University of Calabria

Department of Computer, Modeling, Electronics and Systems Engineering

Rocket Booster Stabilization: Modeling, Stability Analysis, and Control Design

Final Project for Dynamical Systems Theory

Student: Martin Haugen, 265053
Professor: Alessandro Casavola

UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Academic Year 2024-2025, First Semester

Abstract

This project presents the modeling, analysis, and control design of a rocket booster stabilization system. A nonlinear model of the booster dynamics was developed and subsequently linearized around an equilibrium point to facilitate stability analysis and controller design. Internal and external stability were investigated using Lyapunov's indirect method and BIBO stability criteria. A state feedback controller was designed using pole placement to ensure desired closed-loop performance. To address the challenge of unmeasured states, a Luenberger observer was designed to estimate the full state vector from available measurements. Finally, a compensator was implemented that combined the controller and the observer to stabilize the system using estimated states. The simulation results demonstrated the effectiveness of the proposed control strategy, achieving stabilization of the booster around the chosen equilibrium point. This work highlights the importance of stability analysis and state estimation in the design of control systems for aerospace applications.

Contents

1 Introduction

Space exploration has always been a source of fascination and inspiration, leading humanity to push the limits of what is possible. One of the most significant advances in recent years has been the development of reusable rockets, spearheaded by companies like SpaceX. These advancements mark a monumental shift in the aerospace industry, moving toward more sustainable and cost-effective solutions.

Reusability in spaceflight offers profound economic and environmental benefits. By reducing the need to build entirely new rockets for each new mission, launch costs are significantly reduced, making space more accessible. In addition, minimizing waste aligns with global efforts toward sustainability, reducing the environmental impact of space exploration. This progress could bring us closer to reaching ambitious goals like manned missions to Mars or establishing lunar bases.

However, the process of making rockets reusable presents a myriad of challenges. A critical aspect of this is stabilizing the booster during descent and landing. Factors such as aerodynamic forces, high sensitivity to disturbances, and the need for precise control systems make this a highly complex problem. Successful stabilization requires robust modeling of the dynamics of the system, careful design of control strategies, and reliable estimation of states that cannot be measured directly.

In this project, my objective is to explore the stabilization of a rocket booster during its descent phase. By modeling the system, analyzing its dynamics, and synthesizing controllers, I investigate the feasibility of achieving stable and controlled landings. It should be noted that, to make this project fit as a first semester project in dynamical systems, a number of simplifications have been made.

2 System Modeling

2.1 Overview

The system under consideration is a rocket booster in a 2D plane, simplifying the three-dimensional dynamics into a planar system for tractability. This model represents the booster as a rigid body with three thrusters; one located at the bottom and two lateral thrusters on either side. The bottom thruster provides vertical acceleration, while the lateral thrusters control angular acceleration. The primary forces acting on the booster include:

- Thrust forces generated by the thrusters.

- Gravity, modeled as constant.

- Air resistance, modeled using a drag equation.

A schematic of the booster with labeled components and force vectors is shown in Figure 1. These visual aids serve to clarify the geometry and dynamic interactions of the system.

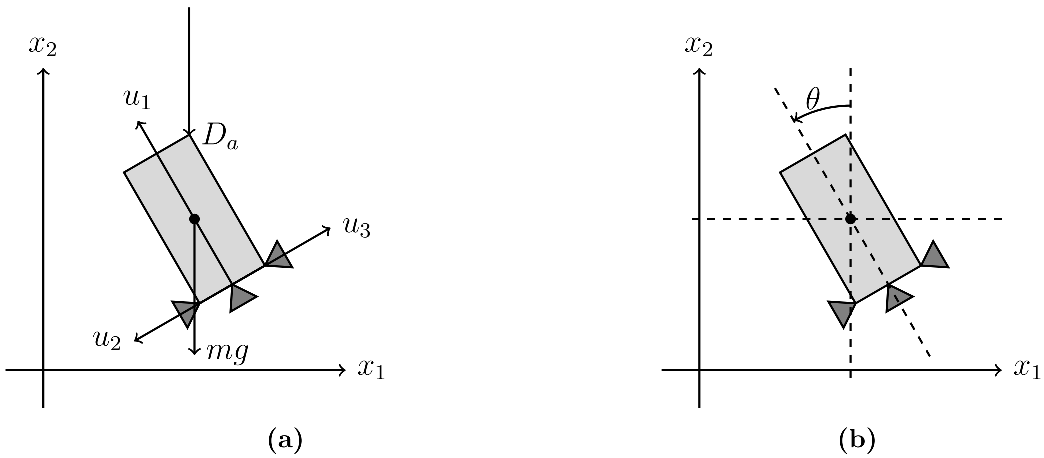

Figure 1: Force (a) and position (b) schematic of rocket booster system

2.2 Assumptions and Simplifications

To make the system more analytically tractable, several assumptions and simplifications were made:

- The system is analyzed in a 2D plane.

- The booster is considered a rigid body, disregarding the internal fuel sloshing effect.

- The mass of the booster is assumed constant, neglecting the reduction in mass due to fuel consumption during the descent phase.

- Gravity is modeled as a constant acceleration, one g, as the stabilization occurs close to the Earth's surface.

- Air resistance is modeled using the drag equation with parameters appropriate for a cylinder, as the booster is assumed to remain upright with small angular perturbations.

- The area in the drag equation corresponds to the cross-sectional area of the booster's bottom.

- The moment of inertia J is calculated using the formula for a rectangular plate [1]:

$$J = \frac{1}{12}m(b^2 + h^2) \tag{1}$$

where m is the mass, b is the width, and h is the height of the booster.

These assumptions, while idealized, allow for a focus on the essential dynamics relevant to vertical and angular stabilization during descent.

2.3 Mathematical Modeling

The equations of motion were derived using Newton's Second Law and d'Alembert's Principle. The translational and rotational dynamics are given as follows:

$$\ddot{x}_2 = \frac{u_1}{m}cos\theta - \frac{1}{2m}\rho C_d A\dot{x}_2^2 - g \tag{2}$$

$$\ddot{\theta} = \frac{h}{2J}(u_3 - u_2) \tag{3}$$

Where:

- $\ddot{x}_2$ represents the vertical acceleration

- $\ddot{\theta}$ represents the angular acceleration

- u_1, u_2, u_3 are the thrust inputs for the bottom, right, and left thrusters, respectively.

- ρ is the air density.

- C_d is the drag coefficient for a cylinder

- A is the cross-sectional area of the booster

- g is the gravitational acceleration.

A screenshot of the Simulink model is shown in Figure 2.

Parameters

The parameters used in the equations are summarized in Table 1

3 Linear System Representation

The dynamics of the system are represented in state-space form[2], a widely-used method in control theory that enables structured analysis and controller design. This representation

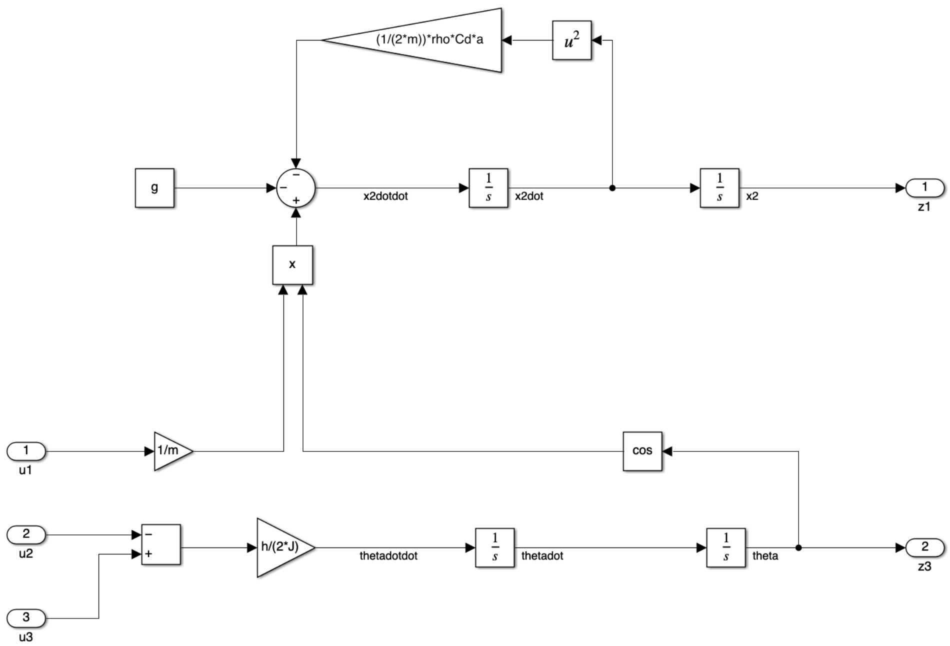

Figure 2: Simulink model of the rocket booster

Parameter	Description	Value
m	Mass of the booster	$500000\,kg$
ρ	Air density	$1.225\,kg/m^3$
C_d	Drag coefficient for a cylinder	1.2
A	Cross-sectional area	$78.5398\,m^2$
J	Moment of inertia of the booster	$2.0833e+08\,kgm^2$
g	Gravitational acceleration	$9.81\,m/s^2$
h	Height of booster	$70\,m$
b	Width of booster	$10\,m$

Table 1: Parameters for rocket booster system

expresses the system's behavior in terms of state variables z, control inputs u, and outputs y. The chosen states are:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ \dot{x}_2 \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{4}$$

The following equation shows the state-space realization of the system. Time dependency for state and input is implied.

$$\dot{z} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_2 \\ \frac{u_1}{M} cos z_3 - \frac{1}{2m}\rho C_d A \dot{z}_2^2 - g \\ z_4 \\ \frac{h}{2J}(u_3 - u_2) \end{bmatrix} \tag{5}$$

As for the output, I have chosen to monitor the first and third state, which are the vertical and angular position, respectively. Time dependence for the output is implied.

$$y = \begin{bmatrix} x_2 \\ \theta \end{bmatrix} = \begin{bmatrix} z_1 \\ z_3 \end{bmatrix} \tag{6}$$

3.1 Equilibrium and Linearization

To further analyze the system and to be able to synthesize a controller, we have to find a linear representation of the system. The linearized version of the system is valid around a small interval of a nominal solution. The nominal solution would for example be an equilibrium point for the system.

An equilibrium point is a point where all the derivatives of the states are equal to zero.

$$\begin{bmatrix} \tilde{z}_2 \\ \frac{\tilde{u}_1}{M} cos \tilde{z}_3 - \frac{1}{2m}\rho C_d A \dot{\tilde{z}}_2^2 - g \\ \tilde{z}_4 \\ \frac{h}{2J}(\tilde{u}_3 - \tilde{u}_3) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

This system of equations has an infinite number of solutions. A logical choice of an equilibrium point for this system is a point where the booster is still in an upright position. We can choose $z_1 = \tilde{z}_1$ and get:

$$\tilde{z} = \begin{bmatrix} \tilde{z}_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \tilde{u} = \begin{bmatrix} mg \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

Choosing $\tilde{z}_1 = \frac{h}{2}$, so that the bottom of the booster is at 0, we get the behavior reported in Figure 3.

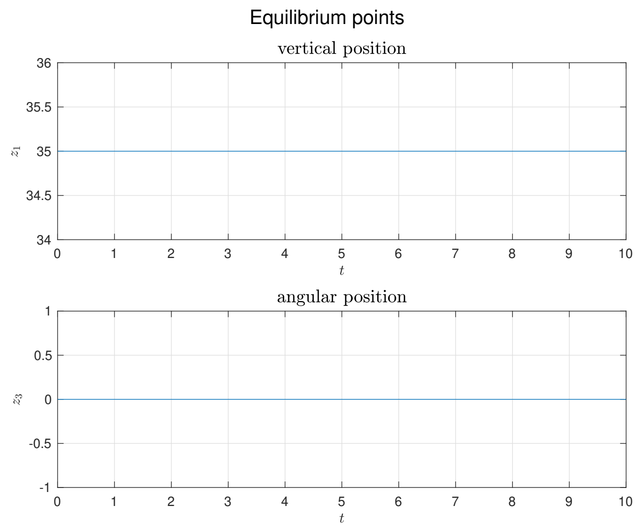

Figure 3: System behavior in chosen equilibrium point

Linearization

We want our system to be on the form

$$\begin{cases} \dot{z}(t) = Az(t) + Bu(t),\ z(0) = z_0 \\ y(t) = Cz(t) + Du(t) \end{cases} \tag{9}$$

and since we want to linearize around our nominal solution $\tilde{z}$ we define new state-space variables as the increments $\delta z = z - \tilde{z}$, $\delta u = u - \tilde{u}$ and $\delta y = y - \tilde{y}$. The linear representation we get around this nominal solution is shown in Equation 10.

$$\begin{cases} \delta\dot{z}(t) = A\delta z(t) + B\delta u(t),\ \delta z(0) = z_0 - \tilde{z} \\ \delta y(t) = C\delta z(t) + D\delta u(t) \end{cases} \tag{10}$$

The matrices A, B, C, D can be calculated as

$$A = \frac{\partial f(z,u)}{\partial z}\Big|_{\tilde{z},\tilde{u}} = \begin{bmatrix} \frac{\delta f_1}{\delta z_1} & \frac{\delta f_1}{\delta z_2} & \frac{\delta f_1}{\delta z_3} & \frac{\delta f_1}{\delta z_4} \\ \frac{\delta f_2}{\delta z_1} & \frac{\delta f_2}{\delta z_2} & \frac{\delta f_2}{\delta z_3} & \frac{\delta f_2}{\delta z_4} \\ \frac{\delta f_3}{\delta z_1} & \frac{\delta f_3}{\delta z_2} & \frac{\delta f_3}{\delta z_3} & \frac{\delta f_3}{\delta z_4} \\ \frac{\delta f_4}{\delta z_1} & \frac{\delta f_4}{\delta z_2} & \frac{\delta f_4}{\delta z_3} & \frac{\delta f_4}{\delta z_4} \end{bmatrix}_{\tilde{z},\tilde{u}} \tag{11}$$

$$B = \frac{\partial f(z,u)}{\partial u}\Big|_{\tilde{z},\tilde{u}} = \begin{bmatrix} \frac{\delta f_1}{\delta u_1} & \frac{\delta f_1}{\delta u_2} & \frac{\delta f_1}{\delta u_3} \\ \frac{\delta f_2}{\delta u_1} & \frac{\delta f_2}{\delta u_2} & \frac{\delta f_2}{\delta u_3} \\ \frac{\delta f_3}{\delta u_1} & \frac{\delta f_3}{\delta u_2} & \frac{\delta f_3}{\delta u_3} \\ \frac{\delta f_4}{\delta u_1} & \frac{\delta f_4}{\delta u_2} & \frac{\delta f_4}{\delta u_3} \end{bmatrix}_{\tilde{z},\tilde{u}} \tag{12}$$

$$C = \frac{\partial \eta(z,u)}{\partial z}\Big|_{\tilde{z},\tilde{u}} = \begin{bmatrix} \frac{\delta \eta_1}{\delta z_1} & \frac{\delta \eta_1}{\delta z_2} & \frac{\delta \eta_1}{\delta z_3} & \frac{\delta \eta_1}{\delta z_4} \\ \frac{\delta \eta_2}{\delta z_1} & \frac{\delta \eta_2}{\delta z_2} & \frac{\delta \eta_2}{\delta z_3} & \frac{\delta \eta_2}{\delta z_4} \end{bmatrix}_{\tilde{z},\tilde{u}} \tag{13}$$

$$D = \frac{\partial \eta(z,u)}{\partial u}\Big|_{\tilde{z},\tilde{u}} = \begin{bmatrix} \frac{\delta \eta_1}{\delta u_1} & \frac{\delta \eta_1}{\delta u_2} & \frac{\delta \eta_1}{\delta u_3} \\ \frac{\delta \eta_2}{\delta u_1} & \frac{\delta \eta_2}{\delta u_2} & \frac{\delta \eta_2}{\delta u_3} \end{bmatrix}_{\tilde{z},\tilde{u}} \tag{14}$$

where time dependence is implied.

Using the MATLAB function `linearize()` with $\tilde{z}_1 = \frac{h}{2}$ we obtain the following results:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 2e-06 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -1.68e-07 & 1.68e-07 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

4 Stability Analysis

Now that we have a linearized model of our system, we can analyze the stability of the system around the chosen equilibrium point. There are two different types of stability to look at; internal, and external. Both will be analyzed here.

4.1 Internal Stability

To analyze the internal stability of the nominal solution, we examine the system's response to small perturbations in the initial condition while keeping the input unchanged. *Lyapunov's Indirect Method*, also known as the Reduced Criterion of Stability, provides a straightforward approach for this analysis by evaluating the eigenvalues of the system matrix A. Using MATLAB function `eig()` we obtain

$$\lambda = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

This means that the linearized system is neither asymptotically stable nor unstable, but marginally stable. This often indicates the presence of conserved quantities or neutral modes in the system. This suggests that the system may not settle to a point but may exhibit behaviors such as drifting or oscillations in certain directions. To check the stability of the equilibrium point we have to analyze the nonlinear system when we apply small perturbations to the initial condition. Logically, this makes sense. If we change the initial value of the first state z_1 to a different height, we will see the exact same behavior. However, if we change the third state z_3, so that the booster is at a different initial angle, it may not behave the same. We expect that the booster starts falling when we introduce a different initial angle. The systems response to such a perturbation(Equation 16) is reported in Figure 4.

$$\tilde{z} = \begin{bmatrix} \frac{h}{2} \\ 0 \\ 18.2132 \\ 0 \end{bmatrix} \tag{16}$$

4.2 External Stability

To study the external stability of the nominal solution, we look at the behavior of the system when we only change the input. We say that the system is *bounded-input/bounded-output* (BIBO) *stable* if and only if all poles of the systems transfer function

$$W_{yu}(s) = C(sI - A)^{-1}B + D \tag{17}$$

have a negative real part. To simplify the analysis, it should be noted that the poles are a subset of the eigenvalues. Since we have all eigenvalues equal to 0, we can say that our system is not BIBO stable.

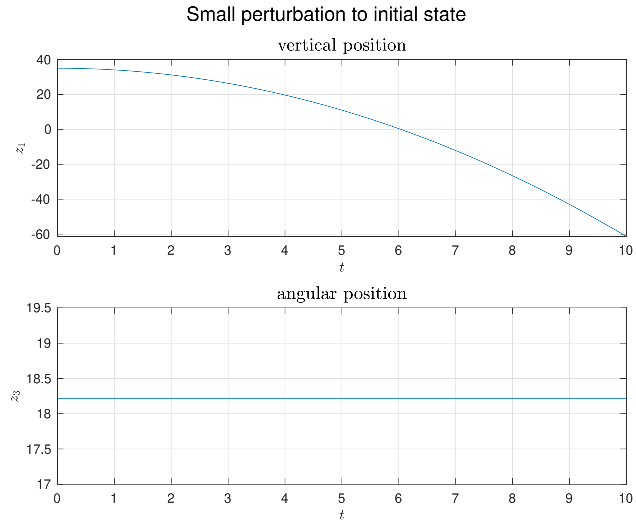

Figure 4: System behavior with small perturbation to initial angle

5 Controller Design

With the stability analysis completed, we proceed to design a controller to stabilize the system around the chosen equilibrium point. In this chapter, we discuss the conditions necessary for designing a state-feedback controller, describe the steps taken to achieve this, and present the results of the controller's performance.

5.1 Prerequisites for State-Feedback Control

State-feedback controllers rely on directly using the state variables of the system to compute control actions. To design such a controller, certain criteria must be satisfied:

- **Controllability**: The system must be controllable, meaning it is possible to drive all the state variables to any desired value using a finite control input. Mathematically, this is determined by the *controllability matrix*:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{18}$$

  where n is the number of state variables, in our case 4. The system is controllable if and only if the rank of $\mathcal{C}$ is equal to the dimension of the state space (n).

- **System Representation**: The system must be described in state-space form, with matrices A (system dynamics) and B (input dynamics) available.

- **Desired Closed-Loop Dynamics**: The desired closed-loop pole locations must be chosen to meet performance requirements, such as stability, damping ratio, and settling time.

5.2 Control Law Formulation

The state-feedback controller is designed by modifying the control input based on the deviation of the state from its equilibrium. The control law used here is defined as:

$$u(t) = u_0 - K(z(t) - z_0) \tag{19}$$

where:

- $u(t)$ is the control input applied to the system.
- u_0 is the equilibrium input corresponding to the nominal operating condition.
- $z(t)$ is the current state vector.
- z_0 is the equilibrium state vector.
- K is the state-feedback gain matrix designed to place the poles of the closed-loop system in desired locations.

This control law modifies the input by applying a correction proportional to the deviation of the state from its equilibrium.

5.3 Closed-Loop System Dynamics

By substituting the control law into the original linearized state-space model:

$$\begin{aligned} \dot{z}(t) &= Az(t) + Bu(t) && (20) \\ &= Az(t) + B\left[u_0 - K(z(t) - z_0)\right] && (21) \end{aligned}$$

Expanding the terms:

$$\dot{z}(t) = Az(t) + Bu_0 - BKz(t) + BKz_0 \tag{22}$$

Collecting terms:

$$\dot{z}(t) = (A - BK)z(t) + B(u_0 + Kz_0) \tag{23}$$

Since the input and state deviations are calculated around the equilibrium point where $Az_0 + Bu_0 = 0$, the constant term simplifies to zero, leading to the final closed-loop system equation:

$$\dot{z}(t) = (A - BK)z(t) \tag{24}$$

The matrix $A - BK$ determines the closed-loop dynamics of the system. The eigenvalues of this matrix correspond to the poles of the closed-loop system and can be chosen to meet desired performance criteria.

5.4 Controller Design Process

The system can be verified to be controllable by calculating the *controllability matrix*:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} \tag{25}$$

The rank of $\mathcal{C}$ was computed using the MATLAB `rank()` function, and it was confirmed that $\text{rank}(\mathcal{C}) = 4$.

Since the system is controllable, we can use the *Eigenvalue allocation theorem* which states that we can choose a set of eigenvalues using *Ackermann's formula*:

$$K = \begin{bmatrix} 0 & \dots & 0 & 1 \end{bmatrix} \mathcal{C}^{-1} P(A) \tag{26}$$

where $P(A)$ is the desired characteristic polynomial evaluated at A. We choose the desired pole locations as:

$$\text{Desired Poles} = \{-3 + 1.5j, -3 - 1.5j, -5, -10\} \tag{27}$$

The state-feedback gain matrix can then be computed using MATLAB's `place()` function.

5.5 Controller Performance

To test the controller, an initial perturbation is introduced to the system:

$$\Delta z_0 = \begin{bmatrix} 0 \\ 0 \\ \theta_{\text{tip}} + 10 \\ 0 \end{bmatrix} \tag{28}$$

The system is then simulated using the designed controller (Figure 7) on the non-linear model (Figure 5) as well as on the linearized model for comparison (Figure 6).

The controller successfully stabilizes the system, with all state variables returning to their equilibrium values with minor oscillations.

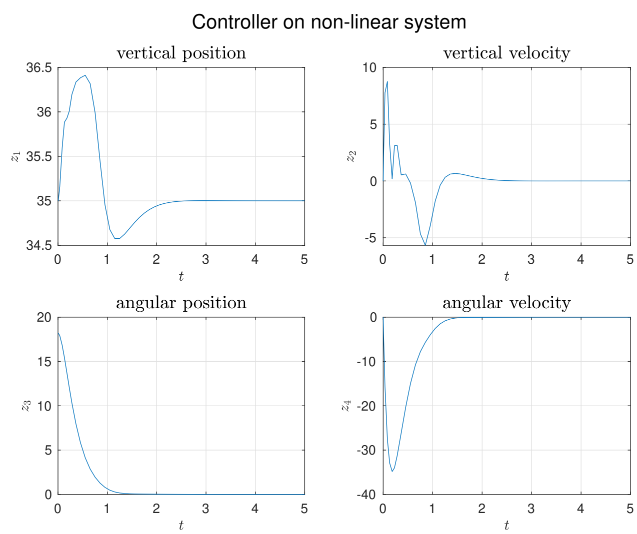

Figure 5: Controller performance on non-linear model

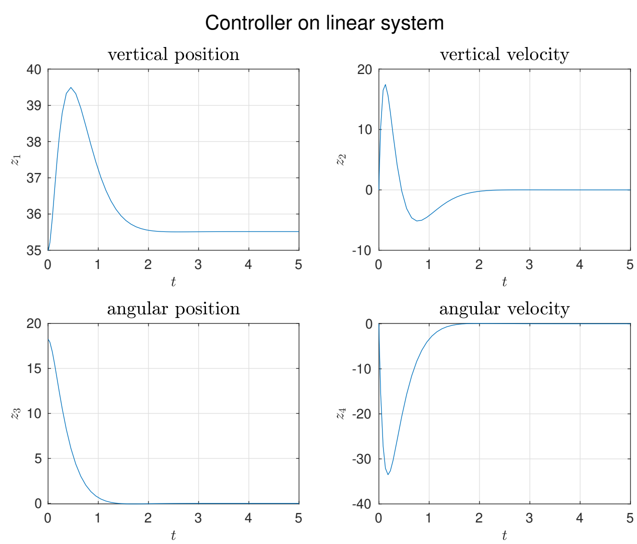

Figure 6: Controller performance on linear model

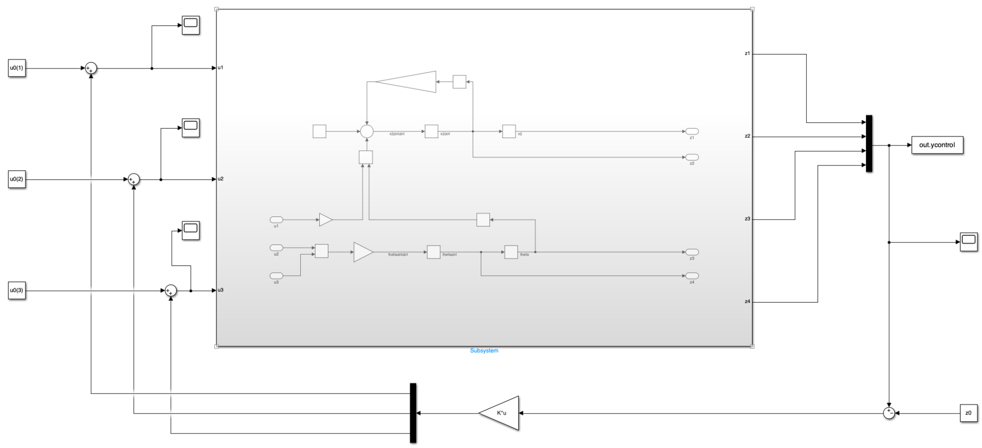

Figure 7: Simulink model of controller

6 Observer Design

In many practical control systems, it is not feasible or cost-effective to measure all the state variables directly. Some states may be difficult to measure due to physical constraints, sensor limitations, or cost considerations. For example, in aerospace applications, while measurements for position and velocity are often readily available, angular rates or internal forces may be harder to obtain directly.

The observer is designed to estimate the full state vector using only the available measurements of position and velocity. By reconstructing the unmeasured states, the observer ensures the controller has access to the full system state, enabling effective feedback control.

Assume that we can only measure the vertical and angular positions $z_1 = x_2, z_3 = \theta$. To accurately estimate the two non-measurable states $z_2 = \dot{x}_2, z_4 = \dot{\theta}$, we can design a *Luenberger Asymptotical Observer*.

6.1 Observability

A necessary condition for designing a state observer such as a *Luenberger Asymptotical Observer* is that the system is observable. Observability describes whether the system's state can be fully reconstructed from its output over time. Mathematically, a linear time-invariant system described by

$$\begin{cases} \dot{z}(t) = Az(t) + Bu(t), \ z(0) = z_0 \\ y(t) = Cz(t) + Du(t) \end{cases} \tag{29}$$

is said to be observable if the *observability matrix*

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{30}$$

is full rank, i.e, $\text{rank}(\mathcal{O}) = n$, where n is the number of states in the system, in our case 4. If the *observability matrix* is full rank, the system states can be uniquely determined from the outputs.

6.2 Luenberger Observer Design

The Luenberger observer estimates the state vector $\hat{z}$ using a dynamic model of the system combined with the measured outputs and a correction term based on the estimation error. The observer dynamics are given by:

$$\begin{cases} \dot{\hat{z}}(t) = A\hat{z}(t) + Bu(t) + L(y(t) - \hat{y}(t)) \\ \hat{y}(t) = C\hat{z}(t) \end{cases} \tag{31}$$

where:

- $\hat{z}(t)$ is the estimated state vector.
- L is the observer gain matrix.
- $y(t)$ is the measured output vector.

The observer gain matrix L is chosen such that the error dynamics $e(t) = z(t) - \hat{z}(t)$ decay to zero asymptotically. Substituting the error definition into the equations gives:

$$\dot{e}(t) = (A - LC)e(t) \tag{32}$$

The eigenvalues of the matrix $A-LC$ determine the convergence rate of the estimation error. The observer gain L can be selected using pole placement to assign desired eigenvalues for the error dynamics. The eigenvalues of $A - LC$ are usually chosen to be about 6 to 10 times further away from the imaginary axis than the eigenvalues of the controller system $A - BK$.

6.3 Observer Design Process

The system can be verified to be observable by computing the *observability matrix*:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix} \tag{33}$$

and using MATLAB's `rank()` function to find that the *observability matrix* is full rank. We then choose the poles to be:

$$\text{Desired Poles} = \{-15, -15, -30, -60\} \tag{34}$$

We then use MATLAB's `place()` function to find the *observer gain matrix* L.

6.4 Observer Performance

To test the observer, we simulate the system using the designed observer (Figure 11) without any perturbation to the initial condition, to see if it accurately can estimate the behavior of the system. As we can see from the results in Figure 10, the error does not decay to zero, but to a steady-state error. This may be due to unmodeled nonlinear dynamics in the linearized system. Since the observer is based on the linearized system, some dynamics may be omitted, and may then cause the observer to behave like we observe here.

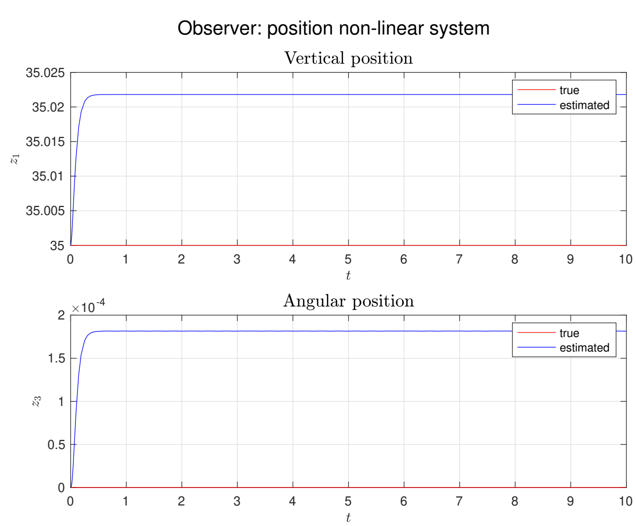

Figure 8: Observer performance on position

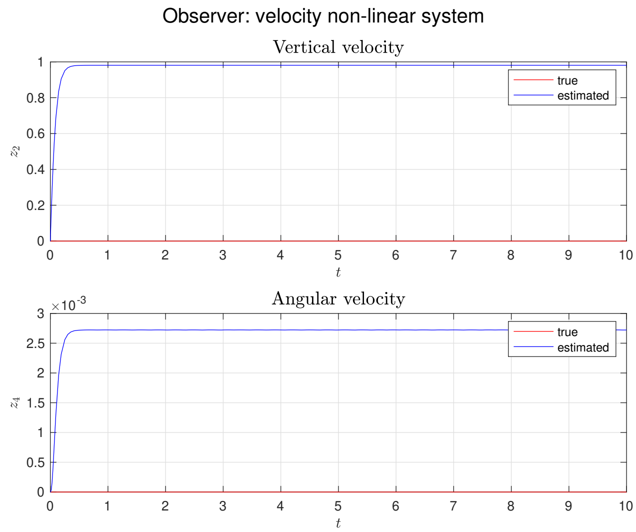

Figure 9: Observer performance on velocity

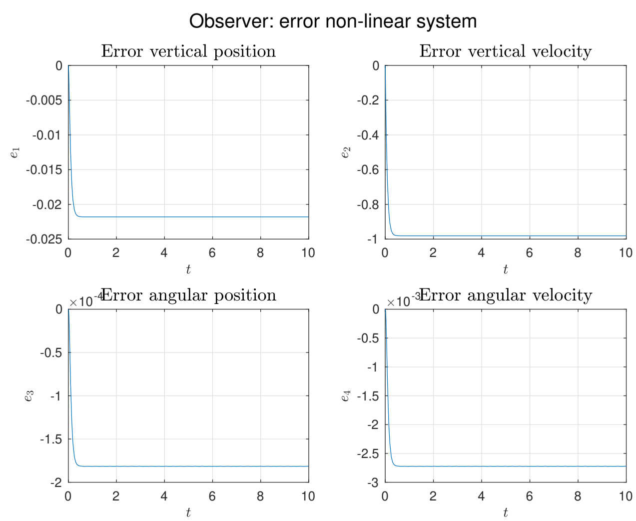

Figure 10: Observer error

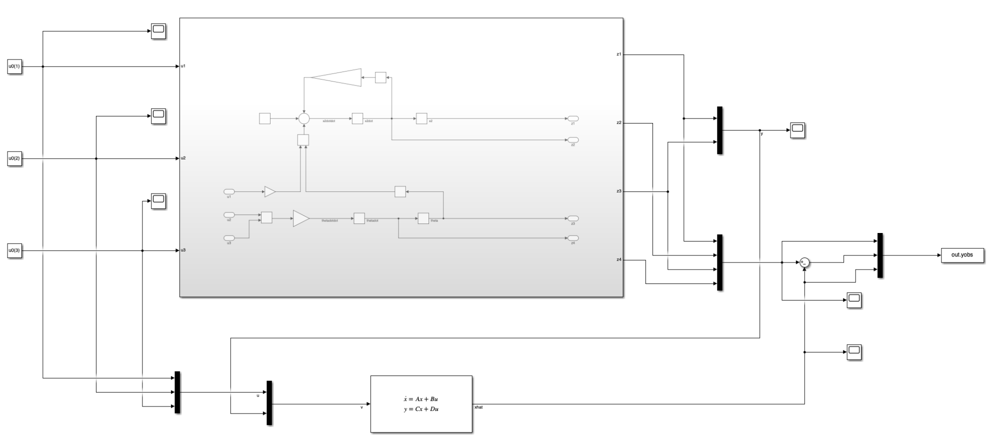

Figure 11: Simulink model of observer

7 Compensator Design

The compensator is constructed by combining the state feedback controller and the state observer developed in the previous chapters. This approach is often necessary in practical control systems where not all state variables are directly measurable. The observer estimates the full state vector from the available measurements, while the controller uses these estimated states to compute the control input.

The structure of the compensator is shown in Figure 13, where the observer provides the estimated state vector $\hat{z}$ and the control input is computed as

$$u = K\hat{z} + u_{eq} \tag{35}$$

where K is the state feedback gain previously designed.

The dynamics of the closed-loop compensator can be described by the combined observer-controller equations:

$$\begin{cases} \dot{z} = Az + BK\hat{z} + Bu_{eq} \\ \dot{\hat{z}} = LCz + (A - LC + BK)\hat{z} + Bu_{eq} \\ y = Cz + DK\hat{z} + Du_{eq} \end{cases} \tag{36}$$

where L is the observer gain matrix, ensuring convergence of the estimated states to the true states over time.

The compensator design allows us to regulate the system based on partial measurements, ensuring stability and desired dynamic behavior. The results of the compensator behavior when we introduce a small perturbation to the initial state can be seen in Figure 12.

This design integrates the observer and controller into a unified framework, balancing state estimation accuracy and control performance. The system response demonstrates satisfactory behavior with minimal estimation error and effective control actions.

8 Conclusion

In this project, we have analyzed the stability, controller design, observer design, and compensator synthesis for the booster dynamics system. Starting from a non-linear dynamical model, we derived a linearized representation around the chosen equilibrium point and examined the stability properties using both internal and external stability criteria. The stability

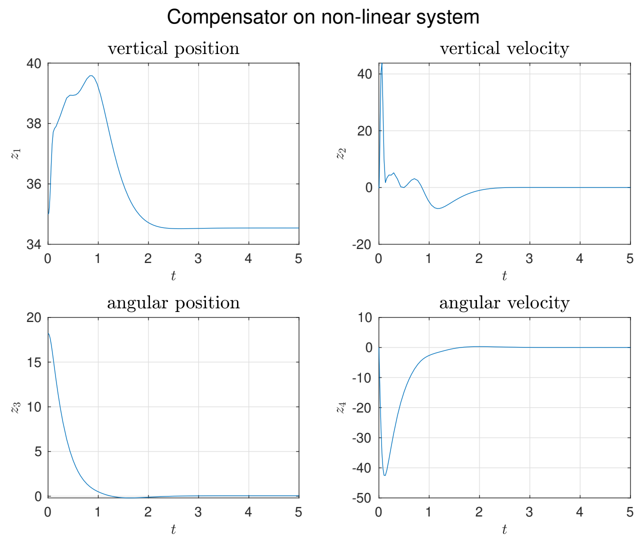

Figure 12: Compensator performance on nonlinear system

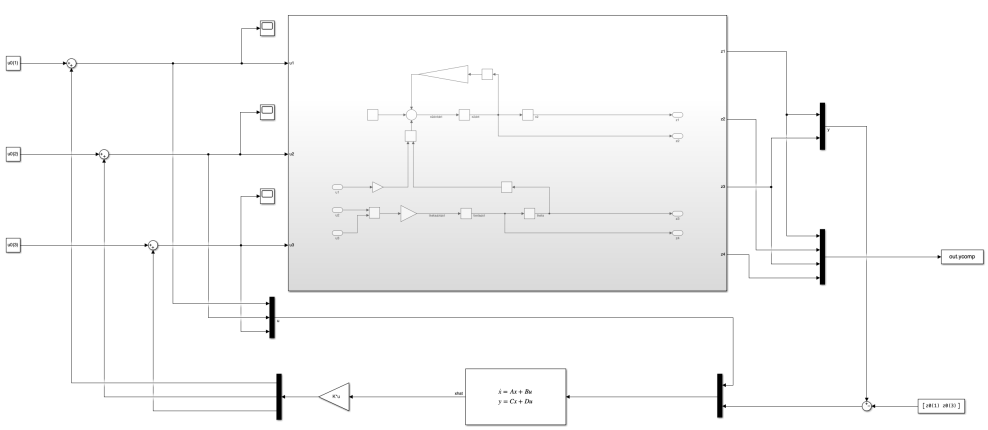

Figure 13: Simulink model of compensator

analysis revealed the presence of non-asymptotic behavior, prompting further investigation into control design strategies.

A state-feedback controller was developed using pole placement techniques, ensuring desired closed-loop dynamics for the system. The design process included verifying the controllability of the system and selecting an appropriate gain matrix to achieve stability and performance objectives.

To handle situations where all states could not be directly measured, a Luenberger observer was designed, estimating the unmeasured states using available position and angular measurements. Observability was confirmed, and the observer poles were placed to ensure rapid convergence of the estimated states.

Finally, a compensator was synthesized by combining the designed controller and observer, forming a complete dynamic feedback control system. The performance of the compensator was validated through simulation, demonstrating effective stabilization and regulation of the booster dynamics.

9 MATLAB code

```
%%%%%%%%%%%% project dynamical systems theory:
%%%%%%%%%%%% stabilizing a landing booster

clear;
close all;
clc;


%% parameters

Tfin = 10;                  % final simulation time

g = 9.81;                   % m/s^2 - gravity constant

m = 500000;                 % kg - booster with a little fuel left

h = 70;                     % m - height of booster

b = 10;                     % m - width of booster

rho = 1.225;                % kg/m^3 - air density

Cd = 1.2;                   % [] - drag coefficient (cylinder)

r = b/2;                    % m - radius of booster (from top view)
```

```
a = pi*r^2;                  % m^2 - surface area of bottom of
    booster

J = (1/12)*m*(b^2 + h^2);    % kg*m^2 - inertia of booster

drag = (1/2)*rho*Cd*a;       % kg*m/s^2 - constant drag

theta_tip = asin(b/h);           % tipping angle
theta_tip = rad2deg(theta_tip); % in degrees


%% equilibrium points

x2eq = h/2;
x2doteq = 0;
thetaeq = 0;
thetadoteq = 0;

u1eq = m*g;
u2eq = 0;
u3eq = 0;

z0 = [x2eq x2doteq thetaeq thetadoteq]';
u0 = [u1eq u2eq u3eq]';

model = 'nonlinear_system';
modelSim = sim(model);

t = modelSim.y.Time;

z1 = modelSim.y.Data(:,1);
z3 = modelSim.y.Data(:,2);

figure
subplot(2,1,1);
plot(t, z1);
title("vertical position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,1,2);
plot(t, z3);
title("angular position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
```

```
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

sgtitle('Equilibrium points');

print -depsc figures/eq_points


%% linearization

model_lin = 'nonlinear_sys_linearization';

op = operspec(model_lin);                     % operating point specification

op.States(1).x = z0(1);                       % x2
op.States(2).x = z0(2);                       % x2dot
op.States(3).x = z0(3);                       % theta
op.States(4).x = z0(4);                       % thetadot

stateorder = {'z1', 'z2', 'z3', 'z4'};        % state order

linsys = linearize(model_lin, op, 'StateOrder', stateorder);   % linearize model around eq point

[A, B, C, D] = ssdata(linsys);


%% stability

% internal stability - Lyapunovs reduced criteria of stability

eigenA = eig(A);                        % eigenvalues of A

dz0 = [0, 0, theta_tip+10, 0]';        % small perturbation to initial state
du0 = [0, 0, 0]';

model_stab = 'booster_stability';

out_stab = sim(model_stab);

t_stab = out_stab.ystab.Time;

z1_stab = out_stab.ystab.Data(:,1);
z3_stab = out_stab.ystab.Data(:,2);
```

```matlab

figure
subplot(2,1,1);
plot(t_stab, z1_stab);
title("vertical position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,1,2);
plot(t_stab, z3_stab);
title("angular position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

sgtitle('Small perturbation to initial state');

print -depsc figures/stability

% external stability - BIBO stability - all poles of tf negative real part

W = tf(linsys);

p = pole(W);


%% controller

Tfin_cont = 5;       % sim time for controller model

R = ctrb(A, B);      % controllability matrix
rankR = rank(R);     % rank of controllability matrix

if rankR == size(A, 1)
disp('The system is controllable');
else
disp('The system is NOT controllable');
end

desired_poles = [-3 + 1.5i, -3 - 1.5i, -5, -10];    % desired poles for the closed-loop system
K = -place(A, B, desired_poles);                    % state-feedback gain
```

```
dz0_cont = [0, 0, theta_tip+10, 0]';
z0_cont = z0 + dz0_cont;

model_cont = 'booster_control';

out_cont = sim(model_cont);

% on non-linear system

t_cont = out_cont.ycontrol.Time;

z1_cont = out_cont.ycontrol.Data(:,1);
z2_cont = out_cont.ycontrol.Data(:,2);
z3_cont = out_cont.ycontrol.Data(:,3);
z4_cont = out_cont.ycontrol.Data(:,4);

figure
subplot(2,2,1);
plot(t_cont, z1_cont);
title("vertical position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_cont, z2_cont);
title("vertical velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_cont, z3_cont);
title("angular position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

subplot(2,2,4);
plot(t_cont, z4_cont);
title("angular velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
```

```
ylabel('$z_4$','Interpreter','latex');
grid;

sgtitle('Controller on non-linear system');

print -depsc figures/cont_nonlin

% on linear system:

t_cont_lin = out_cont.ycont_lin.Time;

z1_cont_lin = out_cont.ycont_lin.Data(:,1);
z2_cont_lin = out_cont.ycont_lin.Data(:,2);
z3_cont_lin = out_cont.ycont_lin.Data(:,3);
z4_cont_lin = out_cont.ycont_lin.Data(:,4);

figure
subplot(2,2,1);
plot(t_cont_lin, z1_cont_lin);
title("vertical position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_cont_lin, z2_cont_lin);
title("vertical velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_cont_lin, z3_cont_lin);
title("angular position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

subplot(2,2,4);
plot(t_cont_lin, z4_cont_lin);
title("angular velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_4$','Interpreter','latex');
```

```
grid;

sgtitle('Controller on linear system');

print -depsc figures/cont_lin


%% observer

Tfin_obs = 10;       % sim time observer model

O = obsv(A, C);      % observability matrix
rankO = rank(O);     % rank of observability matrix

if rankO == size(A, 1)
disp('The system is observable');
else
disp('The system is NOT observable');
end

desired_poles_obs = [-15, -15, -30, -60];  % desired poles for the observer
L = place(A', C', desired_poles_obs);      % observer gain
L = L';

Aobs = A-L*C;
Bobs = [B L];
Cobs = eye(4);
Dobs = zeros(4, 5);

poles_placed = eig(Aobs);

dz0_obs = [0, 0, 0, 0]';
z0_obs = z0 + dz0_obs;

model_obs = 'booster_observer';

out_obs = sim(model_obs);

%% observer on non-linear system

t_obs = out_obs.yobs.Time;

z1_obs_true = out_obs.yobs.Data(:,1);
z2_obs_true = out_obs.yobs.Data(:,2);
z3_obs_true = out_obs.yobs.Data(:,3);
z4_obs_true = out_obs.yobs.Data(:,4);
```

```
e1_nonlin = out_obs.yobs.Data(:,5);
e2_nonlin = out_obs.yobs.Data(:,6);
e3_nonlin = out_obs.yobs.Data(:,7);
e4_nonlin = out_obs.yobs.Data(:,8);
z1_obs_est = out_obs.yobs.Data(:,9);
z2_obs_est = out_obs.yobs.Data(:,10);
z3_obs_est = out_obs.yobs.Data(:,11);
z4_obs_est = out_obs.yobs.Data(:,12);

% compare position

figure
subplot(2,1,1);
plot(t_obs, z1_obs_true,'r');
hold on;
plot(t_obs, z1_obs_est,'b');
title("Vertical position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
legend('true', 'estimated');
grid;

subplot(2,1,2);
plot(t_obs, z3_obs_true,'r');
hold on;
plot(t_obs, z3_obs_est,'b');
title("Angular position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
legend('true', 'estimated');
grid;

sgtitle('Observer: position non-linear system');

print -depsc figures/obs_pos_nonlin

% compare velocity

figure
subplot(2,1,1);
plot(t_obs, z2_obs_true,'r');
hold on;
plot(t_obs, z2_obs_est,'b');
title("Vertical velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
```

```
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
legend('true', 'estimated');
grid;

subplot(2,1,2);
plot(t_obs, z4_obs_true,'r');
hold on;
plot(t_obs, z4_obs_est,'b');
title("Angular velocity", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_4$','Interpreter','latex');
legend('true', 'estimated');
grid;

sgtitle('Observer: velocity non-linear system');

print -depsc figures/obs_vel_nonlin

%error

figure
subplot(2,2,1);
plot(t_obs, e1_nonlin);
title("Error vertical position", 'Interpreter','latex','FontSize'
    ,14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_obs, e2_nonlin);
title("Error vertical velocity", 'Interpreter','latex','FontSize'
    ,14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_obs, e3_nonlin);
title("Error angular position", 'Interpreter','latex','FontSize',14,
    'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_3$','Interpreter','latex');
grid;

```

```
subplot(2,2,4);
plot(t_obs, e4_nonlin);
title("Error angular velocity", 'Interpreter','latex','FontSize',14,
    'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_4$','Interpreter','latex');
grid;

sgtitle('Observer: error non-linear system');

print -depsc figures/obs_err_nonlin

%% observer on linear system:

t_obs_lin = out_obs.yobs_lin.Time;

z1_obs_true_lin = out_obs.yobs_lin.Data(:,1);
z2_obs_true_lin = out_obs.yobs_lin.Data(:,2);
z3_obs_true_lin = out_obs.yobs_lin.Data(:,3);
z4_obs_true_lin = out_obs.yobs_lin.Data(:,4);
e1_lin = out_obs.yobs_lin.Data(:,5);
e2_lin = out_obs.yobs_lin.Data(:,6);
e3_lin = out_obs.yobs_lin.Data(:,7);
e4_lin = out_obs.yobs_lin.Data(:,8);
z1_obs_est_lin = out_obs.yobs_lin.Data(:,9);
z2_obs_est_lin = out_obs.yobs_lin.Data(:,10);
z3_obs_est_lin = out_obs.yobs_lin.Data(:,11);
z4_obs_est_lin = out_obs.yobs_lin.Data(:,12);

% compare position

figure
subplot(2,1,1);
plot(t_obs_lin, z1_obs_true_lin,'r');
hold on;
plot(t_obs_lin, z1_obs_est_lin,'b');
title("Vertical position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
legend('true', 'estimated');
grid;

subplot(2,1,2);
plot(t_obs_lin, z3_obs_true_lin,'r');
hold on;
plot(t_obs_lin, z3_obs_est_lin,'b');
```

```
title("Angular position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
legend('true', 'estimated');
grid;

sgtitle('Observer: position linear system');

print -depsc figures/obs_pos_lin

% compare velocity

figure
subplot(2,1,1);
plot(t_obs_lin, z2_obs_true_lin,'r');
hold on;
plot(t_obs_lin, z2_obs_est_lin,'b');
title("Vertical velocity", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
legend('true', 'estimated');
grid;

subplot(2,1,2);
plot(t_obs_lin, z4_obs_true_lin,'r');
hold on;
plot(t_obs_lin, z4_obs_est,'b');
title("Angular velocity", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_4$','Interpreter','latex');
legend('true', 'estimated');
grid;

sgtitle('Observer: velocity linear system');

print -depsc figures/obs_vel_lin

%error

figure
subplot(2,2,1);
plot(t_obs_lin, e1_lin);
title("Error vertical position", 'Interpreter','latex','FontSize'
    ,14,'FontWeight','bold');
```

```
xlabel('$t$','Interpreter','latex');
ylabel('$e_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_obs_lin, e2_lin);
title("Error vertical velocity", 'Interpreter','latex','FontSize'
    ,14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_obs_lin, e3_lin);
title("Error angular position", 'Interpreter','latex','FontSize',14,
    'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_3$','Interpreter','latex');
grid;

subplot(2,2,4);
plot(t_obs_lin, e4_lin);
title("Error angular velocity", 'Interpreter','latex','FontSize',14,
    'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$e_4$','Interpreter','latex');
grid;

sgtitle('Observer: error linear system');

print -depsc figures/obs_err_lin


%% Dynamic regulator (compensator)

Tfin_comp = 5;          % sim time compensator model

dz0_comp = [0, 0, theta_tip+10, 0]';
z0_comp = z0 + dz0_comp;

model_comp = 'booster_compensator';

out_comp = sim(model_comp);

% on non-linear system

t_comp = out_comp.ycomp.Time;
```

```

z1_comp = out_comp.ycomp.Data(:,1);
z2_comp = out_comp.ycomp.Data(:,2);
z3_comp = out_comp.ycomp.Data(:,3);
z4_comp = out_comp.ycomp.Data(:,4);

figure
subplot(2,2,1);
plot(t_comp, z1_comp);
title("vertical position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_comp, z2_comp);
title("vertical velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_comp, z3_comp);
title("angular position", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

subplot(2,2,4);
plot(t_comp, z4_comp);
title("angular velocity", 'Interpreter','latex','FontSize',14,'FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_4$','Interpreter','latex');
grid;

sgtitle('Compensator on non-linear system');

print -depsc figures/comp_nonlin

% on linear system:

t_comp_lin = out_comp.ycomp_lin.Time;

```

```matlab
z1_comp_lin = out_comp.ycomp_lin.Data(:,1);
z2_comp_lin = out_comp.ycomp_lin.Data(:,2);
z3_comp_lin = out_comp.ycomp_lin.Data(:,3);
z4_comp_lin = out_comp.ycomp_lin.Data(:,4);

figure
subplot(2,2,1);
plot(t_comp_lin, z1_comp_lin);
title("vertical position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_1$','Interpreter','latex');
grid;

subplot(2,2,2);
plot(t_comp_lin, z2_comp_lin);
title("vertical velocity", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_2$','Interpreter','latex');
grid;

subplot(2,2,3);
plot(t_comp_lin, z3_comp_lin);
title("angular position", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_3$','Interpreter','latex');
grid;

subplot(2,2,4);
plot(t_comp_lin, z4_comp_lin);
title("angular velocity", 'Interpreter','latex','FontSize',14,'
    FontWeight','bold');
xlabel('$t$','Interpreter','latex');
ylabel('$z_4$','Interpreter','latex');
grid;

sgtitle('Compensator on linear system');

print -depsc figures/comp_lin
```

Listing 1: MATLAB code for project

10 References

References

(1) Serway, R. A.; Jewett, J. W.; Peroomian, V., *Physics for scientists and engineers*; Saunders college publishing Philadelphia: 2000; Vol. 2.

(2) Antsaklis, P. J.; Michel, A. N., *Linear systems*; Springer: 1997; Vol. 8.