, and MATLAB representation in both transfer function and state-space forms. The system's input is the voltage applied to the motor's armature, while the output is the rotational speed of the shaft.

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Compare Go To Find Refactor Analyze Run Section Run and Advance Run Step Stop

FILE NAVIGATE CODE ANALYZE SECTION RUN

Current Folder: C:\Users\shrav\OneDrive\Desktop\Angstromers

Workspace: Editor - motor_speed1.m

```

1 D = 0.01;
2 b = 0.1;
3 K = 0.01;
4 R = 1;
5 L = 0.5;
6 s = tf('s');
7 P_motor = K/((J*s+b)*(L*s+R)+K*2)

```

Command Window

```

>> motor_speed1

P_motor =

          0.01
-----
0.005 s^2 + 0.06 s + 0.1001

Continuous-time transfer function.
Model Properties
fx >>

```

motor_speed1.m (Script)

Zoom: 100% UTF-8 CRLF script Ln 1 Col 1

85°F Mostly cloudy 02:40 PM 18-07-2024

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Compare Go To Find Refactor Analyze Run Section Run and Advance Run Step Stop

FILE NAVIGATE CODE ANALYZE SECTION RUN

Current Folder: C:\Users\shrav\OneDrive\Desktop\Angstromers

Workspace: Editor - motor_speed2.m

```

1 A = [-b/J K/J;
2      -K/L -R/L];
3 B = [0;
4      1/L];
5 C = [1 0];
6 D = 0;
7 motor_ss = ss(A,B,C,D)

```

Command Window

```

>> motor_speed2

motor_ss =

A =
      x1      x2
x1    -10      1
x2   -0.02    -2

B =
      u1
x1      0
x2      2

C =
      x1      x2
y1      1      0

D =
      u1
y1      0

Continuous-time state-space model.
Model Properties
fx >>

```

motor_speed2.m (Script)

Zoom: 100% UTF-8 CRLF script Ln 7 Col 23

85°F Mostly cloudy 02:40 PM 18-07-2024

Learnings

1. **Physical Setup Understanding:** Understanding the components of a DC motor system, including the armature, rotor, and the interaction between electrical and mechanical domains.
2. **System Equations Derivation:** Learning how to derive the system equations using Newton's 2nd law for rotational systems and Kirchhoff's voltage law for electrical circuits.
3. **Parameter Identification:** Understanding the role of physical parameters such as moment of inertia, friction constant, electromotive force constant, torque constant, resistance, and inductance in the behavior of the motor.
4. **Transfer Function Representation:** Learning to apply the Laplace transform to derive the transfer function of the system and understanding its significance in control systems.
5. **State-Space Model Representation:** Gaining knowledge on representing the system in state-space form, identifying state variables, and writing the state-space equations.
6. **Design Requirements Understanding:** Learning to specify and interpret performance criteria such as settling time, overshoot, and steady-state error for the control system.
7. **MATLAB Implementation:** Becoming proficient in using MATLAB commands such as `tf` for transfer function models and `ss` for state-space models to represent and analyze dynamic systems.

Use Cases

1. **Motor Control Systems:** Engineers can apply these principles to design and implement speed control systems for DC motors used in various applications like robotics, industrial machinery, and electric vehicles.
2. **Simulation and Testing:** Using the state-space and transfer function models in MATLAB to simulate the DC motor's behavior and testing different control strategies.
3. **Educational Demonstrations:** Teaching students about dynamic systems, electromechanical interactions, and the use of MATLAB for system modeling and analysis.
4. **Prototyping:** Developing prototype models for new DC motor control systems and testing their performance in a simulated environment before actual implementation.
5. **Performance Optimization:** Using these models to optimize the performance of existing DC motor control systems, improving response time and accuracy.
6. **Comparative Analysis:** Comparing different control strategies and their effectiveness in meeting design requirements such as settling time, overshoot, and steady-state error.
7. **Research:** Conducting research in the field of motor control systems, exploring new methods for enhancing the performance and reliability of DC motors.

DC Motor Position: System Modeling

DC motors are pivotal components in control systems, often employed in applications requiring rotational motion. They offer versatility in various systems, from simple mechanical devices to sophisticated robotic systems. Understanding and modeling the dynamics of DC motors is essential for designing effective control strategies and ensuring optimal performance. This tutorial focuses on modeling the position control of a DC motor using MATLAB, which involves deriving

mathematical models, converting them into different representations, and simulating them to meet specific design requirements.

The screenshot shows the MATLAB R2024a interface. The Editor window displays the script `motor_position1.m` with the following code:

```
1 J = 3.2284E-6;  
2 b = 3.5077E-6;  
3 K = 0.0274;  
4 R = 4;  
5 L = 2.75E-6;  
6 s = tf('s');  
7 P_motor = K/(s*((J*s+b)*(L*s+R)+K^2));
```

The Command Window shows the execution of `motor_position1`, resulting in the transfer function:

$$P_{\text{motor}} = \frac{0.0274}{8.878\text{e-}12\text{ s}^3 + 1.291\text{e-}05\text{ s}^2 + 0.0007648\text{ s}}$$

The Workspace window lists the variables: `A`, `b`, `B`, `C`, `cruise_ss`, `D`, `den`, `f`, `J`, `K`, `L`, `m`, `motor_ss`, `num`, `P_cruise`, `P_motor`, `R`, `s`, and `sys`.

The screenshot shows the MATLAB R2024a interface. The Editor window displays the script `motor_position2.m` with the following code:

```
1 A = [0 1 0  
2       0 -b/J K/J  
3       0 -K/L -R/L];  
4 B = [0; 0; 1/L];  
5 C = [1 0 0];  
6 D = [0];  
7  
8 motor_ss = ss(A,B,C,D);
```

The Command Window shows the execution of `motor_position2`, resulting in the state-space model:

$$\text{motor_ss} = \begin{matrix} A = & \begin{matrix} x1 & x2 & x3 \end{matrix} \\ \begin{matrix} x1 \\ x2 \\ x3 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1.087 & 8487 \\ 0 & -9964 & -1.455\text{e}+06 \end{bmatrix} \end{matrix}$$
$$\begin{matrix} B = & \begin{matrix} u1 \end{matrix} \\ \begin{matrix} x1 \\ x2 \\ x3 \end{matrix} & \begin{bmatrix} 0 \\ 0 \\ 3.636\text{e}+05 \end{bmatrix} \end{matrix}$$
$$\begin{matrix} C = & \begin{matrix} x1 & x2 & x3 \end{matrix} \\ \begin{matrix} y1 \end{matrix} & \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \end{matrix}$$
$$\begin{matrix} D = & \begin{matrix} u1 \end{matrix} \\ \begin{matrix} y1 \end{matrix} & \begin{bmatrix} 0 \end{bmatrix} \end{matrix}$$

The Command Window also displays the text: "Continuous-time state-space model."

Learnings

1. Physical Setup and Parameters:

- **Moment of Inertia (J)**: Represents the resistance of the motor's rotor to changes in its rotational speed.
- **Motor Viscous Friction Constant (b)**: Indicates the frictional resistance opposing the motor's rotation.
- **Electromotive Force Constant (K_b) and Motor Torque Constant (K_t)**: Determine the relationship between the motor's electrical input and its mechanical output.
- **Electric Resistance (R) and Inductance (L)**: Define the electrical characteristics affecting the motor's current and voltage dynamics.

2. System Equations:

- **Torque Equation ($T = K_t \cdot i$)**: Shows the proportionality between the torque produced by the motor and the armature current.
- **Back EMF Equation ($e = K_b \cdot \omega$)**: Relates the back electromotive force to the angular velocity of the motor's shaft.
- **Governing Equations**:
 - ✦ **Newton's 2nd Law**: Relates the motor's torque to its rotational dynamics.
 - ✦ **Kirchhoff's Voltage Law**: Describes the electrical dynamics within the motor circuit.

3. Transfer Function:

- **Laplace Transforms**: Convert the differential equations into the Laplace domain to obtain the transfer function.
- **Position Transfer Function ($\Theta(s) / V(s)$)**: Determines how the motor's position responds to changes in the input voltage.
- **Speed Transfer Function**: Initially obtained and used to derive the position transfer function by dividing by s.

4. State-Space Representation:

- **State Variables**: Include motor position, speed, and armature current.
- **State-Space Equations**:
 - ✦ **State Matrix (A)**: Describes the system dynamics.
 - ✦ **Input Matrix (B)**: Represents how the input voltage affects the state variables.
 - ✦ **Output Matrix (C)**: Relates the state variables to the output (motor position).
 - ✦ **Feedthrough Matrix (D)**: Represents direct influence of the input on the output.

5. MATLAB Implementation:

- **Transfer Function Model**: Implemented using the `tf` command to define and manipulate the motor's transfer function.
- **State-Space Model**: Implemented using the `ss` command to define and analyze the system's state-space representation.

Use Cases

1. Robotics:

- **Motion Control**: DC motors are used in robotic arms and wheels. Precise modeling ensures accurate control over movement and positioning.
- **Actuation Systems**: Essential for tasks requiring precise rotary motion and quick response.

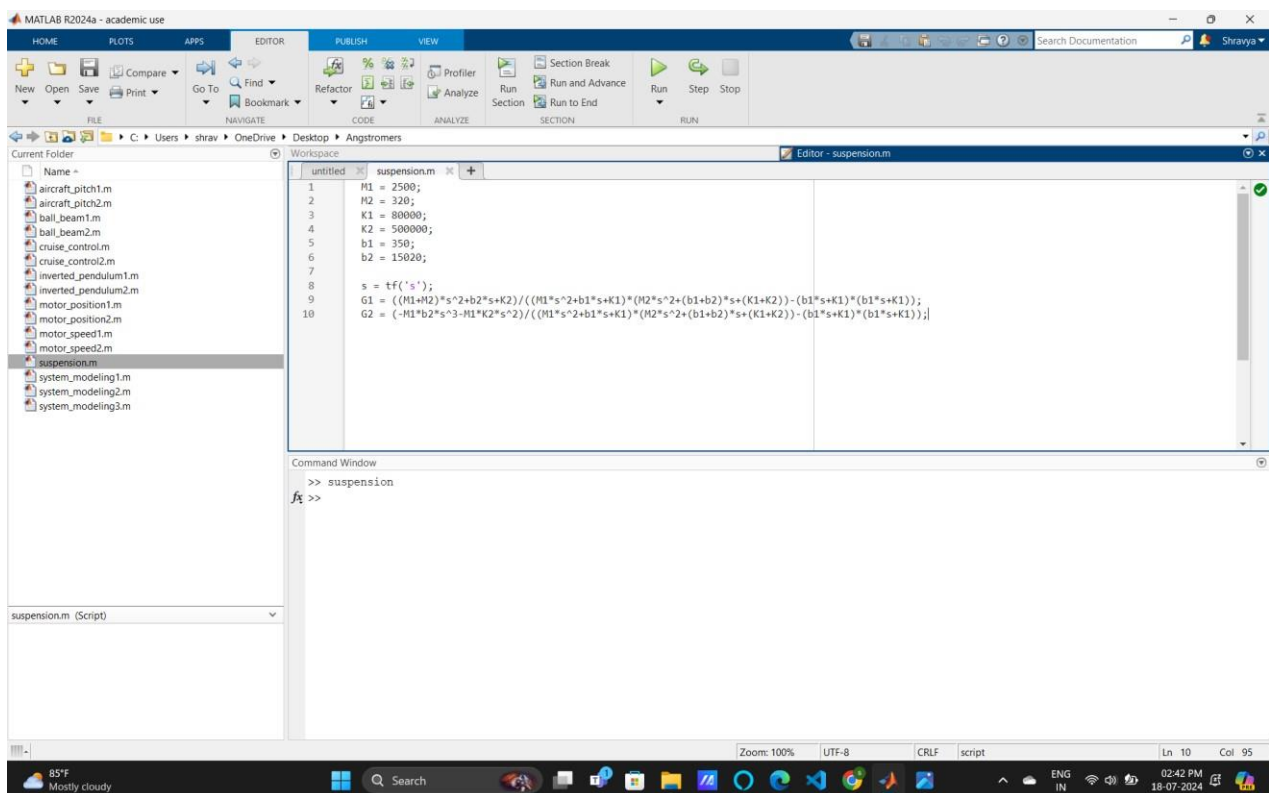
2. Automation Systems:

- **Manufacturing**: In automated production lines, DC motors control conveyor belts and other moving parts. Accurate models help in designing efficient control systems.

- **Process Control:** For regulating machinery and ensuring consistent performance.
- 3. **Industrial Machinery:**
 - **Conveyor Systems:** DC motors drive conveyors and other mechanical systems. Modeling helps in optimizing performance and maintaining reliability.
 - **Positioning Systems:** Used in equipment requiring precise positioning and speed control.
- 4. **Precision Engineering:**
 - **CNC Machines:** DC motors control the movement of cutting tools and workpieces. Modeling ensures high precision and stability in operations.
 - **Optical Systems:** For adjusting lenses and mirrors with high accuracy.
- 5. **Educational Purposes:**
 - **Control Theory:** Provides a practical example for understanding dynamic systems, feedback control, and system modeling.
 - **Simulation:** Helps students and engineers visualize and analyze the behavior of DC motors in various scenarios.

Suspension: System Modeling

Designing an automotive suspension system is a complex control problem that can be simplified by using a 1/4 model, representing one of the four wheels. This model is a 1-D multiple spring-damper system. The active suspension system model includes an actuator that generates a control force (U) to control the motion of the vehicle body. The system parameters and equations of motion are defined, and the dynamic equations are converted into transfer function models for analysis and simulation using MATLAB.



The image shows the MATLAB R2024a - academic use interface. The main window displays a script named 'suspension.m' with the following code:

```

1 M1 = 2500;
2 M2 = 320;
3 K1 = 80000;
4 K2 = 500000;
5 b1 = 350;
6 b2 = 15020;
7
8 s = tf('s');
9 G1 = ((M1+M2)*s^2+b2*s+K2)/((M1*s^2+b1*s+K1)*(M2*s^2+(b1+b2)*s+(K1+K2))-(b1*s+K1)*(b1*s+K1));
10 G2 = (-M1*b2*s^3-M1*K2*s^2)/((M1*s^2+b1*s+K1)*(M2*s^2+(b1+b2)*s+(K1+K2))-(b1*s+K1)*(b1*s+K1));

```

The Command Window shows the command `>> suspension` and the prompt `>>`. The left sidebar shows the current folder structure, including files like 'aircraft_pitch1.m', 'aircraft_pitch2.m', 'ball_beam1.m', 'ball_beam2.m', 'cruise_control.m', 'cruise_control2.m', 'inverted_pendulum1.m', 'inverted_pendulum2.m', 'motor_position1.m', 'motor_position2.m', 'motor_speed1.m', 'motor_speed2.m', 'suspension.m', 'system_modeling1.m', 'system_modeling2.m', and 'system_modeling3.m'.

Learnings

1. Physical Setup and System Parameters

- **Physical Setup:** The 1/4 model simplifies the suspension system into a 1-D multiple spring-damper system.
- **System Parameters:** Parameters include the mass of the bus body and suspension, spring constants, and damping constants. These parameters are critical for defining the dynamics of the system.

2. Equations of Motion

- Derived using Newton's law for the motion of the bus body and suspension mass.
- The equations incorporate the control force U and the dynamics between the bus body and the suspension.

- ## 3. Transfer Function Models
- The equations of motion are transformed into transfer functions by taking the Laplace Transform.
 - Transfer functions $G_1(s)$, $G_2(s)$, and $G_3(s)$ are derived to represent the system's response to control force U and disturbance input W , respectively.

4. MATLAB Implementation

- MATLAB commands are used to define the system parameters and create transfer function models.
- The `tf` command in MATLAB creates transfer functions, and the models can be used for further analysis, such as step response or frequency response.

Use Cases

1. Control System Design

- **Active Suspension Systems:** Designing active suspension systems to improve vehicle ride quality and handling by controlling the motion of the bus body.
- **Optimization:** Using the transfer function models to optimize the suspension parameters for better performance under various driving conditions.

2. Simulation and Analysis

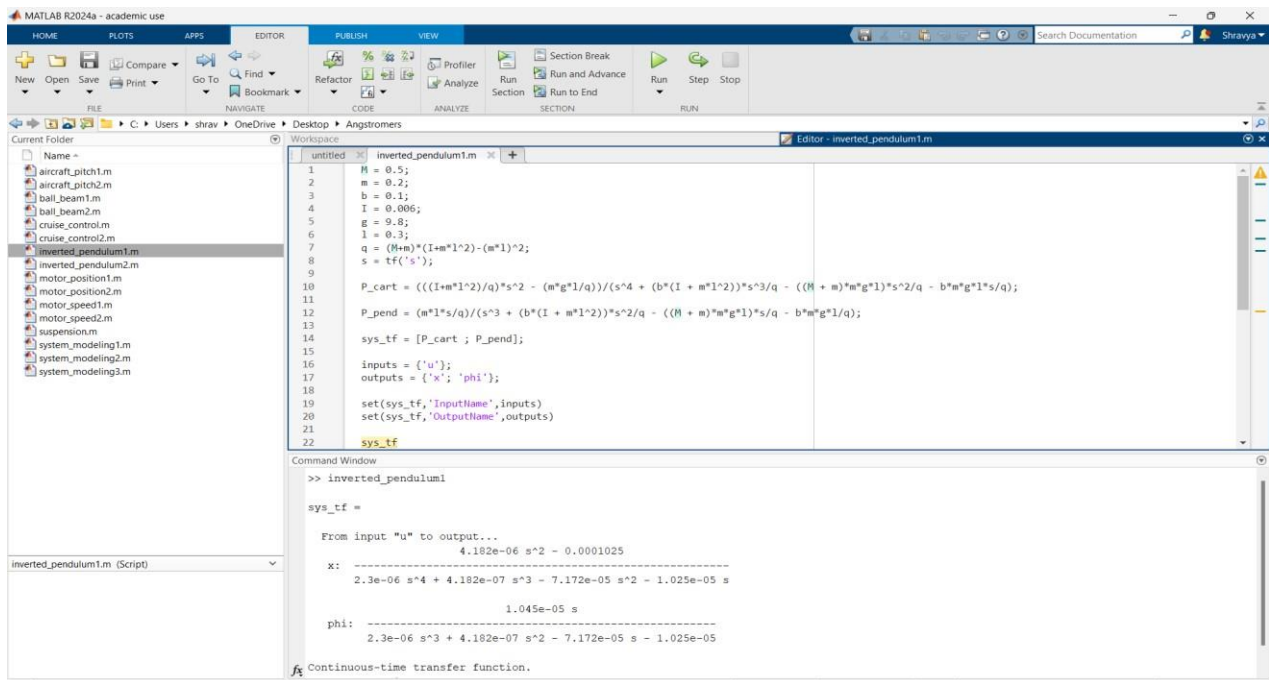
- **Simulation:** Simulating the suspension system's response to different inputs, such as road disturbances or control forces, using MATLAB.
- **Analysis:** Analyzing the system's performance, stability, and robustness by examining the transfer function responses.

3. Educational Purposes

- **Teaching Tool:** Using the simplified 1/4 model and MATLAB implementation to teach control system design and analysis in engineering courses.
- **Research:** Conducting research on advanced suspension systems and control strategies by extending the basic model to more complex scenarios.

Inverted Pendulum: System Modeling

Inverted pendulum systems are fundamental in the study of control systems due to their inherent instability and nonlinear dynamics. This tutorial guides you through the modeling of an inverted pendulum mounted on a motorized cart using MATLAB. The objective is to balance the pendulum in an upright position by applying a horizontal force to the cart. This example provides insight into both transfer function and state-space representations, essential tools in control system design.

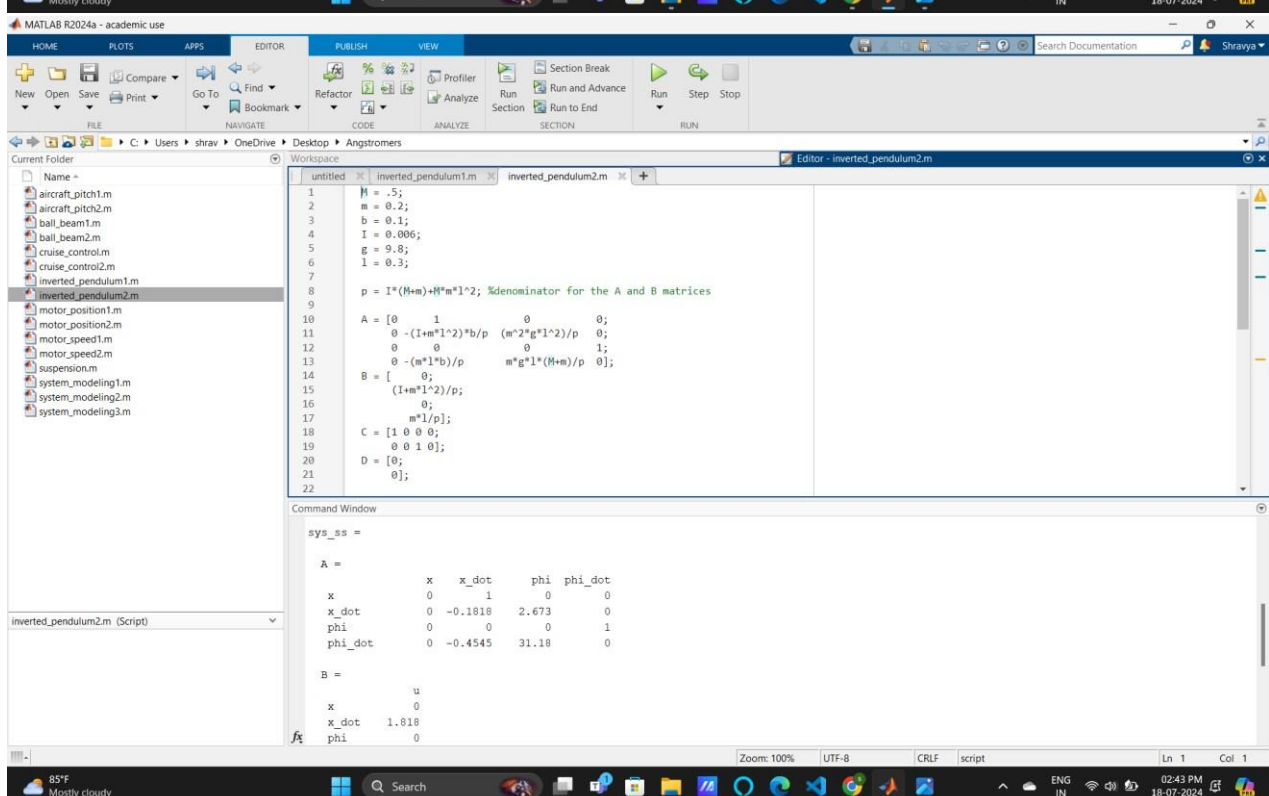


The screenshot displays the MATLAB R2024a environment with the 'inverted_pendulum1.m' script open. The script defines the system parameters and constructs a transfer function model. The Command Window shows the resulting transfer function for the system.

```
1 H = 0.5;  
2 m = 0.2;  
3 b = 0.1;  
4 I = 0.006;  
5 g = 9.8;  
6 l = 0.3;  
7 q = (H+m)*(I+m*l^2)-(m*l)^2;  
8 s = tf('s');  
9  
10 P_cart = ((I+m*l^2)/q)*s^2 - (m*g*l/q)/(s^4 + (b*(I + m*l^2))*s^3/q - ((H + m)*m*g*l)*s^2/q - b*m*g*l*s/q);  
11  
12 P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((H + m)*m*g*l)*s/q - b*m*g*l/q);  
13  
14 sys_tf = [P_cart ; P_pend];  
15  
16 inputs = {'u'};  
17 outputs = {'x'; 'phi'};  
18  
19 set(sys_tf,'InputName',inputs)  
20 set(sys_tf,'OutputName',outputs)  
21  
22 sys_tf
```

Command Window:

```
>> inverted_pendulum1  
  
sys_tf =  
  
From input "u" to output...  
4.182e-06 s^2 - 0.0001025  
-----  
x: 2.3e-06 s^4 + 4.182e-07 s^3 - 7.172e-05 s^2 - 1.025e-05 s  
-----  
1.045e-05 s  
-----  
phi: 2.3e-06 s^3 + 4.182e-07 s^2 - 7.172e-05 s - 1.025e-05  
-----  
Continuous-time transfer function.
```



The screenshot displays the MATLAB R2024a environment with the 'inverted_pendulum2.m' script open. The script defines the system parameters and constructs a state-space model. The Command Window shows the resulting state-space matrices.

```
1 H = .5;  
2 m = 0.2;  
3 b = 0.1;  
4 I = 0.006;  
5 g = 9.8;  
6 l = 0.3;  
7  
8 p = I*(H+m)+H*m*l^2; %denominator for the A and B matrices  
9  
10 A = [0 1 0 0;  
11 0 -(I+m*l^2)*b/p (m^2*g*l^2)/p 0;  
12 0 0 (m*l*b)/p m*g*l*(H+m)/p 1;  
13 0 -(m*l*b)/p m*g*l*(H+m)/p 0];  
14 B = [0;  
15 (I+m*l^2)/p;  
16 0;  
17 m*l/p];  
18 C = [1 0 0 0;  
19 0 0 1 0];  
20 D = [0;  
21 0];  
22
```

Command Window:

```
sys_ss =  
  
A =  
  
x x_dot phi phi_dot  
x 0 1 0 0  
x_dot 0 -0.1818 2.673 0  
phi 0 0 0 1  
phi_dot 0 -0.4545 31.18 0  
  
B =  
  
u  
x 0  
x_dot 1.818  
phi 0
```

Key Learnings

1. **Modeling Complex Systems:** Learn to derive and linearize the equations of motion for a nonlinear system.
2. **MATLAB Representation:** Understand how to use MATLAB commands to model systems both in transfer function and state-space forms.
3. **Control System Design:** Gain knowledge in designing controllers to meet specific performance criteria such as settling time, rise time, and steady-state error.
4. **Analysis Techniques:** Familiarize with techniques such as force analysis, Laplace transforms, and linear approximations for dynamic systems.
5. **State-Space vs. Transfer Function:** Comprehend the differences and conversions between state-space models and transfer function models, recognizing the benefits of each approach.

Use Cases

1. **Educational Tools:** This tutorial serves as an excellent educational resource for students and instructors in control systems courses, providing a practical example of theoretical concepts.
2. **Robotics:** In robotics, inverted pendulum models are used to design and control bipedal robots and self-balancing robots like segways.
3. **Aerospace Engineering:** The principles applied here can be extended to control systems in aerospace engineering, such as the attitude control of rockets and spacecraft.
4. **Automotive Systems:** This modeling technique is useful in the design of vehicle stability control systems to enhance safety and performance.
5. **Industrial Automation:** Inverted pendulum models can be applied to balance and control various industrial automation systems where stability and precision are crucial.

Aircraft Pitch: System Modeling

This tutorial illustrates the process of modeling an aircraft's pitch control system using MATLAB. It includes deriving the system equations, converting them to transfer function and state-space representations, and defining design requirements for the system. The objective is to design an autopilot that manages the aircraft's pitch under certain simplified assumptions.

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Tools Window Help

Current Folder: Desktop > Angstromers

Workspace: aircraft_pitch1.m

```

1 s = tf('s');
2 P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s)

```

Command Window

```

>> aircraft_pitch1

P_pitch =

    1.151 s + 0.1774
    -----
    s^3 + 0.739 s^2 + 0.921 s

Continuous-time transfer function.
Model Properties
fx >>

```

aircraft_pitch1.m (Script)

Zoom: 100% UTF-8 CRLF script Ln 1 Col 1

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Tools Window Help

Current Folder: Desktop > Angstromers

Workspace: aircraft_pitch1.m, aircraft_pitch2.m

```

1 A = [-0.313 56.7 0; -0.0139 -0.426 0; 0 56.7 0];
2 B = [0.232; 0.0203; 0];
3 C = [0 0 1];
4 D = [0];
5 pitch_ss = ss(A,B,C,D)

```

Command Window

```

>> aircraft_pitch2

pitch_ss =

A =

    x1    x2    x3
x1  -0.313  56.7    0
x2  -0.0139 -0.426    0
x3    0    56.7    0

B =

    u1
x1  0.232
x2  0.0203
x3    0

C =

    x1 x2 x3
y1    0  0  1

D =

    u1
y1    0

Continuous-time state-space model.
Model Properties
fx >>

```

aircraft_pitch2.m (Script)

Zoom: 100% UTF-8 CRLF script Ln 1 Col 1

Learnings

1. **Understanding System Dynamics:** Gain insights into the longitudinal dynamics of an aircraft and how they can be modeled using differential equations.
2. **Linearization of Equations:** Learn the importance of simplifying complex nonlinear systems to linear models for easier analysis and control design.
3. **Transfer Function Representation:** Understand how to derive the transfer function of a system from its differential equations.
4. **State-Space Representation:** Learn how to represent a system in state-space form, which is particularly useful for modern control design techniques.
5. **MATLAB Commands:** Familiarize yourself with key MATLAB commands (tf, ss) used for creating transfer function and state-space models.
6. **Control System Design:** Understand the design criteria for a control system, including overshoot, rise time, settling time, and steady-state error.

Use Cases

1. **Autopilot Design:** The principles and methods discussed can be applied to design autopilot systems for aircraft, ensuring stability and desired performance in pitch control.
2. **Educational Purposes:** This tutorial serves as a comprehensive guide for students and educators in aerospace engineering and control systems courses.
3. **Simulation and Analysis:** Engineers can use the techniques to simulate and analyze the behavior of aircraft control systems under various conditions.
4. **Prototyping Control Algorithms:** The tutorial provides a foundation for prototyping and testing new control algorithms in a simulated environment before real-world implementation.
5. **System Optimization:** By understanding the system's dynamics and control requirements, engineers can optimize the performance of existing control systems.

Ball & Beam: System Modeling

This tutorial centers on modeling the dynamics of a ball-on-beam system using MATLAB. The system involves a ball rolling along a beam, with the beam's angle controlled by a servo mechanism. The goal is to design a control system that can adjust the beam's angle to control the ball's position. The tutorial includes deriving system equations, transfer function and state-space representations, and applying design criteria to achieve the desired system performance.

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Command Window

Current Folder: C:\Users\shrav\OneDrive\Desktop\Angstromers

Workspace: untitled, ball_beam1.m

Editor - ball_beam1.m

```

1 m = 0.111;
2 R = 0.015;
3 g = -9.8;
4 L = 1.0;
5 d = 0.03;
6 J = 9.99e-6;
7
8 s = tf('s');
9 P_ball = -m*g*d/L/(J/R^2+m)/s^2

```

Command Window

```

>> ball_beam1

P_ball =

    0.21
    ----
    s^2

Continuous-time transfer function.
Model Properties
f>>

```

ball_beam1.m (Script)

85°F Mostly cloudy 02:44 PM 18-07-2024

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Command Window

Current Folder: C:\Users\shrav\OneDrive\Desktop\Angstromers

Workspace: untitled, ball_beam1.m, ball_beam2.m

Editor - ball_beam2.m

```

1 H = -m*g/(J/(R^2)+m);
2 A = [0 1 0 0
3       0 0 H 0
4       0 0 0 1
5       0 0 0 0];
6 B = [0 0 0 1]';
7 C = [1 0 0 0];
8 D = [0];
9 ball_ss = ss(A,B,C,D)

```

Command Window

```

ball_ss =

A =

    x1    x2    x3    x4
x1    0     1     0     0
x2    0     0     7     0
x3    0     0     0     1
x4    0     0     0     0

B =

    u1
x1    0
x2    0
x3    0
x4    1

C =

    x1    x2    x3    x4
y1    1     0     0     0

D =

    u1
y1    0

```

ball_beam2.m (Script)

85°F Mostly cloudy 02:44 PM 18-07-2024

Learnings

1. **Physical System Understanding:** Understand the physical setup of a ball-on-beam system and how the motion is influenced by the beam's angle.
2. **System Parameters:** Learn about key system parameters such as mass, radius, lever arm offset, gravitational acceleration, and the moment of inertia.
3. **Linearization of Equations:** Gain insights into the process of linearizing nonlinear equations to simplify the modeling and control design.
4. **Transfer Function Representation:** Understand how to derive the transfer function from the linearized system equations.
5. **State-Space Representation:** Learn how to represent the system in state-space form, which is essential for modern control design techniques.
6. **MATLAB Commands:** Familiarize yourself with MATLAB commands (tf, ss) used for creating transfer function and state-space models.
7. **Control Design Criteria:** Learn about design criteria such as settling time and overshoot, and how they influence the control system design.

Use Cases

1. **Control System Design:** Apply the principles and methods discussed to design control systems for similar mechanical systems, ensuring stability and desired performance.
2. **Educational Purposes:** Serve as a comprehensive guide for students and educators in control systems and mechanical engineering courses.
3. **Simulation and Analysis:** Use the techniques to simulate and analyze the behavior of control systems under various conditions.
4. **Prototyping Control Algorithms:** Provide a foundation for prototyping and testing new control algorithms in a simulated environment before real-world implementation.
5. **System Optimization:** By understanding the system's dynamics and control requirements, optimize the performance of existing control systems.

Summary of Key Findings

1. **Modeling Dynamics:**
 - The ball-on-beam system was modeled by linearizing the nonlinear equations of motion.
 - The system equations were simplified assuming negligible friction and rolling without slipping.
 - Linear approximations provided practical ways to describe the system dynamics for control purposes.
2. **Transfer Function:**
 - The transfer function from the gear angle to the ball position was derived. ○ It was found to be a double integrator, which implies marginal stability and poses a control challenge. ○ MATLAB commands were used to create the transfer function model for simulation and analysis.
3. **State-Space Representation:**
 - The state-space representation was derived, highlighting the relationship between the state variables (ball position and velocity) and the input (gear angle or torque).

- A state-space model allows for the application of modern control design techniques.
- MATLAB was used to represent and simulate the state-space model.

4. Design Criteria:

- Specific design criteria were set, including settling time, overshoot, and steady-state error, which guide the control system design to ensure desired performance.
- These criteria ensure the system responds quickly, with minimal overshoot and accurate steady-state behavior.

Implications for Control Systems

1. Stability and Performance:

- Ensuring stability in systems with inherent challenges (like double integrators) is crucial.
- Design criteria help achieve not only stability but also performance goals like quick response and minimal overshoot.

2. Practical Implementation:

- The simplified models and linear approximations allow for practical implementation of control systems in real-world applications.
- Understanding the dynamics through both transfer function and state-space representations provides flexibility in choosing control strategies.

3. Educational Value:

- The methodologies used in this tutorial serve as an excellent educational tool for learning control system design and analysis.
- Students and practitioners can gain hands-on experience with MATLAB, enhancing their understanding of theoretical concepts through simulation.

4. Advanced Control Techniques:

- The state-space model lays the groundwork for applying advanced control techniques such as state feedback, observers, and optimal control.
- These techniques can be used to further improve system performance and robustness.

5. System Optimization:

- By understanding the dynamic behavior and control requirements, existing systems can be optimized for better performance.
- This approach can lead to more efficient and effective control solutions in various engineering applications.

Conclusion

The tutorial offers a thorough approach to modeling, analyzing, and designing control systems for the ball-on-beam system. It emphasizes the significance of linearization, transfer function, and state-space representations in comprehending system dynamics. Utilizing MATLAB for simulation and analysis aids in practical implementation and further optimization of control systems. These insights and methods have wide-ranging implications for both educational purposes and real-world engineering applications, facilitating the creation of stable and high-performance control systems.