Topic: Exploring 3-DOF Robotic Arm controls and design (Team 11)

Member: KengYu Huang, LiangChih Fu, Ezhilan Veluchami, Shiva Sam Kumar Govindan

- **Introduction:**

This project addresses the challenge of precise and stable control in a 3-DOF robotic arm [1] (Appendix fig1.), a common issue in robotics that requires sophisticated control mechanisms to ensure accuracy and efficiency. The problem originates from the complex nature of robotic systems, where precision and stability are essential for diverse tasks, from industrial automation to delicate surgeries [2]. To tackle this, the project explores the integration of advanced control theories, specifically state feedback control, into the kinematics, dynamics, and state-space analysis of the robotic arm. The project aims to design and implement an optimal state feedback controller, potentially with Observer-Based Compensators, to improve the robotic arm's dynamic response, focusing on reducing overshoot and achieving faster settling times.

- **Modeling 3-DOF Robotic ARM:**

Kinematics and Dynamics:

In robotic kinematics, Forward Kinematics is used to determine a robot's joint positions, enabling the calculation of joint angles and link points as outlined in Appendix eq1. Jacobian matrices, particularly JPl and JOl for links and JPM and JOM for motors (Appendix eq2.), are crucial for precision in movement, as they establish the relationship between joint velocities

and workspace velocities.[3]

For the dynamics aspect (Appendix eq3.), key matrices like the Inertia Matrix (B) (Appendix eq4.), combining link and motor inertias, the Coriolis Matrix (C) (Appendix eq5.), representing Coriolis and centrifugal forces, and the Gravity Term (g) (Appendix eq6.), for gravitational forces, are essential. The verification of the skew-symmetry of matrix N (B−2C) is a vital step in confirming the dynamic model's accuracy and its compliance with physical laws, critical for effective control system design and analysis.[3]

Linearization of Equation of Motion and State Space Model:

To linearize the equation of motion for state-space modeling, the first step (Appendix eq7.) involves inverting the inertia matrix B. This inversion is critical as the equations of motion are typically represented in a form where the inertia matrix is a key factor. The angular accelerations ($\ddot{\theta}$) (Appendix eq8.) are computed by incorporating the effects of input torques (τ), Coriolis forces ($C\dot{\theta}$), and gravitational forces (g), according to the rearranged equation of motion. The state vector (x) (Appendix eq9. and 10.) includes joint angles (θ) and their velocities ($\ddot{\theta}$).

In the state-space model, the state matrix (A) (Appendix eq11.) describes the evolution of the state vector over time, while the input matrix (B) (Appendix eq12.) illustrates how inputs (torques) affect state changes. The output matrix (C) (Appendix eq13.) extracts the desired output from the state vector. By inserting specific parameters (Appendix eq14.) into these

matrices, one can obtain a state-space form (Appendix eq15.) that effectively represents the system's dynamics, essential for designing control systems and understanding system behavior under various conditions.

- **Analysis of properties**

The Controllability Matrix P (Appendix eq16.) is full rank, indicating all states are controllable. Similarly, the Observability matrix Q (Appendix eq17.) is also full rank, ensuring all states are observable. In stability analysis, eigenvalues of matrix A (Appendix eq18.) are purely imaginary and in conjugate pairs, indicating sustained oscillations without natural return to equilibrium or divergence. This classifies the system as Marginally Stable.

Controller design:

1. **State feedback control**

The State feedback control system can be described by the equation: $u=-Kx+Gr$

2. **Observer Based Compensator:**

Observer is defined by $\dot{\hat{x}} = A\,\hat{x} + B\,u + L\,(x - C\,\hat{x})$

where controller equations defined by u = -K$\hat{x}$+ G r

Augmented system for the Controller with both x and $\hat{x}$ as represented in Appendix eq20.

- u is the control input (torque).
- K is the state feedback gain matrix.
- x is the state vector (joint angles and velocities).

- $\hat{x}$ is the estimated state estimated state obtained from the observer.
- *G* is the input scaling matrix, derived from the inverse of the DC gain.
- *r* is the reference input, the desired state we want the system to achieve.

Objective

The objective of this section is to design a state feedback controller for our system. The design process aims to determine the optimal set of feedback gains that result in desired dynamic characteristics, specifically targeting overshoot and settling time performance criteria.

Methodology

In this controller design, a state feedback system aims to optimize overshoot and settling time. The process starts by initializing state-space matrices and setting ranges for parameters alpha_1, alpha_2, and alpha_3. A thorough search identifies the optimal alpha combination by evaluating the state feedback matrix K and the system's response for each alpha set. The best alphas are chosen based on the highest count of input-output pairs meeting performance criteria and the lowest average overshoot and settling time. This iterative approach ensures the controller meets specific dynamic requirements for minimal overshoot and optimal settling times.

The Observer is Designed to meet the same specifications and the desired pole of the observer is picked arbitrarily by scaling the optimal poles of the State Feedback controller to meet specification.

- **Results**

The optimal pole locations for the state feedback controller are in Appendix eq19. This optimal combination's effectiveness is confirmed by analyzing the closed-loop system's step response and checking the overshoot and settling time for each input-output pair, as detailed in Appendices fig2 and fig3, ensuring the controller meets the desired performance criteria. We empirically determined the poles by scaling the State Feedback controller poles by a factor of 5 (Refer Appendix fig4.).

Plot Interpretation:

The step response plots depict the angular positions (θ1, θ2, θ3) of the system in response to unit step inputs applied to their respective joints. Key observations form Fig5. are as follows.

1. **Stability**: Both controllers demonstrate system stability. There are no sustained oscillations or divergences in any response. The smooth convergence to steady-state values signifies well-damped system behavior.
2. **Dynamic Response:**

   Overshoot: Remarkably, both controllers exhibit minimal to zero overshoot in most cases. In one instance, a slight overshoot is observed but remains within acceptable limits.

   Settling Time: The State Feedback controller achieves rapid settling times, all comfortably below the 3-second target. The longest settling time is approximately 1.88 seconds. Conversely, the Observer-Based compensator shows significantly longer settling times,

roughly 1.5 times that of the State Feedback controller.

3. **Steady-State Tracking**: The steady-state of I/O pair 1, 2, and 3 are 1, the others are 0. It is a good phenomenon because each motor will only affect its corresponding output.

- **Conclusions and discussions**

In this this project focused on the design and application of a state feedback controller to enhance the performance of a 3-DOF robotic arm. By integrating aspects of kinematics, dynamics, and state-space analysis, the controller, employing optimal pole placement, demonstrated remarkable improvements in the system's dynamic response. It achieved minimal overshoot and rapid settling times during simulations, showcasing its robust and efficient control capabilities. Notably, when compared to the controller with an Observer-based compensator, the State Feedback Controller emerged as the superior choice, particularly in terms of settling time. This preference can be attributed to its simplicity, reduced sensitivity to model errors, robustness, reduced computational overhead, and ease of implementation. In addition to these advantages, the State Feedback Controller's direct utilization of available state information ensures reliable and precise control, making it an excellent choice for systems where all states are readily accessible. As future work, the testing of the controller's resilience against disturbances will further validate its practical applicability in real-world scenarios.

- **Contributions of team members:** Each team member contribute equally!

Reference:

1. Singla, A. (2007, February). Redundancy control of robot manipulators using task priority -researchgate. https://www.researchgate.net/publication/281374317_Redundancy_control_of_robot_manipulators_using_task_priority
2. Mahil, S. M., & Al-Durra, A. (2016). Modeling analysis and simulation of 2-DOF robotic manipulator. 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS). https://doi.org/10.1109/mwscas.2016.7870099
3. Siciliano, B. (2009). Robotics: Modelling, planning and control. Springer.

Appendix:

Eq1.

$$x_1 = L_1 \sin(\theta_1)$$

$$y_1 = L_1 \cos(\theta_1)$$

$$x_2 = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$$

$$y_2 = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$

$$x_3 = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

$$y_3 = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

Eq2.

Jacobian Matrices for Links (*JPl*)

$$J_{Pl}(:,:,1) = \begin{bmatrix} -l_1 \sin(\theta_1) & 0 & 0 \\ l_1 \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$J_{Pl}(:,:,2) = \begin{bmatrix} -a_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) & 0 \\ a_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$J_{Pl}(:,:,3)$

$$= \begin{bmatrix} -a_1 \sin(\theta_1) - a_2 \sin(\theta_1 + \theta_2) - l_3 \sin(\theta_1 + \theta_2 + \theta_3) & -a_2 \sin(\theta_1 + \theta_2) - l_3 \sin(\theta_1 + \theta_2 + \theta_3) & -l_3 \\ a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) & a_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) & l_3 \, c \\ 0 & 0 & \end{bmatrix}$$

Jacobian Matrices for Motors (*JPM*)

$$J_{PM}(:,:,1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$J_{PM}(:,:,2) = \begin{bmatrix} -a_1 \sin(\theta_1) & 0 & 0 \\ a_1 \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$J_{PM}(:,:,3) = \begin{bmatrix} -a_1\sin(\theta_1) - a_2\sin(\theta_1+\theta_2) & -a_1\sin(\theta_1) & 0 \\ a_1\cos(\theta_1) + a_2\cos(\theta_1+\theta_2) & a_1\cos(\theta_1) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Jacobian Matrices for motor for Links (*J0l*)

$$J_{0l}(:,:,1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$J_{0l}(:,:,2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

$$J_{0l}(:,:,3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Jacobian Matrices for Rotational Motion for Motors (*J0M*)

$$J_{0M}(:,:,1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ k_{r1} & 0 & 0 \end{bmatrix}$$

$$J_{0M}(:,:,2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & k_{r2} & 0 \end{bmatrix}$$

$$J_{0M}(:,:,3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & k_{r3} \end{bmatrix}$$

Eq3.

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = F$$

Eq4.

$$B(q) = \sum_{i=1}^{3}\Big[m_{li}(i)\cdot\left(J_{Pl}(:,:,i)^T\cdot J_{Pl}(:,:,i)\right) + J_{0l}(:,:,i)^T\cdot\left(R_{s_i}\cdot I_l(i)\cdot R_{s_i}^T\right)\cdot J_{0l}(:,:,i)$$

$$+ m_{mi}(i)\cdot\left(J_{PM}(:,:,i)^T\cdot J_{PM}(:,:,i)\right) + J_{0M}(:,:,i)^T$$

$$\cdot\left(R_{s_{i+3}}\cdot I_m(i)\cdot R_{s_{i+3}}^T\right)\cdot J_{0M}(:,:,i)\Big]$$

Eq5.

$$C_{ij}(q,\dot{q}) = \sum_{k=1}^{3} \frac{1}{2}\left(\frac{\partial B_{ij}}{\partial \theta_k} + \frac{\partial B_{ik}}{\partial \theta_j} - \frac{\partial B_{jk}}{\partial \theta_i}\right) \cdot \dot{\theta_k}$$

Eq6.

$$g(q) = -\sum_{i=1}^{3} \left[m_{li}(i) \cdot \left(g_0^T \cdot J_{Pl}(:,:,i)\right)^T + m_{mi}(i) \cdot \left(g_0^T \cdot J_{PM}(:,:,i)\right)^T \right]$$

Eq7.

$$B\ddot{\theta} = \tau - C\dot{\theta} - g$$

Eq8.

$$\ddot{\theta} = B^{-1}(\tau - C\dot{\theta} - g)$$

Eq9.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \qquad \dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix}$$

Eq10.

$$f(x,u) = \begin{bmatrix} \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = B^{-1}(\tau - C \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix} - g)$$

Eq11.

$$A = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\partial \dot{x}_4}{\partial \mathrm{x}_1} & \frac{\partial \dot{x}_4}{\partial \mathrm{x}_2} & \frac{\partial \dot{x}_4}{\partial x_3} & \frac{\partial \dot{x}_4}{\partial x_4} & \frac{\partial \dot{x}_4}{\partial x_5} & \frac{\partial \dot{x}_4}{\partial x_6} \\ \frac{\partial \dot{x}_5}{\partial x_1} & \frac{\partial \dot{x}_5}{\partial x_2} & \frac{\partial \dot{x}_5}{\partial x_3} & \frac{\partial \dot{x}_5}{\partial x_4} & \frac{\partial \dot{x}_5}{\partial x_5} & \frac{\partial \dot{x}_5}{\partial x_6} \\ \frac{\partial \dot{x}_6}{\partial x_1} & \frac{\partial \dot{x}_6}{\partial x_2} & \frac{\partial \dot{x}_6}{\partial x_3} & \frac{\partial \dot{x}_6}{\partial x_4} & \frac{\partial \dot{x}_6}{\partial x_5} & \frac{\partial \dot{x}_6}{\partial x_6} \end{matrix}$$

Eq12.

$$B = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{\partial \dot{x}_4}{\partial \tau_1} & \frac{\partial \dot{x}_4}{\partial \tau_2} & \frac{\partial \dot{x}_4}{\partial \tau_3} \\ \frac{\partial \dot{x}_5}{\partial \tau_1} & \frac{\partial \dot{x}_5}{\partial \tau_2} & \frac{\partial \dot{x}_5}{\partial \tau_3} \\ \frac{\partial \dot{x}_6}{\partial \tau_1} & \frac{\partial \dot{x}_6}{\partial \tau_2} & \frac{\partial \dot{x}_6}{\partial \tau_3} \end{matrix}$$

Eq13.

$$C = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$

Eq14.

$\theta_1, \theta_2, \theta_3$: Initial pose = $\left[-\frac{\pi}{2}, \frac{\pi}{2}, 0\right]$

m_{l_1} , m_{l_2} , m_{l_3} : The masses of the 3 links = 0.5(kg)

$m_{m_1}, m_{m_2}, m_{m_3}$: The masses of the rotors of the joint motors = 0.5(kg)

$I_{l_1}, I_{l_2}, I_{l_3}$: The moments of inertia relative to the centers of mass of the two links:

$$(m_l \ * a^2) \ / \ 12$$

a1,a2,a3: Link length:0.5(m)

l_1, l_2, l_3: The distances of the centers of mass to respective joint = Link length / 2

$k_{r_1}, k_{r_2}, k_{r_3}$: gear ratios = 100

I_{m1}, I_{m2}, I_{m3}: The moments of inertia with respect to the axes of the two rotors

I_m = k * mm * r^2 (where r is the motor shaft radius and k is a constant)

For simplicity, let's assume k = 1 and r = 0.01m

τ_1, τ_2, τ_3: forces applied to the two joints(assume no torque apply)

Eq15.

$$A = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -7.77942 & -2.69827 & -0.467624 & 0 & 0 & 0 \\ 1.04786 & -0.225818 & 0.0141864 & 0 & 0 & 0 \\ 1.35667 & 0.142804 & -0.355128 & 0 & 0 & 0 \end{matrix}$$

$$B = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.787374 & -0.371144 & -0.0838863 \\ -0.371144 & 1.24289 & -0.17569 \\ -0.0838863 & -0.17569 & 1.89847 \end{matrix}$$

Eq16.

$$P = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B]$$

$$= \begin{matrix} 0 & 0 & 0 & 0.787 & -0.371 & -0.0839 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.371 & 1.24 & -0.176 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.839 & -0.176 & 1.898 & 0 & 0 & 0 \\ 0.787 & -0.371 & -0.0839 & 0 & 0 & 0 & -5.085 & -0.384 & 0.2389 \\ -0.371 & 1.24 & -0.176 & 0 & 0 & 0 & 0.9077 & -0.672 & -0.0213 \\ -0.839 & -0.176 & 1.898 & 0 & 0 & 0 & 1.045 & -0.264 & -0.813 \end{matrix}$$

Eq17.

$$\text{Observability matrix Q} = \begin{bmatrix} C \\ CA \\ CA^2 \\ . \\ . \\ . \\ CA^n - 1 \end{bmatrix} = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -7.779 & -2.698 & -0.4676 & 0 & 0 & 0 \\ 1.048 & -0.2258 & 0.01419 & 0 & 0 & 0 \\ 1.357 & 0.1428 & -0.3551 & 0 & 0 & 0 \end{matrix}$$

Eq18.

(1.) 0.0000 +2.7000i

(2.) 0.0000 +0.8395i

(3.) 0.0000 -0.6045i

(4.) 0.0000 -2.7000i

(5.) 0.0000 -0.8395i

(6.) 0.0000 +0.6045i

Eq19.

5 +2.7000i

7.5 +0.8395i

7.3 -0.6045i

5 -2.7000i

7.5 -0.8395i

7.3 +0.6045i

Eq20.

$$\begin{bmatrix} \dot{x} \\ \dot{\hat{x}} \end{bmatrix} = \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \end{bmatrix} + \begin{bmatrix} BG \\ BG \end{bmatrix} r$$

Fig 1.

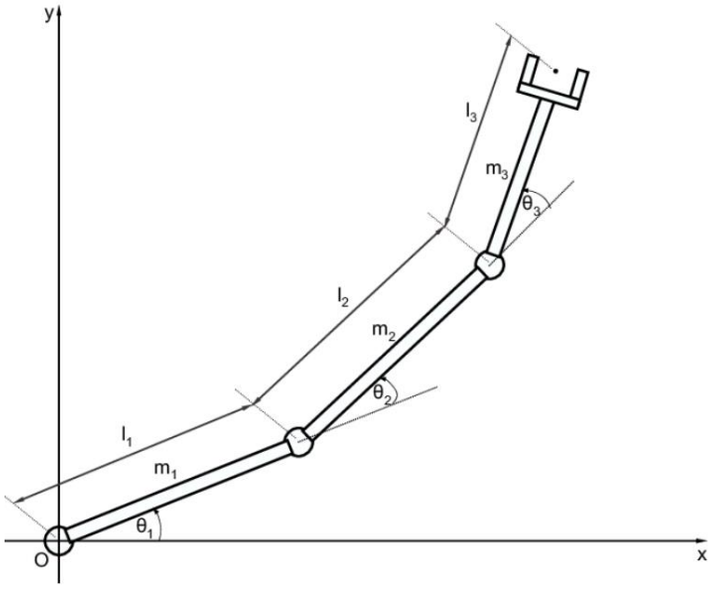


Fig 2.

```
Input 1 - Output 1: Overshoot = 0.000000%, Settling Time = 0.770050 s
Input 1 - Output 2: Overshoot = 0.000000%, Settling Time = 1.476550 s
Input 1 - Output 3: Overshoot = 0.000000%, Settling Time = 1.877892 s
Input 2 - Output 1: Overshoot = 0.000000%, Settling Time = 1.765974 s
Input 2 - Output 2: Overshoot = 0.002247%, Settling Time = 0.757795 s
Input 2 - Output 3: Overshoot = 0.000000%, Settling Time = 1.352534 s
Input 3 - Output 1: Overshoot = 0.000000%, Settling Time = 1.358327 s
Input 3 - Output 2: Overshoot = 0.000000%, Settling Time = 1.878043 s
Input 3 - Output 3: Overshoot = 0.000000%, Settling Time = 0.757786 s
```

Fig 3.

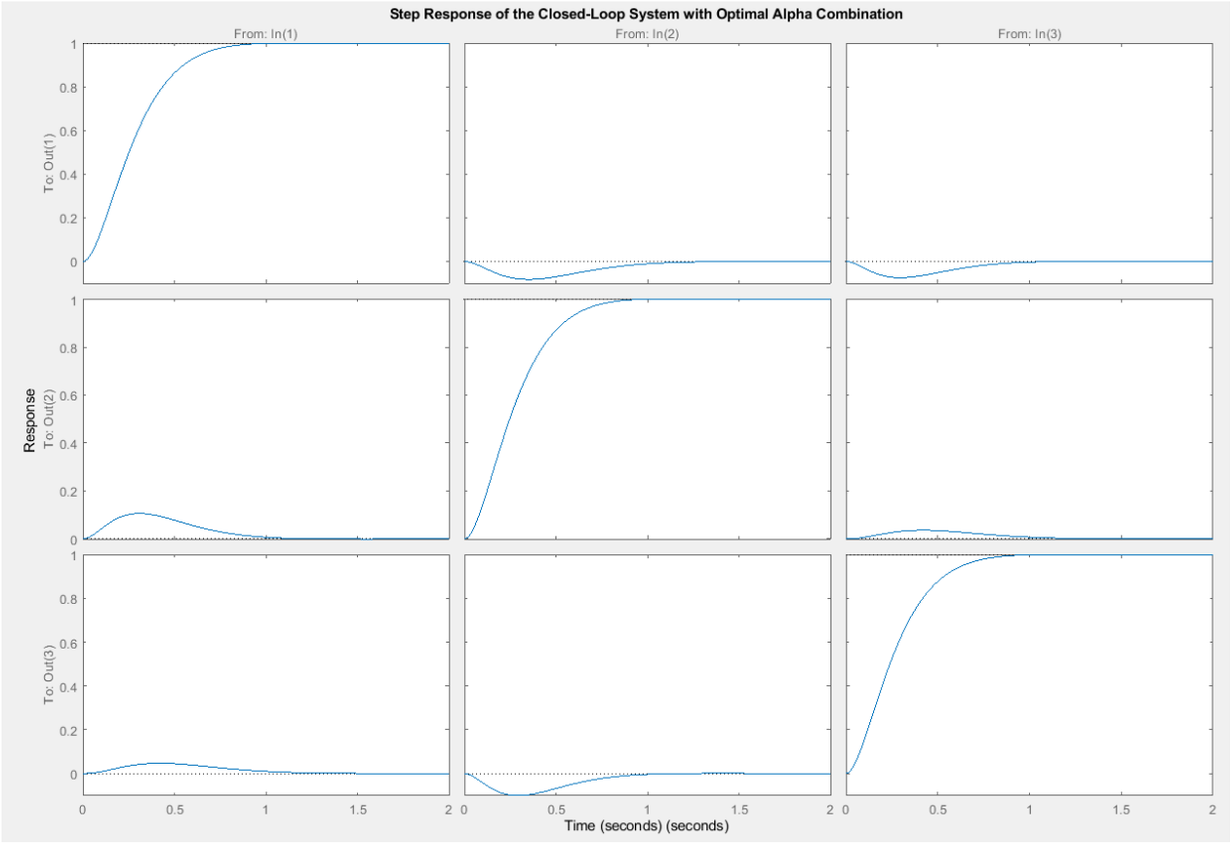

Step Response of the Closed-Loop System with Optimal Alpha Combination
From: In(1)
From: In(2)
From: In(3)
To: Out(1)
To: Out(2)
To: Out(3)
Response
Time (seconds) (seconds)

Fig 4.

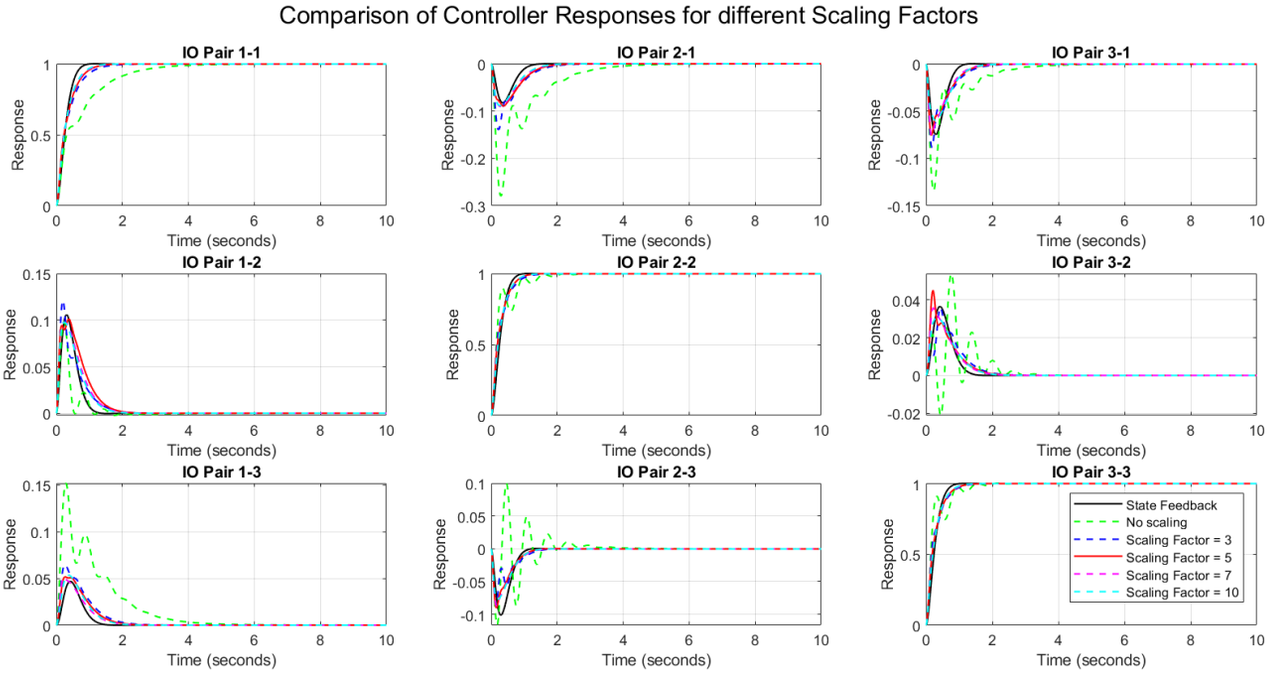
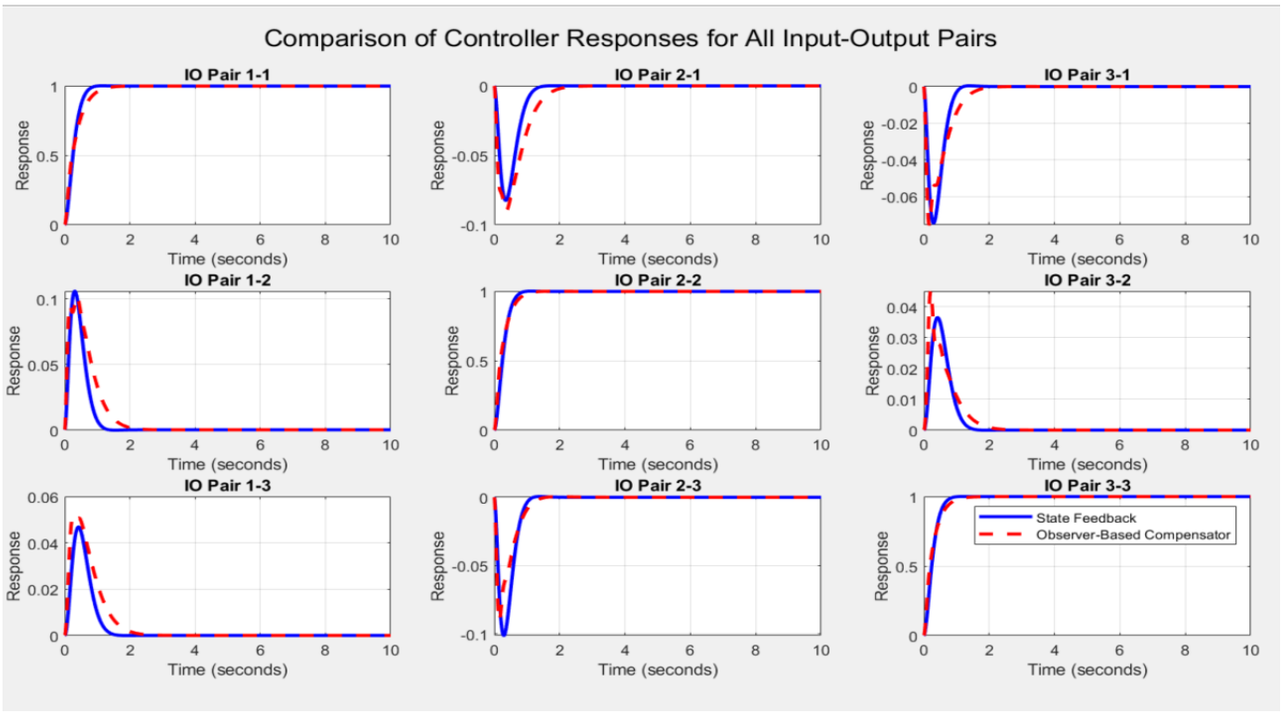

Comparison of Controller Responses for different Scaling Factors
IO Pair 1-1
IO Pair 2-1
IO Pair 3-1
IO Pair 1-2
IO Pair 2-2
IO Pair 3-2
IO Pair 1-3
IO Pair 2-3
IO Pair 3-3
Response
Time (seconds)
State Feedback
No scaling
Scaling Factor = 3
Scaling Factor = 5
Scaling Factor = 7
Scaling Factor = 10

Fig 5.


Comparison of Controller Responses for All Input-Output Pairs
IO Pair 1-1
IO Pair 2-1
IO Pair 3-1
IO Pair 1-2
IO Pair 2-2
IO Pair 3-2
IO Pair 1-3
IO Pair 2-3
IO Pair 3-3
Response
Time (seconds)
State Feedback
Observer-Based Compensator

Matlab code:

Contents


```
% Initializing symbolic toolbox
syms a1 a2 l1 l2 l3 theta1 theta2 theta3 kr1 kr2 kr3 real
syms ml1 ml2 ml3 mm1 mm2 mm3 Il1 Il2 Il3 Im1 Im2 Im3 real

% Cosine and sine functions for joint angles
c1 = cos(theta1);
c12 = cos(theta1 + theta2);
c123 = cos(theta1 + theta2 + theta3);

s1 = sin(theta1);
s12 = sin(theta1 + theta2);
s123 = sin(theta1 + theta2 + theta3);

% Define the link and motor points
P1 = [l1*c1 ; l2*s1 ; 0];
P2 = [a1*c1 + l2*c12 ; a1*s1 + l2*s12 ; 0];
P3 = [a1*c1 + a2*c12 + l3*c123 ; a1*s1 + a2*s12 + l3*s123 ; 0];

% Preallocate Jacobians for link points
JPl = sym(zeros(3,3,3));
J0l = sym(zeros(3,3,3));

JPl(:,:,1) = [-l1*s1, 0, 0
               l1*c1, 0, 0;
               0, 0, 0];
JPl(:,:,2) = [(-a1*s1 - l2*s12), (-l2*s12), 0;
               (a1*c1 + l2*c12), (l2*c12), 0;
               0, 0, 0];
JPl(:,:,3) = [(-a1*s1 - a2*s12 - l3*s123), (-a2*s12 - l3*s123), (-l3*s123);
              (a1*c1 + a2*c12 + l3*c123), (a2*c12 + l3*c123), (+l3*c123);
              0, 0, 0];

J0l(:,:,1) = [0 0 0; 0 0 0; 1 0 0];
J0l(:,:,2) = [0 0 0; 0 0 0; 1 1 0];
J0l(:,:,3) = [0 0 0; 0 0 0; 1 1 1];

% Preallocate Jacobians for Motor
JPM = sym(zeros(3,3,3));
J0M = sym(zeros(3,3,3));

JPM(:,:,1) = [0 0 0; 0 0 0; 0 0 0];
JPM(:,:,2) = [-a1*s1, 0, 0;
               a1*c1, 0, 0;
               0 0 0];
JPM(:,:,3) = [(-a1*s1 - a2*s12), (-a1*s1), 0;
               (a1*c1 + a2*c12), (a1*c1), 0;
               0, 0, 0];

J0M(:,:,1) = [0 0 0;
              0 0 0;
              kr1 0 0];
J0M(:,:,2) = [0 0 0;
              0 0 0;
              1 kr2 0];
J0M(:,:,3) = [0 0 0;
              0 0 0;
              1 1 kr3];

% Given rotation matrices for link and motor
R1 = [0; 0; 1];
R2 = [0; 0; 1];
R3 = [0; 0; 1];
Rm1 = [0; 0; 1];
Rm2 = [0; 0; 1];
Rm3 = [0; 0; 1];
Rs = {R1, R2, R3, Rm1, Rm2, Rm3};  % Consolidated rotation matrices

% Given masses and moments of inertia
mli = [ml1, ml2, ml3];
mmi = [mm1, mm2, mm3];
Il = [Il1, Il2, Il3];
Im = [Im1, Im2, Im3];
```

```
% Calculate the inertia matrix B
B = sym(zeros(3,3));
for i = 1:3
    B = B + mli(i) * (JPl(:,:,i)' * JPl(:,:,i)) ...
        + J0l(:,:,i)' * (Rs{i} * Il(i) * Rs{i}') * J0l(:,:,i) ...
        + mmi(i) * (JPM(:,:,i)' * JPM(:,:,i)) ...
        + J0M(:,:,i)' * (Rs{i+3} * Im(i) * Rs{i+3}') * J0M(:,:,i);
end


% Simplify each element of the inertia matrix B
B_simplified = simplify(B, 'Steps', 100);

% Display simplified B
disp('Simplified B:');
disp(B_simplified);
```

```
Simplified B:
[Il1 + Il2 + Il3 + Im2 + Im3 + Im1*kr1^2 + a1^2*ml2 + a1^2*ml3 + a2^2*ml3 + a1^2*mm2 + a1^2*mm3 + a2^2*mm3 + l1^2*ml1 + l2^2*ml2 + l3^2*ml3 + 2*a1*l3*ml3*cos(t
[                                                                              Il2 + Il3 + Im3 + Im2*kr2 + a2^2*ml3 + a1^2*mm3 + l2^2*ml2 + l3^2*ml3 + a1*l3*ml3
[
```

Compute C Matric

```
syms dtheta1 dtheta2 dtheta3 real

C = sym(zeros(3,3));
for i = 1:3
    for j = 1:3
        C(i,j) = 0;
        for k = 1:3
            Christoffel = 0.5 * (diff(B_simplified(i, j), ['theta', num2str(k)]) + ...
                                 diff(B_simplified(i, k), ['theta', num2str(j)]) - ...
                                 diff(B_simplified(j, k), ['theta', num2str(i)]));
            C(i,j) = C(i,j) + Christoffel * ['dtheta', num2str(k)];
        end
    end
end
C_simplified = simplify(C, 'Steps', 100);
display(C_simplified)

B_dot = diff(B_simplified, theta1) * dtheta1 + diff(B_simplified, theta2) * dtheta2 + diff(B_simplified, theta3) * dtheta3;
B_dot_simplified = simplify(B_dot, 'Steps', 100);

display(B_dot_simplified)
```

```
C_simplified =

[- a1*dtheta2*(a2*ml3*sin(theta2) + a2*mm3*sin(theta2) + l2*ml2*sin(theta2) + l3*ml3*sin(theta2 + theta3)) - dtheta3*l3*ml3*(a1*sin(theta2 + theta3) + a2*sin(t
[                   dtheta1*(a1*l3*ml3*sin(theta2 + theta3) + a1*a2*ml3*sin(theta2) + a1*a2*mm3*sin(theta2) + a1*l2*ml2*sin(theta2)) - a2*dtheta3*l3*ml3*sin
[                                                                       dtheta1*(a1*l3*ml3*sin(theta2 + theta3) + a2*l3*ml3*sin(theta3)) + a2*dtheta2*l3*ml3*sin


B_dot_simplified =

[- 2*a1*dtheta2*(a2*ml3*sin(theta2) + a2*mm3*sin(theta2) + l2*ml2*sin(theta2) + l3*ml3*sin(theta2 + theta3)) - 2*dtheta3*l3*ml3*(a1*sin(theta2 + theta3) + a2*s
[  - a1*dtheta2*(a2*ml3*sin(theta2) + a2*mm3*sin(theta2) + l2*ml2*sin(theta2) + l3*ml3*sin(theta2 + theta3)) - dtheta3*l3*ml3*(a1*sin(theta2 + theta3) + 2*a2*s
[                                                              - dtheta3*(a1*l3*ml3*sin(theta2 + theta3) + a2*l3*ml3*sin(theta3)) - a1*dtheta2*l3*ml3*sin(theta
```

Check N is skew-symmetric or not

```
% Calculate N
N = B_dot_simplified - 2 * C_simplified;
N_simplified = simplify(N, 'Steps', 100);
disp('N_simplified:');
disp(N_simplified);

% Check if N is skew-symmetric
isSkewSymmetric = simplify(N_simplified + transpose(N_simplified)) == zeros(3);
disp('Is N skew-symmetric?');

if all(isSkewSymmetric(:))
    disp('Yes');
else
    disp('No');
end
```

```
N_simplified:
[
[-a1*(2*dtheta1*l3*ml3*sin(theta2 + theta3) + dtheta2*l3*ml3*sin(theta2 + theta3) + dtheta3*l3*ml3*sin(theta2 + theta3) + 2*a2*dtheta1*ml3*sin(theta2) + a2*dth
[                                                                                                         -l3*ml3*(2*a2*dtheta1*sin(theta3) + 2*a2*dtheta2*
```

```
Is N skew-symmetric?
Yes
```

Gravity Term

```
% Gravity vector in base frame (assuming it's acting in the negative y-direction)
g0 = [0; -9.81; 0];

% Calculate gravity term g
g = sym(zeros(3,1));
for i = 1:3
    g = g - (mli(i) * (g0.' * JPl(:,:,i)).' + mmi(i) * (g0.' * JPM(:,:,i)).');
end

% Simplify gravity term g
g_simplified = simplify(g, 'Steps', 100);

% Display simplified g
disp('Simplified g:');
disp(g_simplified);
```

```
Simplified g:
ml3*((981*a2*cos(theta1 + theta2))/100 + (981*a1*cos(theta1))/100 + (981*l3*cos(theta1 + theta2 + theta3))/100) + mm3*((981*a2*cos(theta1 + theta2))/100 + (981
                                                                                                                                                        ml3
```

State-Space Form

```
syms theta1 theta2 theta3 dtheta1 dtheta2 dtheta3 ddtheta1 ddtheta2 ddtheta3 tau1 tau2 tau3 real
% Define state and input vectors
x = [theta1; theta2; theta3; dtheta1; dtheta2; dtheta3];
u = [tau1; tau2; tau3];

% Define the equations of motion
B_inv = inv(B_simplified);
C_term = C_simplified * [dtheta1; dtheta2; dtheta3];
g_term = g_simplified;
ddtheta = B_inv * (u - C_term - g_term);

% Define the A_hat matrix
A_hat_top = [zeros(3), eye(3)];
A_hat_bottom = jacobian(ddtheta, x);  % 3x6 matrix of partial derivatives
A_hat = [A_hat_top; A_hat_bottom];

% Define the B_hat matrix
B_hat_bottom = jacobian(ddtheta, u);  % 3x3 matrix of partial derivatives with respect to inputs
B_hat = [zeros(3,3); B_hat_bottom];

% Define the C_hat matrix
C_hat = [eye(3), zeros(3,3)];

% Display the results
disp('A_hat =');
disp(A_hat);

disp('B_hat =');
disp(B_hat);

disp('C_hat =');
disp(C_hat);
```

```
A_hat =
[
[
[
[
[
[((ml3*((981*a2*sin(theta1 + theta2))/100 + (981*a1*sin(theta1))/100 + (981*l3*sin(theta1 + theta2 + theta3))/100) + mm3*((981*a2*sin(theta1 + theta2))/100 +

B_hat =
[
[
[
[
[                                  -(Il2*Il3 + Il3*Im3 + Il3*Im2*kr2 - 2*Il3*Im3*kr3 + a2^2*l3^2*ml3^2 + Il2*Im3*kr3^2 + Il3*Im3*kr3^2 + Il3*a2^2*ml3 + Il3*a1
[(Il3*Im2*kr2 - Il3*Im2*kr2^2 - Im2*Im3*kr2^2*kr3 + Im2*kr2*l3^2*ml3 - Im2*kr2^2*l3^2*ml3 + Im2*Im3*kr2*kr3 - a1^3*l3*ml3*mm3*cos(theta2 + theta3) - a1*a2^2*l3

C_hat =
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
```

Insert parameter

```
% Define the given joint angles
theta1 = -pi / 2;
theta2 = pi/2;
theta3 = 0;

% Define the derivatives of joint angles (speeds and accelerations) as zero
dtheta1 = 0;
dtheta2 = 0;
dtheta3 = 0;

% Define the torques as zero
tau1 = 0;
tau2 = 0;
tau3 = 0;

% Define the link lengths
a1 = 0.5;
a2 = 0.5;
a3 = 0.5;

% Define the distance from link central mass to motor
l1 = a1 / 2;
l2 = a2 / 2;
l3 = a3 / 2;

% Define the masses
ml1 = 0.5;
ml2 = 0.5;
ml3 = 0.5;
mm1 = 0.5;
mm2 = 0.5;
mm3 = 0.5;

% Define the moments of inertia for links
Il1 = (ml1 * a1^2) / 12;
Il2 = (ml2 * a2^2) / 12;
Il3 = (ml3 * a3^2) / 12;

% Define the gear ratios (example values, you may need to adjust)
kr1 = 100;
kr2 = 100;
kr3 = 100;

% Define the moments of inertia for motors
% Example: Im = k * mm * r^2 (where r is the motor shaft radius and k is a constant)
% For simplicity, let's assume k = 1 and r = 0.01m
r = 0.01;
Im1 = mm1 * r^2;
Im2 = mm2 * r^2;
Im3 = mm3 * r^2;

% Insert the parameters into the A_hat, B_hat, C_hat matrices
A_hat_numeric =  double(vpa(subs(A_hat), 32));
B_hat_numeric =  double(vpa(subs(B_hat), 32));
C_hat_numeric =  double(vpa(subs(C_hat), 32));

% Display the numeric state-space matrices
disp('A_hat_numeric =');
disp(vpa(A_hat_numeric, 6));  % Display with 6 decimal places

disp('B_hat_numeric =');
disp(vpa(B_hat_numeric, 6));

disp('C_hat_numeric =');
disp(vpa(C_hat_numeric, 6));
```

```
A_hat_numeric =
[       0,         0,         0, 1.0,   0,   0]
[       0,         0,         0,   0, 1.0,   0]
[       0,         0,         0,   0,   0, 1.0]
[-7.77942,  -2.69827, -0.467624,   0,   0,   0]
[ 1.04786, -0.225818, 0.0141864,   0,   0,   0]
[ 1.35667,  0.142804, -0.355128,   0,   0,   0]

B_hat_numeric =
[         0,         0,          0]
[         0,         0,          0]
[         0,         0,          0]
[  0.787374, -0.371144, -0.0838863]
[ -0.371144,   1.24289,   -0.17569]
[-0.0838863,  -0.17569,    1.89847]

C_hat_numeric =
[1.0,   0,   0, 0, 0, 0]
[  0, 1.0,   0, 0, 0, 0]
[  0,   0, 1.0, 0, 0, 0]
```

Stability, Controllability, and Observability

```
% Compute the Jordan form of A_hat
[J, V] = jordan(A_hat_numeric);

% Display the Jordan form
disp('Jordan Form of A_hat =');
disp(J);

% Extract the diagonal of V which contains the eigenvalues
eigenvalues = diag(V);

% Display the eigenvalues in a+bi form with 4 decimal places
disp('Eigenvalues in the diagonal of V (in a+bi form):');
for i = 1:length(eigenvalues)
    realPart = real(eigenvalues(i));
    imagPart = imag(eigenvalues(i));
    fprintf('%.4f %+ .4fi\n', realPart, imagPart);
end
```

```
Jordan Form of A_hat =
  Columns 1 through 4

  -0.0000 + 0.0249i   0.0000 + 1.9232i   0.0000 - 0.3941i  -0.0000 - 1.9232i
   0.0000 - 0.3551i  -0.0000 - 0.2845i  -0.0000 + 0.8267i   0.0000 + 0.2845i
   0.0000 + 1.6544i   0.0000 - 0.3704i  -0.0000 + 1.1911i  -0.0000 + 0.3704i
   0.0150 + 0.0000i  -5.1927 + 0.0000i  -0.3308 - 0.0000i  -5.1927 + 0.0000i
  -0.2147 - 0.0000i   0.7682 + 0.0000i   0.6941 + 0.0000i   0.7682 + 0.0000i
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i

  Columns 5 through 6

  -0.0000 + 0.3941i   0.0000 - 0.0249i
   0.0000 - 0.8267i  -0.0000 + 0.3551i
   0.0000 - 1.1911i  -0.0000 - 1.6544i
  -0.3308 - 0.0000i   0.0150 + 0.0000i
   0.6941 + 0.0000i  -0.2147 - 0.0000i
   1.0000 + 0.0000i   1.0000 + 0.0000i

Eigenvalues in the diagonal of V (in a+bi form):
-0.0000 -0.6045i
0.0000 +2.7000i
0.0000 -0.8395i
-0.0000 -2.7000i
-0.0000 +0.8395i
0.0000 +0.6045i
```

Stability Type Identification

Initialize stability type

```
stabilityType = 'Stable';

% Check for stability type based on eigenvalues
if all(real(eigenvalues) < 0)
    stabilityType = 'Asymptotically Stable';
elseif any(real(eigenvalues) > 0)
    stabilityType = 'Unstable';
elseif any(real(eigenvalues) == 0)
    stabilityType = 'Marginally Stable';
end

% Display stability type
disp(['Stability Type: ', stabilityType]);
```

```
Stability Type: Unstable
```

Controllability

Calculate the controllability matrix P

```
P = [B_hat_numeric, A_hat_numeric * B_hat_numeric, A_hat_numeric^2 * B_hat_numeric];

% Display the controllability matrix
disp('Controllability matrix P =');
disp(vpa(P, 6));

% Check the rank of P
rank_P = rank(P);
disp(['Rank of P = ', num2str(rank_P)]);

% Check if the system is fully controllable
if rank_P == size(A_hat_numeric, 1)
    disp('The system is fully controllable.');
else
    disp('The system is not fully controllable.');
end
```

```
Controllability matrix P =
[         0,         0,          0,   0.787374, -0.371144, -0.0838863,        0,         0,          0]
[         0,         0,          0,  -0.371144,   1.24289,   -0.17569,        0,         0,          0]
[         0,         0,          0, -0.0838863,  -0.17569,    1.89847,        0,         0,          0]
[  0.787374, -0.371144, -0.0838863,          0,         0,          0, -5.08464, -0.384203,   0.238877]
[ -0.371144,   1.24289,   -0.17569,          0,         0,          0, 0.907679, -0.672066, -0.0212948]
[-0.0838863,  -0.17569,    1.89847,          0,         0,          0,    1.045, -0.263638,  -0.813094]

Rank of P = 6
The system is fully controllable.
```

Observability

Calculate the observability matrix Q

```
Q = [C_hat_numeric; C_hat_numeric * A_hat_numeric; C_hat_numeric * A_hat_numeric^2];

% Display the observability matrix
disp('Observability matrix Q =');
disp(vpa(Q, 6));

% Check the rank of Q
rank_Q = rank(Q);
disp(['Rank of Q = ', num2str(rank_Q)]);

% Check if the system is fully observable
if rank_Q == size(A_hat_numeric, 1)
    disp('The system is fully observable.');
else
    disp('The system is not fully observable.');
end

% Converting symbolic to numeric matrices
A_hat_numeric = double(vpa(subs(A_hat), 32));
B_hat_numeric = double(vpa(subs(B_hat), 32));
C_hat_numeric = double(vpa(subs(C_hat), 32));
% Initialization
STSP_D = zeros(3, 3); % 3x3 zero matrix for D
STSP_A = double(A_hat_numeric);
STSP_B = double(B_hat_numeric);
STSP_C = double(C_hat_numeric);

% Define the Q and R matrices for LQR
Q = diag(repmat(1, 1, 6));  % State weights
R = diag(repmat(0.1, 1, 3));  % Control input weights
```

```
Observability matrix Q =
[     1.0,         0,         0,   0,   0,   0]
[       0,       1.0,         0,   0,   0,   0]
[       0,         0,       1.0,   0,   0,   0]
[       0,         0,         0, 1.0,   0,   0]
[       0,         0,         0,   0, 1.0,   0]
[       0,         0,         0,   0,   0, 1.0]
[-7.77942,  -2.69827, -0.467624,   0,   0,   0]
[ 1.04786, -0.225818, 0.0141864,   0,   0,   0]
[ 1.35667,  0.142804, -0.355128,   0,   0,   0]

Rank of Q = 6
The system is fully observable.
```

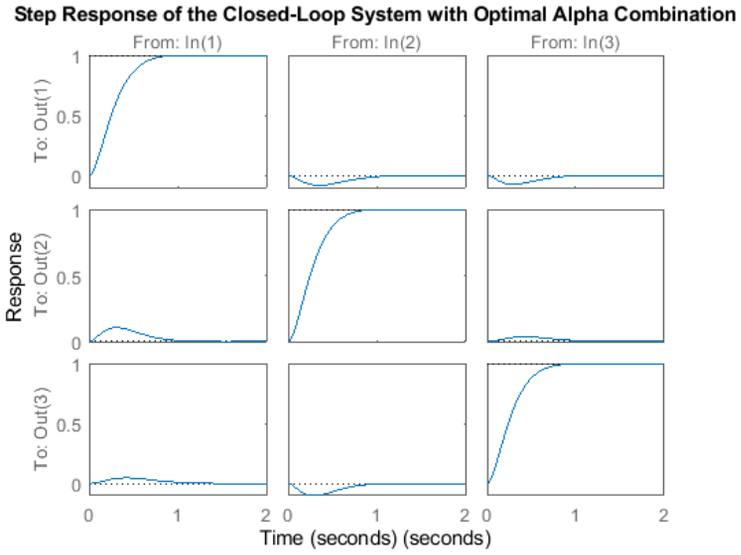
State Feedback Controller

```
% Initialization
STSP_D = zeros(3, 3); % 3x3 zero matrix for D
STSP_A = double(A_hat_numeric);
STSP_B = double(B_hat_numeric);
STSP_C = double(C_hat_numeric);

% Define ranges for alpha_1, alpha_2, and alpha_3
alpha_1_values = 3:0.1:5;
alpha_2_values = 6:0.1:8;
alpha_3_values = 6:0.1:8;

% Initialize a matrix to store the performance and count of IO pairs meeting criteria for each combination
alpha_combinations_performance = zeros(length(alpha_1_values)*length(alpha_2_values)*length(alpha_3_values), 5); % Columns: [alpha_1, alpha_2, alpha_3, count,
combination_idx = 1;

% Target performance criteria
target_overshoot = 5;  % percent
target_settling_time = 3;  % seconds

% Number of input-output pairs
num_io_pairs = size(STSP_B, 2) * size(STSP_C, 1);

% Loop over each combination of alpha values
for alpha_1 = alpha_1_values
    for alpha_2 = alpha_2_values
        for alpha_3 = alpha_3_values
```

```
            % Define the desired poles
            desired_poles = [-alpha_1 + 2.7000i, -alpha_1 - 2.7000i, -alpha_2 + 0.8395i, -alpha_2 - 0.8395i, -alpha_3 + 0.6045i, -alpha_3 - 0.6045i];

            % Calculate the state feedback matrix K
            K = place(STSP_A, STSP_B, desired_poles);

            % Closed-loop A matrix
            A_cl = STSP_A - STSP_B * K;

            % Calculate DC gain and its inverse for input scaling
            ssys_temp = ss(A_cl, STSP_B, STSP_C, zeros(size(STSP_C,1),size(STSP_B,2)));
            dc_gain = dcgain(ssys_temp);
            input_scaling = inv(dc_gain);

            % Apply the scaling to the B matrix
            STSP_B_scaled = STSP_B * input_scaling;

            % Closed-loop system with scaled input
            ssys_cl = ss(A_cl, STSP_B_scaled, STSP_C, zeros(size(STSP_C,1),size(STSP_B,2)));

            % Initialize variables for performance calculation
            total_overshoot = 0;
            total_settling_time = 0;
            count_meeting_criteria = 0;

            % Check each input-output pair
            for i = 1:size(STSP_B_scaled, 2)
                for j = 1:size(STSP_C, 1)
                    [y, t] = step(ssys_cl(j,i));
                    info = stepinfo(y, t);
                    total_overshoot = total_overshoot + info.Overshoot;
                    total_settling_time = total_settling_time + info.SettlingTime;

                    % Check criteria for each IO pair
                    if info.Overshoot <= target_overshoot && info.SettlingTime <= target_settling_time
                        count_meeting_criteria = count_meeting_criteria + 1;
                    end
                end
            end

            % Calculate average performance
            avg_overshoot = total_overshoot / num_io_pairs;
            avg_settling_time = total_settling_time / num_io_pairs;

            % Performance measure
            performance = avg_overshoot + avg_settling_time;

            % Store the alpha combination, count, and performance
            alpha_combinations_performance(combination_idx, :) = [alpha_1, alpha_2, alpha_3, count_meeting_criteria, performance];
            combination_idx = combination_idx + 1;
        end
    end
end

% Find the combinations where the maximum number of IO pairs meet the criteria
max_count = max(alpha_combinations_performance(:,4));
best_combinations = alpha_combinations_performance(alpha_combinations_performance(:,4) == max_count, :);

% If multiple combinations meet the criteria, select the one with the minimum performance
if size(best_combinations, 1) > 1
    [~, min_perf_idx] = min(best_combinations(:, 5));
    best_combination = best_combinations(min_perf_idx, 1:3);
else
    best_combination = best_combinations(1, 1:3);
end

% Extract the best alpha values
best_alpha_1 = best_combination(1);
best_alpha_2 = best_combination(2);
best_alpha_3 = best_combination(3);

% Display the best alpha combination
fprintf('Best alpha_1: %f, alpha_2: %f, alpha_3: %f\n', best_alpha_1, best_alpha_2, best_alpha_3);

% Recalculate K and input scaling for the best alpha combination
desired_poles_best = [-best_alpha_1 + 2.7000i, -best_alpha_1 - 2.7000i, -best_alpha_2 + 0.8395i, -best_alpha_2 - 0.8395i, -best_alpha_3 + 0.6045i, -best_alpha_
K_best = place(STSP_A, STSP_B, desired_poles_best);
A_cl_best = STSP_A - STSP_B * K_best;
ssys_best = ss(A_cl_best, STSP_B, STSP_C, zeros(size(STSP_C,1),size(STSP_B,2)));
dc_gain_best = dcgain(ssys_best);
input_scaling_best = inv(dc_gain_best);
STSP_B_scaled_best = STSP_B * input_scaling_best;

% Closed-loop system with the best alpha combination
ssys_cl_best = ss(A_cl_best, STSP_B_scaled_best, STSP_C, zeros(size(STSP_C,1),size(STSP_B,2)));

% Plot the step response for the best alpha combination
figure;
step(ssys_cl_best);
title('Step Response of the Closed-Loop System with Optimal Alpha Combination');
xlabel('Time (seconds)');
ylabel('Response');
```

```
% Check and print ts and overshoot for each input-output pair in the best alpha combination
for i = 1:size(STSP_B_scaled_best, 2)
    for j = 1:size(STSP_C, 1)
        [y, t] = step(ssys_cl_best(j,i));
        info = stepinfo(y, t);
        fprintf('Input %d - Output %d: Overshoot = %f%%, Settling Time = %f s\n', i, j, info.Overshoot, info.SettlingTime);
    end
end
```

```
Best alpha_1: 5.000000, alpha_2: 7.500000, alpha_3: 7.300000
Input 1 - Output 1: Overshoot = 0.000000%, Settling Time = 0.770050 s
Input 1 - Output 2: Overshoot = 0.000000%, Settling Time = 1.476550 s
Input 1 - Output 3: Overshoot = 0.000000%, Settling Time = 1.877892 s
Input 2 - Output 1: Overshoot = 0.000000%, Settling Time = 1.765974 s
Input 2 - Output 2: Overshoot = 0.002247%, Settling Time = 0.757795 s
Input 2 - Output 3: Overshoot = 0.000000%, Settling Time = 1.352534 s
Input 3 - Output 1: Overshoot = 0.000000%, Settling Time = 1.358327 s
Input 3 - Output 2: Overshoot = 0.000000%, Settling Time = 1.878043 s
Input 3 - Output 3: Overshoot = 0.000000%, Settling Time = 0.757786 s
```

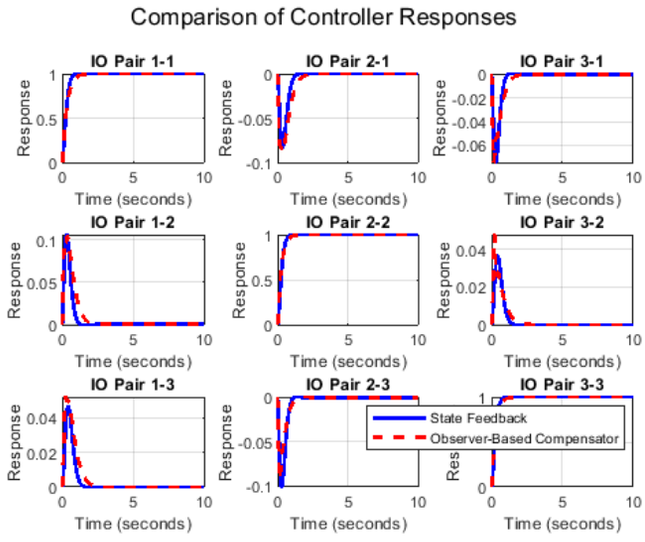


Choose observer poles closer to the dominant poles of the state feedback controller

```
scaling_factor = 5;
desired_poles_obs = desired_poles_best*scaling_factor;
% K_lqr = place(STSP_A, STSP_B, desired_poles_obs);
% Calculate observer gain matrix L
L = place(A_hat_numeric', C_hat_numeric', desired_poles_obs)';
% State-space models for State Feedback Controller and Observer-Based Compensator
% Augmented System Matrices
A_aug = [A_hat_numeric - B_hat_numeric * K_best, -B_hat_numeric * K_best;
         L * C_hat_numeric, A_hat_numeric - L * C_hat_numeric];
B_aug = [B_hat_numeric; zeros(size(B_hat_numeric))];

C_aug = [C_hat_numeric, zeros(size(C_hat_numeric))];
D_aug = zeros(size(C_hat_numeric, 1), size(B_hat_numeric, 2));
% State-space model of the augmented system
sys_compensator_temp = ss(A_aug, B_aug, C_aug, D_aug);
dc_gain = dcgain(sys_compensator_temp);
input_scaling = inv(dc_gain);
B_aug_scaled = B_aug * input_scaling;
sys_compensator1 = ss(A_aug, B_aug_scaled, C_aug, D_aug);
system =ss(A_hat_numeric,B_hat_numeric,C_hat_numeric,D_aug);
% Time vector for simulation
t = 0:0.01:10; % Time vector
% Number of inputs and outputs
num_inputs = size(B_hat_numeric, 2);
num_outputs = size(C_hat_numeric, 1);
% Create a new figure
figure;
% Loop through each input-output pair and plot responses
for i = 1:num_inputs
    for j = 1:num_outputs
            % Linear index for subplot
        subplot_idx = (j - 1) * num_inputs + i;
        subplot(num_outputs, num_inputs, subplot_idx);

        % Responses

        [response_sys, ~] = step(system(j, i), t);
        [response_sf, ~] = step(ssys_cl_best(j, i), t);
        [response_obs, ~] = step(sys_compensator1(j, i), t);

        % Plotting
        plot(t, response_sf, 'b-', t, response_obs, 'r--', 'LineWidth', 2);
```

```
        xlabel('Time (seconds)');
        ylabel('Response');
        title(sprintf('IO Pair %d-%d', i, j));
        grid on;
    end
end
legend('State Feedback', 'Observer-Based Compensator');
sgtitle('Comparison of Controller Responses');
fprintf('Settling Time of SF vs OBS\n');
for i = 1:num_inputs
    for j = 1:num_outputs
        [y, t] = step(ssys_cl_best(j,i));
        info_sf = stepinfo(y, t);
        [y, t] = step(sys_compensator1(j,i));
        info_obs = stepinfo(y, t);
        fprintf('I/P %d - O/P %d: OverShoot = %f SF= %.2fs,  OBC= %.2fs\n' ...
            , i, j,info_sf.Overshoot, info_sf.SettlingTime, ...
            info_obs.SettlingTime);
    end
end
```

```
Settling Time of SF vs OBS
I/P 1 - O/P 1: OverShoot = 0.000000 SF= 0.77s,  OBC= 1.16s
I/P 1 - O/P 2: OverShoot = 0.000000 SF= 1.48s,  OBC= 2.89s
I/P 1 - O/P 3: OverShoot = 0.000000 SF= 1.88s,  OBC= 3.07s
I/P 2 - O/P 1: OverShoot = 0.000000 SF= 1.77s,  OBC= 2.74s
I/P 2 - O/P 2: OverShoot = 0.002247 SF= 0.76s,  OBC= 0.97s
I/P 2 - O/P 3: OverShoot = 0.000000 SF= 1.35s,  OBC= 1.72s
I/P 3 - O/P 1: OverShoot = 0.000000 SF= 1.36s,  OBC= 2.30s
I/P 3 - O/P 2: OverShoot = 0.000000 SF= 1.88s,  OBC= 2.69s
I/P 3 - O/P 3: OverShoot = 0.000000 SF= 0.76s,  OBC= 1.01s
```



Published with MATLAB® R2023a