

REPORT

Control Systems

Trainee Name : Shaik Athiya

Date of Submission : 22 July 2024

Training Program Name : Angstromers Engineering
Solution Pvt.Ltd

Brief Summary:

Objective:

The control system project aimed to develop mathematical models for dynamic systems, utilizing state-space and transfer function representations. By applying these modeling techniques to both mechanical and electrical systems, the project demonstrated how to effectively describe and analyze system behaviors. MATLAB was used extensively to implement and validate these models, showcasing its capabilities in handling complex system analyses and providing valuable insights into the dynamics of various systems.

Methods Used:

- System Modeling
- State-Space Representation
- Transfer Function Representation
- Mechanical Systems Modeling
- Electrical Systems Modeling
- System Identification and Conversions

Results:

Successfully derived and implemented state-space and transfer function models. Validated models through MATLAB simulations. The methods can be applied to a variety of systems, enhancing understanding and control of dynamic behaviors.

Conclusion:

State-space and transfer function representations are effective for modeling and analyzing dynamic systems. MATLAB provides robust tools for implementing and validating these models.

Introduction: System Modeling

The control design process starts with developing mathematical models of the system to be controlled. These models can be derived from physical laws or experimental data. This tutorial covers state-space and transfer function representations of dynamic systems, along with approaches to modeling mechanical and electrical systems, demonstrating how to implement these models in MATLAB for further analysis.

Key MATLAB Commands

- `ss` : Define state-space models.
- `tf` : Define transfer function models.

Related Resources

- **MATLAB Basics**
- **RC Circuit Activity**
- **Pendulum Activity**
- **External Videos:** MATLAB State Space, Modeling Intro

Dynamic Systems

Dynamic systems evolve over time according to a fixed rule, often described by first-order differential equations:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \quad \frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \quad \mathbf{x}' = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$$

- **State Vector ($\mathbf{x}(t)$)**: Represents the system configuration at time t .
- **Input Vector ($\mathbf{u}(t)$)**: External inputs to the system.

- **Function \mathbf{f} :** Produces the time derivative of the state vector.

State-Space Representation

For continuous linear time-invariant (LTI) systems, the state-space representation is:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\end{aligned}$$

- \mathbf{x} : State variables vector
- \mathbf{u} : Input/control vector
- \mathbf{y} : Output vector
- $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$: System matrices

Example: Mass-Spring-Damper System

Governing Equation

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

State-Space Model

$$\begin{aligned}\dot{\mathbf{x}} &= \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t) \\ \mathbf{y} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = x\end{aligned}$$

Example: RLC Circuit

Governing Equation

$$V(t) - Ri - L\frac{di}{dt} - \frac{1}{C}\int i dt = 0$$

State-Space Model

$$\mathbf{\dot{x}} = \begin{bmatrix} 0 & 1 \\ -1/LC & -R/L \end{bmatrix} \mathbf{q} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} V(t)$$
$$\mathbf{x}' = \begin{bmatrix} i \\ q \end{bmatrix}$$

System Identification

When physical parameters are unknown, experimental data and statistical techniques are used to develop system models, a process known as system identification.

System Conversions

MATLAB supports converting between transfer function, state-space, and zero-pole-gain forms, facilitating various analysis and control design tasks.

Cruise Control: System Modeling

we model an automatic cruise control system, a classic example of a feedback control system used to maintain a vehicle's constant speed despite external disturbances. Here's a detailed breakdown:

Physical Setup

The cruise control system in a vehicle adjusts the throttle to maintain a set speed. The system can be simplified as a mass-damper system where the control force u represents the force at the road/tire interface. The resistive force bv due to rolling resistance and wind drag varies linearly with the vehicle velocity v .

System Equations

The system equation derived from Newton's 2nd law is: $m\dot{v} + bv = u$

The output equation, since we're interested in controlling the speed v , is: $y = v$

System Parameters

For our example:

- Vehicle mass (m): 1000 kg
- Damping coefficient (b): 50 N.s/m

State-Space Model

The state-space representation for a first-order system like ours (which has only one state variable, the velocity v) is:

$$\dot{\mathbf{x}} = \begin{bmatrix} -b/m \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/m \end{bmatrix} u$$

$$y = [1]v \quad y = [1]v \quad y = [1]v$$

Transfer Function Model

Taking the Laplace transform of the differential equation and assuming zero initial conditions, the transfer function $P(s)$ of the cruise control system is:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms + b} \quad P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms + b}$$

Summary of Key MATLAB Commands

- **ss(A, B, C, D):** Defines the state-space model.
- **tf('s'):** Defines the Laplace variable s .
- **1/(m*s + b):** Defines the transfer function model.

Conclusion

By using these models, you can simulate and analyze the behavior of the cruise control system under various conditions and design appropriate controllers to maintain the desired vehicle speed.

DC Motor Speed: System Modeling

This model explains how to model a DC motor in MATLAB using both transfer function and state-space representations. Here's a detailed breakdown:

Physical Setup

A DC motor provides rotary motion and, when coupled with wheels or cables, can offer translational motion. The input to the system is the voltage V applied to the motor's armature, and the output is the rotational speed of the shaft $\dot{\theta}$. The motor's physical parameters are as follows:

- J : Moment of inertia of the rotor (0.01 kg.m²)
- b : Motor viscous friction constant (0.1 N.m.s)
- K_e : Electromotive force constant (0.01 V/rad/sec)
- K_t : Motor torque constant (0.01 N.m/Amp)
- R : Electric resistance (1 Ohm)
- L : Electric inductance (0.5 H)

System Equations

1. Torque and Back EMF Equations:

- $T = K_t i$ (Torque)
- $e = K_e \dot{\theta}$ (Back EMF)

Since $K_t = K_e = K$, we simplify to:

2. Newton's 2nd Law and Kirchhoff's Voltage Law:

- $J\ddot{\theta} + b\dot{\theta} = K i$
- $L\frac{di}{dt} + Ri = V - K\dot{\theta}$

Transfer Function

By applying the Laplace transform and eliminating $I(s)$ between the equations, we derive the transfer function:

$$P(s) = \Theta'(s)V(s) = K(Js+b)(Ls+R) + K^2 P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R) + K^2} P(s) = V(s)\Theta'(s) = (Js+b)(Ls+R) + K^2 K$$

State-Space Representation

The state-space model is:

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} -b/J & -K/L \\ 1/L & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} V \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \end{aligned}$$

Design Requirements

For a unit step command in motor speed, the system should meet the following:

- Settling time: < 2 seconds
- Overshoot: < 5%
- Steady-state error: < 1%

Summary of Key MATLAB Commands

- **tf('s')**: Defines the Laplace variable s.
- **ss(A, B, C, D)**: Defines the state-space model.
- **ss(P_motor)**: Converts the transfer function model to state-space.

These models allow you to simulate and analyze the DC motor's behavior and design appropriate controllers to meet performance specifications.

DC Motor Position: System Modeling

This model covers the modeling of a DC motor in MATLAB using both transfer function and state-space representations. Below are the details:

Physical Setup

A DC motor provides rotary motion and can be coupled with wheels or cables to provide translational motion. The input to the system is the voltage V applied to the motor's armature, and the output is the position of the shaft θ . Here are the physical parameters derived from experiments:

- J : Moment of inertia of the rotor ($3.2284 \times 10^{-6} \text{ kg.m}^2$)
- b : Motor viscous friction constant ($3.5077 \times 10^{-6} \text{ N.m.s}$)
- K_b : Electromotive force constant (0.0274 V/rad/sec)
- K_t : Motor torque constant (0.0274 N.m/Amp)
- R : Electric resistance (4 Ohm)
- L : Electric inductance ($2.75 \times 10^{-6} \text{ H}$)

System Equations

Torque and Back EMF Equations

1. Torque equation:

$$T = K_t i$$

2. Back EMF equation:

$$e = K_b \dot{\theta}$$

Since $K_t = K_b = K$, the equations simplify to:

Newton's 2nd Law and Kirchhoff's Voltage Law

3. Newton's 2nd law: $J\ddot{\theta} + b\dot{\theta} = K_i I$
 $J\ddot{\theta} + b\dot{\theta} = K_i I$

4. Kirchhoff's voltage law:

$$L \frac{di}{dt} + R_i = V - K \dot{\theta}$$
$$+ R_i = V - K \dot{\theta}$$

Transfer Function

By applying the Laplace transform and eliminating $I(s)$, we derive the open-loop transfer function:

$$P(s) = \frac{\dot{\theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2}$$
$$P(s) = \frac{\dot{\theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2}$$

Since we are interested in position θ as the output, we integrate the speed to get:

$$\Theta(s) = \frac{\theta(s)}{V(s)} = \frac{K}{s((Js + b)(Ls + R) + K^2)}$$
$$\Theta(s) = \frac{\theta(s)}{V(s)} = \frac{K}{s((Js + b)(Ls + R) + K^2)}$$

State-Space Representation

The differential equations can be expressed in state-space form by choosing motor position, motor speed, and armature current as state variables:

Design Requirements

For a unit step input, the motor position output should meet these requirements:

- Settling time: < 40 milliseconds
- Overshoot: < 16%
- No steady-state error, even with a step disturbance input

Summary of Key MATLAB Commands

- **tf('s')**: Defines the Laplace variable s .
- **ss(A, B, C, D)**: Defines the state-space model.
- **ss(P_motor)**: Converts the transfer function model to state-space.

These models allow for simulation and analysis of the DC motor's behavior, facilitating the design of controllers to meet performance specifications.

Suspension: System Modeling

This model explores the modeling of an automotive suspension system in MATLAB using transfer functions. Below are the details:

Physical Setup

An automotive suspension system can be modeled as a 1-D multiple spring-damper system. This example focuses on a quarter model of the suspension system, including an actuator that generates the control force UUU to control the bus body motion.

System Parameters

- $M1$: 1/4 bus body mass = 2500 kg
- $M2$: Suspension mass = 320 kg
- $K1$: Spring constant of suspension system = 80,000 N/m
- $K2$: Spring constant of wheel and tire = 500,000 N/m
- $b1$: Damping constant of suspension system = 350 N.s/m
- $b2$: Damping constant of wheel and tire = 15,020 N.s/m
- UUU : Control force

Equations of Motion

From Newton's law, we derive the dynamic equations:

1. For the bus body mass:

$$M1\ddot{X}_1 = -b1(\dot{X}_1 - \dot{X}_2) - K1(X1 - X2) + U$$

2. For the suspension mass:

$$M2\ddot{X}_2 = b1(\dot{X}_1 - \dot{X}_2) + K1(X1 - X2) + b2(\dot{W} - \dot{X}_2) + K2(W - X2) - U$$

$$b_2 (\dot{W} - \dot{X}_2) + K_2 (W - X_2) - U M_2 \ddot{X}_2 = b_1 (\dot{X}_1 - \dot{X}_2) + K_1 (X_1 - X_2) + b_2 (\dot{W} - \dot{X}_2) + K_2 (W - X_2) - U$$

Transfer Function Models

By taking the Laplace Transform, we obtain the transfer functions $G_1(s)$ and $G_2(s)$:

$$\begin{aligned} [(M_1 s^2 + b_1 s + K_1) - (b_1 s + K_1) - (b_1 s + K_1)(M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2))][X_1(s) - X_2(s)] \\ = [U(s)(b_2 s + K_2)W(s) - U(s)] \left[\begin{array}{cc} (M_1 s^2 + b_1 s + K_1) & -(b_1 s + K_1) \\ -(b_1 s + K_1) & (M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2)) \end{array} \right] \begin{bmatrix} X_1(s) \\ X_2(s) \end{bmatrix} \\ = \left[\begin{array}{c} U(s) \\ (b_2 s + K_2)W(s) - U(s) \end{array} \right] \left[\begin{array}{cc} (M_1 s^2 + b_1 s + K_1) & -(b_1 s + K_1) \\ -(b_1 s + K_1) & (M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2)) \end{array} \right] \begin{bmatrix} X_1(s) \\ X_2(s) \end{bmatrix} \\ = [U(s)(b_2 s + K_2)W(s) - U(s)] \end{aligned}$$

The determinant Δ is:

$$\begin{aligned} \Delta &= (M_1 s^2 + b_1 s + K_1)(M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2)) - (b_1 s + K_1)^2 \\ \Delta &= (M_1 s^2 + b_1 s + K_1)(M_2 s^2 + (b_1 + b_2)s + (K_1 + K_2)) - (b_1 s + K_1)^2 \end{aligned}$$

Transfer Function for Control Input $U(s)$

Setting $W(s) = 0$:

$$\begin{aligned} G_1(s) = \frac{X_1(s)}{U(s)} = \frac{(M_1 + M_2)s^2 + b_2 s + K_2}{\Delta} \\ G_2(s) = \frac{X_2(s)}{U(s)} = \frac{(M_1 + M_2)s^2 + b_2 s + K_2}{\Delta} \end{aligned}$$

Summary of Key MATLAB Commands

- **tf('s')**: Defines the Laplace variable s .
- **step(G1)**: Plots the step response of the transfer function G_1 .

Inverted Pendulum: System Modeling

The inverted pendulum model explored in this tutorial is used to study and control a classic control system problem.

Key MATLAB Commands

- `tf`: Creates transfer functions.
- `ss`: Creates state-space models.
- `set`: Assigns names to inputs and outputs.

Problem Setup

- **System**: Inverted pendulum on a cart.
- **Objective**: Control the pendulum's angle and cart's position.
- **Design Requirements**:
 - Pendulum angle (θ) should settle within 5 seconds and not deviate more than 0.05 radians.
 - Cart position (x) should achieve its target within 5 seconds with a rise time under 0.5 seconds.

Equations of Motion

1. Horizontal motion of the cart: $M\ddot{x} + b\dot{x} + N = F$
2. Vertical motion of the pendulum:
 $(I + ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta$
 $mgl\sin\theta = -ml\ddot{x}\cos\theta$
3. Linearized system (small angle approximation):
 $(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x}$
 $(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x} + b\dot{x} - ml\ddot{\phi} = u$

Transfer Functions

1. Pendulum Angle:

$$\Phi(s)U(s) = mlqs^3 + b(l+ml^2)qs^2 - (M+m)mglqs - bmglq \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s^3 + \frac{b(l+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}}{s^3 + qb(l+ml^2)s^2 - q(M+m)mgl - qbmgl}$$

2. Cart Position:

$$X(s)U(s) = (l+ml^2)s^2 - gmlqs^4 + b(l+ml^2)qs^3 - (M+m)mglqs^2 - bmglqs \frac{X(s)}{U(s)} = \frac{\frac{(l+ml^2)s^2 - gml}{q}s^4 + \frac{b(l+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s}{s^4 + qb(l+ml^2)s^3 - q(M+m)mgl s^2 - qbmgl}$$

State-Space Representation

• Matrices:

- **A:** System matrix
- **B:** Input matrix
- **C:** Output matrix
- **D:** Feedthrough matrix

Related Links

- Intro to Modeling
- Pendulum Activity
- System Representation in MATLAB Video
- Modeling Intro Video

This summary covers the basics of setting up and analyzing an inverted pendulum system in MATLAB, including both transfer function and state-space representations.

Aircraft Pitch: System Modeling

This modelling focuses on the pitch control of an aircraft using both transfer function and state-space representations.

1. Physical Setup and System Equations

Aircraft Dynamics

- **Equations of Motion:** Governed by a set of nonlinear differential equations that describe the longitudinal dynamics of the aircraft.
- **Simplifications:**
 - Aircraft is in steady-cruise at constant altitude and velocity.

Equations:

- **Longitudinal Dynamics:** $\dot{\alpha} = -0.313\alpha + 56.7q + 0.232\delta$
 $\dot{q} = -0.0139\alpha - 0.426q + 0.0203\delta$
 $\dot{\theta} = 56.7q$

Here, α is the angle of attack, q is the pitch rate, θ is the pitch angle, and δ is the elevator deflection angle.

2. Transfer Function and State-Space Models

Transfer Function:

- The transfer function describes the relationship between the elevator deflection δ and the pitch angle θ .

Steps to derive:

- Take Laplace transforms of the equations.
- Solve the resulting algebraic equations to obtain:

State-Space Model:

- The state-space model provides a matrix representation of the system dynamics and is useful for control design and analysis.

State-Space Equations:

- **State Equation:**

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.313 & 56.70 & -0.0139 \\ -0.426 & 0.0567 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} \delta$$

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} \delta$$

$$\Delta \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} = \begin{bmatrix} -0.313 & 56.7 & -0.0139 \\ -0.426 & 0.0567 & 0 \end{bmatrix} \Delta \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} \Delta \delta$$

- **Output Equation:**

$$y = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix}$$

$$y = \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix}$$

3. Design Requirements

- **Overshoot:** Less than 10%
- **Rise Time:** Less than 2 seconds
- **Settling Time:** Less than 10 seconds
- **Steady-State Error:** Less than 2%

These requirements guide the design of a feedback controller to ensure desired performance in the pitch control system.

Additional Resources

- **System Representation in MATLAB:** [System Rep in MATLAB Video](#)
 - **Modeling Intro Video:** [Modeling Intro Video](#)
 - **System Conversions:** [Extras: System Conversions](#)
-

Ball & Beam: System Modeling

This model provides an overview of modeling a ball and beam system, including deriving transfer functions and state-space representations.

1. Physical Setup

- **System Description:** A ball rolls along a beam which is inclined by a servo-controlled lever. The ball's position is controlled by varying the angle of the beam, which is adjusted by the servo gear.
- **Components:**
 - **Ball:** Rolls along the beam.
 - **Beam:** Inclined by an angle α .
 - **Servo Gear:** Controls the angle α of the beam.

2. System Parameters

- **Mass of Ball (mmm):** 0.11 kg
- **Radius of Ball (RRR):** 0.015 m
- **Lever Arm Offset (ddd):** 0.03 m
- **Gravitational Acceleration (ggg):** 9.8 m/s²
- **Length of Beam (LLL):** 1.0 m
- **Moment of Inertia (JJJ):** 9.99e-6 kg·m²

3. Design Criteria

- **Settling Time:** < 3 seconds
- **Overshoot:** < 5%

4. System Equations

Linearized Equations

1. Equation of Motion:

$$(JR^2+m)r'' = -mgdL\theta \left(\frac{J}{R^2} + m \right) \ddot{r} = -m g \frac{d}{L} \theta$$

$$(R^2J+m)r'' = -mgLd\theta$$

Where:

- r'' is the acceleration of the ball.
- $\alpha = dL\theta$ relates the beam angle to the servo angle.

2. Transfer Function

Taking the Laplace transform of the linearized equation:

$$(JR^2+m)s^2R(s) = -mgdL\Theta(s) \left(\frac{J}{R^2} + m \right) s^2 R(s) = -m g \frac{d}{L} \Theta(s)$$

$$(R^2J+m)s^2R(s) = -mgLd\Theta(s)$$

Rearranging:

$$P(s) = R(s)\Theta(s) = -mgdL(JR^2+m) \frac{1}{s^2} P(s) = \frac{R(s)}{\Theta(s)} = - \frac{m g d}{L \left(\frac{J}{R^2} + m \right)}$$

$$\frac{1}{s^2} P(s) = \Theta(s) R(s) = -L(R^2J+m)mgds^2$$

. Additional Resources

- **System Representation in MATLAB:** [System Rep in MATLAB Video](#)
- **Modeling Intro Video:** [Modeling Intro Video](#)

Control Techniques Used:

1. Proportional-Integral-Derivative (PID) Control

- **Proportional (P):** Reacts to the current error, providing a correction proportional to the error magnitude.
- **Integral (I):** Reacts to the accumulated error over time, eliminating steady-state error.
- **Derivative (D):** Reacts to the rate of change of the error, predicting future error and improving system stability.

2. State-Space Control

- Represents the system with a set of first-order differential equations, capturing the system's dynamics in state-space form.
- Uses state feedback to control the system and stabilize it.

3. Model Predictive Control (MPC)

- Uses a model of the system to predict future behavior and optimize control inputs over a finite time horizon.
- Continuously updates control actions based on predictions and measurements.

4. Adaptive Control

- Adjusts control parameters automatically based on changing system dynamics or operating conditions.
- Can be used to handle systems with uncertain or varying parameters.

5. Robust Control

- Ensures system stability and performance in the presence of uncertainties and disturbances.
- **H_{∞} Control:** Minimizes the worst-case impact of disturbances on system performance.

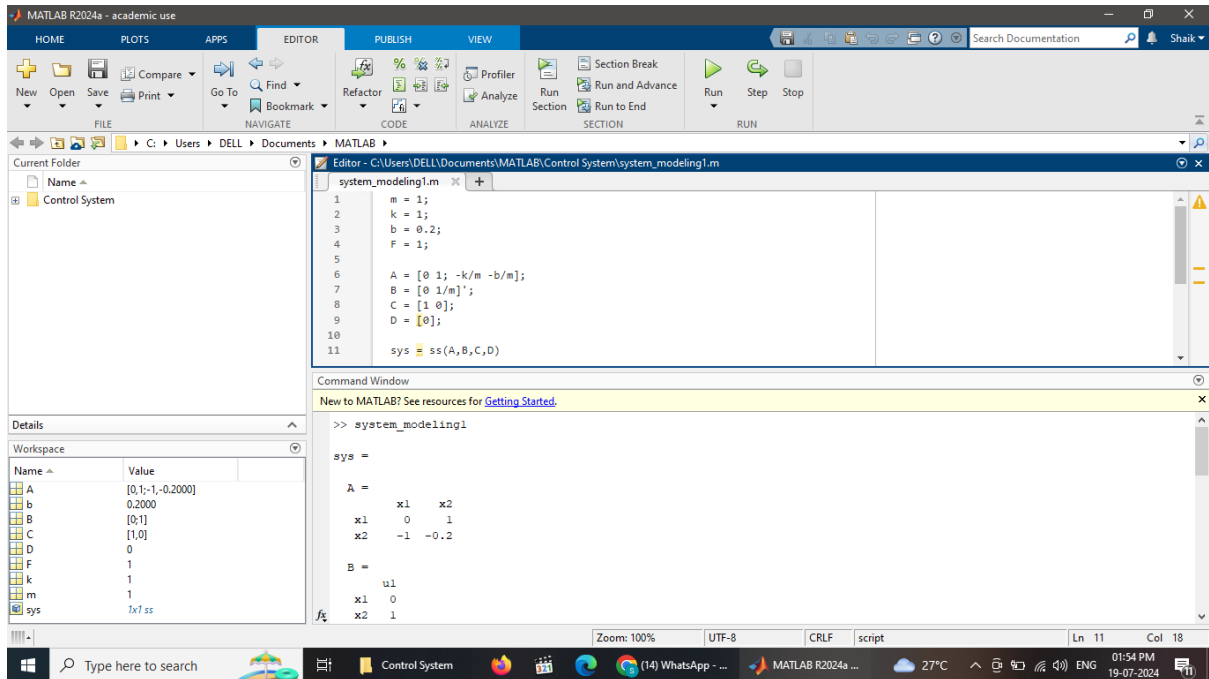
Implementation Process:

First, the system is analyzed to understand its dynamics and parameters. Next, the system is modeled using transfer function or state-space representations with tools like MATLAB or Python. The control algorithm is then designed based on system requirements, using techniques such as PID control or Model Predictive Control (MPC). The algorithm is implemented and tested in a simulation environment to verify performance. Comprehensive testing is conducted to ensure the system meets performance criteria like stability and accuracy. Tools such as MATLAB, Simulink, or Python libraries aid in modeling and analysis, while real-time processing may involve sensors and actuators.

Learnings:

- **System Modeling:** Understanding how to accurately model a system using transfer functions or state-space representations is crucial for designing effective control strategies.
- **Control Design:** Implementing various control techniques, such as PID, LQR, or MPC, highlights the importance of selecting the right approach based on system dynamics and performance requirements.
- **Simulation Testing:** Using simulation tools like MATLAB or Python to test control algorithms helps in identifying potential issues before hardware implementation, saving time and resources.
- **Hardware Integration:** Integrating control algorithms with hardware involves addressing real-world challenges such as sensor noise, actuator limits, and real-time processing constraints.

Outcomes:



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

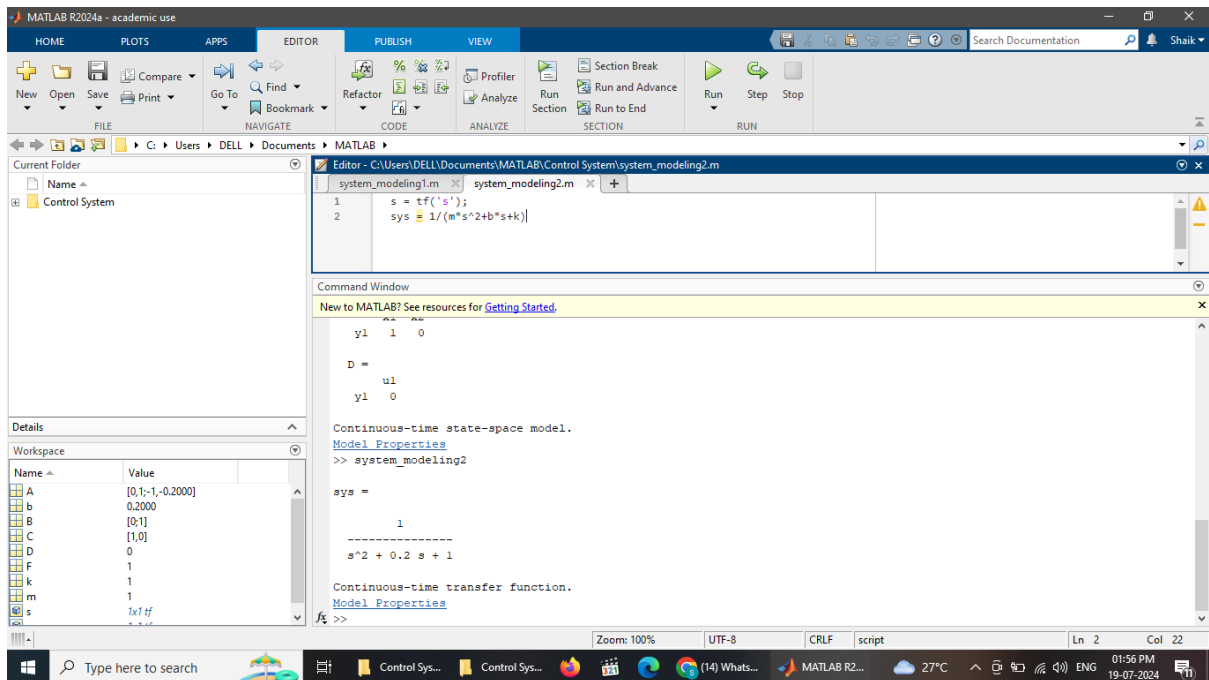
```
1 m = 1;  
2 k = 1;  
3 b = 0.2;  
4 F = 1;  
5  
6 A = [0 1; -k/m -b/m];  
7 B = [0 1/m]';  
8 C = [1 0];  
9 D = [0];  
10  
11 sys = ss(A,B,C,D)
```

The Command Window shows the execution of `>> system_modeling1` and the resulting system model:

```
sys =  
  
A =  
      x1      x2  
x1      0      1  
x2     -1     -0.2  
  
B =  
      u1  
x1      0  
x2      1  
  
C =  
      y1  
x1      1  
x2      0  
  
D =  
      y1  
x1      0  
x2      0
```

The Workspace window shows the following variables:

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



The screenshot shows the MATLAB R2024a interface. The Editor window displays the file `system_modeling2.m` with the following code:

```
1 s = tf('s');  
2 sys = 1/(m*s^2+b*s+k)
```

The Command Window shows the execution of `>> system_modeling2` and the resulting system model:

```
y1      1      0  
-----  
D =  
      u1  
y1      0  
  
Continuous-time state-space model.  
Model Properties  
>> system_modeling2  
  
sys =  
  
      1  
-----  
s^2 + 0.2 s + 1  
  
Continuous-time transfer function.  
Model Properties  
>>
```

The Workspace window shows the following variables:

Name	Value
A	[0,1;-1,-0.2000]
b	0.2000
B	[0;1]
C	[1,0]
D	0
F	1
k	1
m	1
s	1x1 tf

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Tools Window Help

Current Folder: C:\Users\DELL\Documents\MATLAB\Control System\motor_speed1.m

Workspace:

Name	Value
A	[0,1;-1,-0.2000]
b	0.1000
B	[0;1]
b1	350
b2	15020
C	[1,0]
D	0
den	[1,0.2000,1]
F	1

Editor: C:\Users\DELL\Documents\MATLAB\Control System\motor_speed1.m

```

1 J = 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.
f_s >>

```

Taskbar: Type here to search, Co..., Co..., Co..., Co..., Co..., Co..., 14..., M..., 27°C, ENG, 02:00 PM 19-07-2024

MATLAB R2024a - academic use

HOME PLOTS APPS EDITOR PUBLISH VIEW

File Edit View Tools Window Help

Current Folder: C:\Users\DELL\Documents\MATLAB\Control System\motor_speed2.m

Workspace:

Name	Value
A	[-10,1;-0.0200,-2]
b	0.1000
B	[0;2]
b1	350
b2	15020
C	[1,0]
D	0
den	[1,0.2000,1]
F	1

Editor: C:\Users\DELL\Documents\MATLAB\Control System\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 =

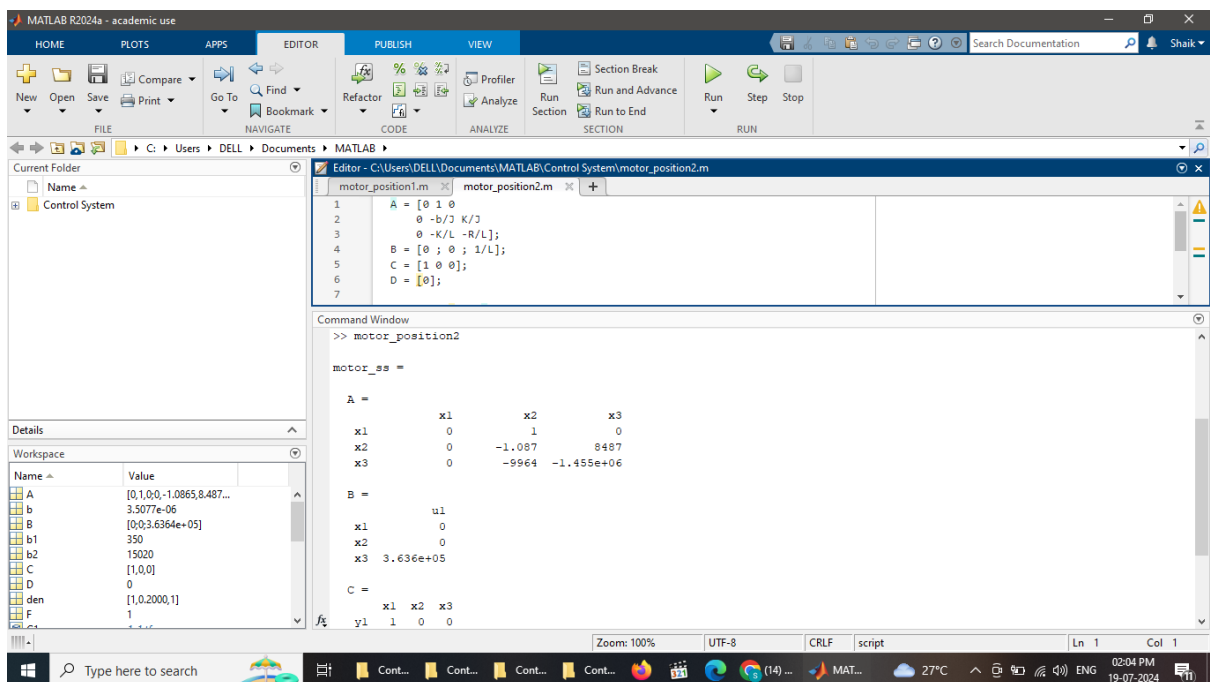
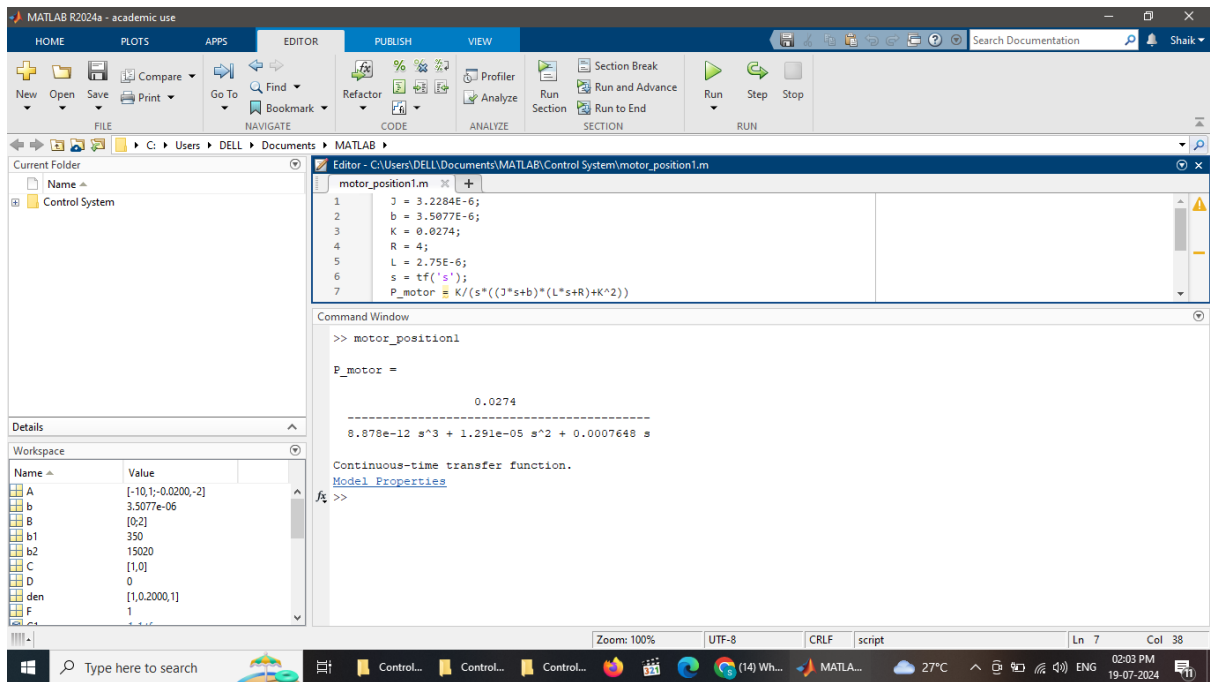
A =
      x1      x2
x1   -10         1
x2   -0.02      -2

B =
      u1
x1      0
x2      2

C =
      x1      x2
y1      1      0

```

Taskbar: Type here to search, Control Sys..., Control Sys..., 14) Whats..., MATLAB R2..., 27°C, ENG, 02:02 PM 19-07-2024



inverted_pendulum1.m

```

9      P_cart = (((I+m*I^2)/q)*s^2 - (m*g*I/q))/(s^4 + (b*(I + m*I^2))*s^3/q - ((M + m)*m*g*I)*s^2/q - b*m*g*I*s/q);
10
11      P_pend = (m*I*s/q)/(s^3 + (b*(I + m*I^2))*s^2/q - ((M + m)*m*g*I)*s/q - b*m*g*I/q);
12
13      sys_tf = [P_cart ; P_pend];
14
15      inputs = {'u'};
16      outputs = {'x'; 'phi'};
17

```

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.
Model Properties

```

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

```

Overall Results:

- The control systems exhibit desired performance metrics such as stability, accuracy, and responsiveness.
- Each system presents unique challenges related to dynamics, disturbances, and parameter tuning.

Unique Area Of Application:

- Cruise Control System
- Motor Speed Control
- Motor Position Control
- Suspension System:
- Inverted Pendulum
- Aircraft Pitch Control
- Ball and Beam System

Recommendations for Future Work:

1. **Enhanced Control Algorithms:** Investigate advanced control techniques like Model Predictive Control (MPC) or Adaptive Control to improve system performance, particularly in environments with varying conditions or disturbances.
2. **Integration with AI and Machine Learning:** Explore the application of machine learning algorithms for adaptive control, fault detection, and system optimization to enhance performance and robustness.
3. **Hardware-In-the-Loop (HIL) Simulation:** Implement HIL simulation to test and validate control algorithms in a real-time environment before deployment in actual hardware systems.

Reference:

<https://ctms.engin.umich.edu/CTMS/index.php?aux=Home>