KTH Royal Institute of Technology

EL2700 - Model Predictive Control

Assignment 1: State Feedback Control Design

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science

Authors:
Jacopo Dallafior,Stefano Tonini

September 1, 2024

Contents

1 Design Task **2**
1.1 Question 1: Implementing the Model 2
1.2 Question 2: Discretization and Comparison 3
1.3 Question 3: Transfer Function Analysis 4
1.3.1 Continuous-Time Model 4
1.3.2 Discrete-Time Model 4
1.4 Question 4: State Feedback Controller Design 5
1.5 Question 5: Integral Action Design 7
1.6 Extra: Two-Axis Dynamics Model 9

Chapter 1

Design Task

In order to prepare our Astrobee's to be deployed in Space, we first need to do a series of ground tests...

1.1 Question 1: Implementing the Model

To implement the model for the one-axis ground dynamics of the Astrobee robot, we define the continuous-time dynamics using the matrices A_c and B_c. The following Python code snippet shows the implementation of the function 'one_axis_ground_dynamics', which populates these matrices:

```
def one_axis_ground_dynamics(self):
    """
    Helper function to populate Ac and Bc with continuous-time
    dynamics of the system.
    """

    # Jacobian of exact discretization
    Ac = np.zeros((2, 2))
    Bc = np.zeros((2, 1))

    # TODO: Complete the entries of the matrices
    #       Ac and Bc. Note that the system mass
    #       is available with self.mass

    Ac = np.array([[0,1],[0,0]])
    Bc = np.array([[0],[1/self.mass]])

    self.Ac = Ac
    self.Bc = Bc

    return self.Ac, self.Bc
```

In this function, the matrices A_c and B_c are populated with the continuous-time dynamics of the system. Specifically, A_c represents the system's state matrix, and B_c represents the input matrix. The mass of the system, which is a combination of the Astrobee and the air-carriage, is used to compute B_c. The function then assigns these matrices to the class variables 'self.Ac' and 'self.Bc' for later use.

1.2 Question 2: Discretization and Comparison

To discretize our system analytically, we used the method that can be seen in the exercise 1.13 (c), with the following formulas:

$$A = e^{A_c h} = I + A_c h + \frac{A_c^2 h^2}{2!} + \frac{A_c^3 h^3}{3!} + \cdots$$

$$B = \int_0^h e^{A_c s} B_c \, ds$$

Below are reported the calculation executeed to find the numerical values:

$$A_c = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ \frac{1}{m_G} \end{bmatrix}$$

$$A = e^{A_c h} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} h + \left(\frac{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} h^2}{2} \right) = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}$$

$$B = \int_0^h e^{A_c s} B_c \, ds = \int_0^h \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{m_G} \end{bmatrix} ds = \int_0^h \begin{bmatrix} \frac{s}{m_G} \\ \frac{1}{m_G} \end{bmatrix} ds = \left[\begin{bmatrix} \frac{s^2}{2m_G} \\ \frac{s}{mg} \end{bmatrix} \right]_0^h = \begin{bmatrix} \frac{h^2}{2m_G} \\ \frac{h}{m_G} \end{bmatrix}$$

Our matrix A is nilpotent (there is a finite t_0 such that $A_t = 0$ for all $t \geq t_0$), so for this reason after t=2 all other terms are zero.
The result obtained, using the given data (h = 0.1 , mg = 20.9) as can be seen from our numerical results are the same that we obtain from the function c2d:

Matrices obtained with function `c2d`:

$$A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.00023923 \\ 0.00478469 \end{bmatrix}$$

Matrices obtained with analytic calculation:

$$A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.000239 \\ 0.004784 \end{bmatrix}$$

1.3 Question 3: Transfer Function Analysis

In this section, we analyze the poles and zeros of both the continuous-time and discrete-time models of the system.

1.3.1 Continuous-Time Model

For the continuous-time model, the system is a double integrator. This is characterized by having two poles with a real part equal to zero. These poles are located at the origin of the complex plane, which indicates that the system has marginal stability in continuous time. The transfer function from the control input $u(t)$ to the output $y(t)$ is given by:

$$H(s) = \frac{1}{m_G s^2}$$

This means the system has no zeros and two poles at the origin. The behavior of the system in the continuous-time domain is predictable with no unexpected dynamics.

1.3.2 Discrete-Time Model

After discretizing the continuous-time model, we use the pulse-transfer function formula to compute the transfer function of the discrete-time model. The pulse-transfer function $G(z)$ for a discrete-time linear system (A, B, C, D) is given by:

$$G(z) = C(zI - A)^{-1}B + D$$

Using this formula, we first compute the transfer function of the discrete-time model. We expected the poles to be located on the unit circle in the z-domain, reflecting the behavior of an integrator in discrete time. Specifically, we anticipated two poles on the unit circle. However, the analysis revealed an additional zero. The pole-zero map for the discrete-time system is shown in Figure 1.1 below.

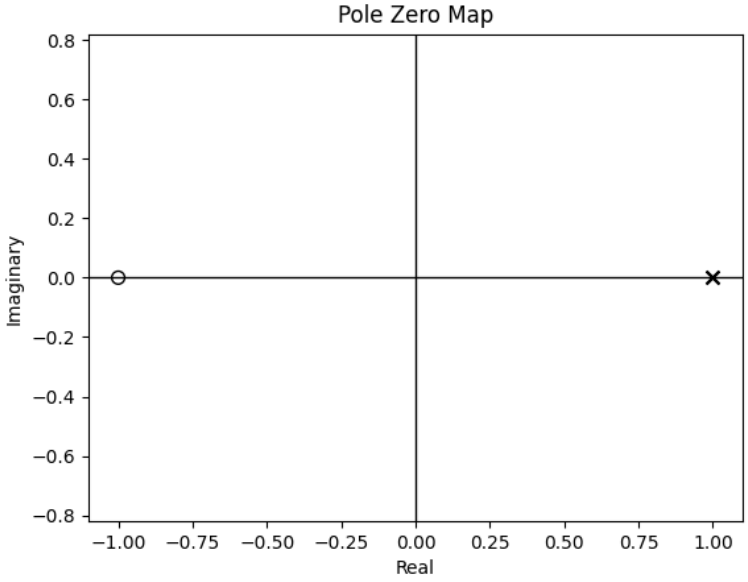

Figure 1.1: Pole-Zero Map of the Discrete-Time System

As shown in the pole-zero map (Figure 1.1), the discrete-time model has one pole located at $z = 1$ as expected, and also includes a zero at $z = -1$. The presence of this zero is a direct consequence of the discretization process, which introduces additional dynamics that were not present in the original continuous-time system.

These findings highlight the differences that can arise when transitioning from continuous to discrete time, particularly the introduction of zeros that were not present in the continuous model. Understanding these differences is crucial for the correct design and implementation of discrete-time controllers.

1.4 Question 4: State Feedback Controller Design

To answer this question we initially considered to put the poles in the middle area of unitary circle following the guideline given in class:

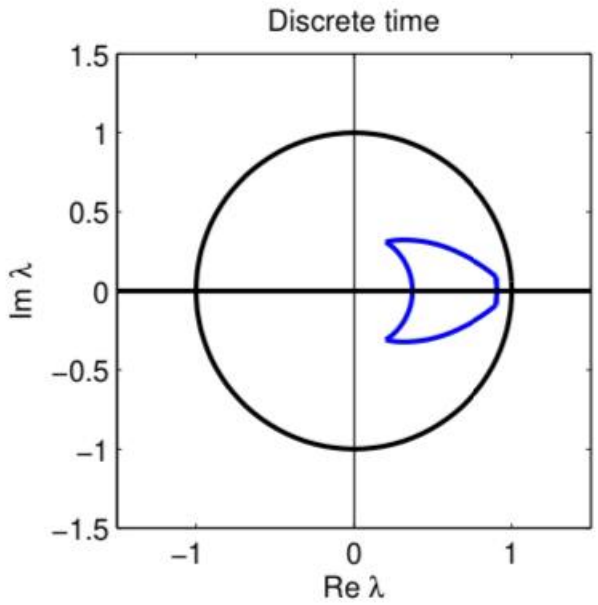

Figure 1.2: Pole-Zero area of the Discrete-Time System

The main problem was that this area change according to the sample time and in our case, with the sampling of 0.1 s, choosing poles in the middle (0.5 and 0.4) results in a incredibly high overshoot of the input force (almost 200 N with the limit of 0.85 N).

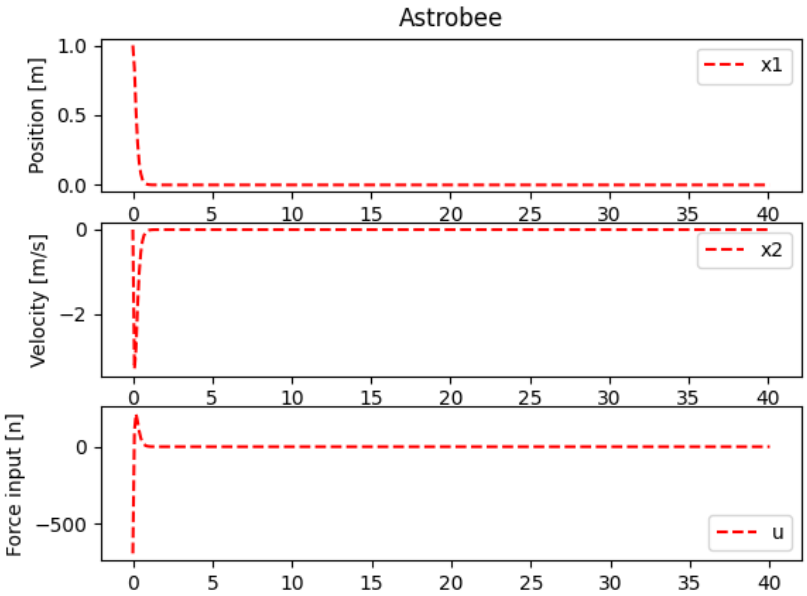

Figure 1.3: Control input and system response choosing 0.4 and 0.45 as poles

In order to choose two poles that can fit our system we decided to put them near to the unitary circle, tuning our values in the order of hundredths.
Our final value is:

$$\begin{bmatrix}\lambda_1 = 0.982 & \lambda_2 = 0.976\end{bmatrix}$$

Once our poles are placed we used the function *get_closed_loop_gain* to find the values of the gain matrix K:

$$K = \begin{bmatrix}0.90288 & 8.732856\end{bmatrix}$$

To implement the closed loop formulation in our system we modified the function *control_law*.

```
def control_law(self, x):
    """
    Nonlinear control law.

    :param x: state
    :type x: np.ndarray
    :param ref: cart reference position, defaults to 10
    :type ref: float, optional
    """

    if self.use_integral is True:
        self.update_integral(x)

    # CONTROL LAW
    u = 0.0

    u = -(self.L @ (x - self.ref ))

    return u
```

After simulating the closed-loop system we obtained the graph showed above, from those graph we were able to tune properly the value of the poles that, as said before, were choseen near the unitary boundary with 0 Immaginary part.
In the results, we can observe that the input consistently remains below 0.85N, and the resulting performance meets the requirement of achieving 90% of the reference within 25 seconds, as requested.

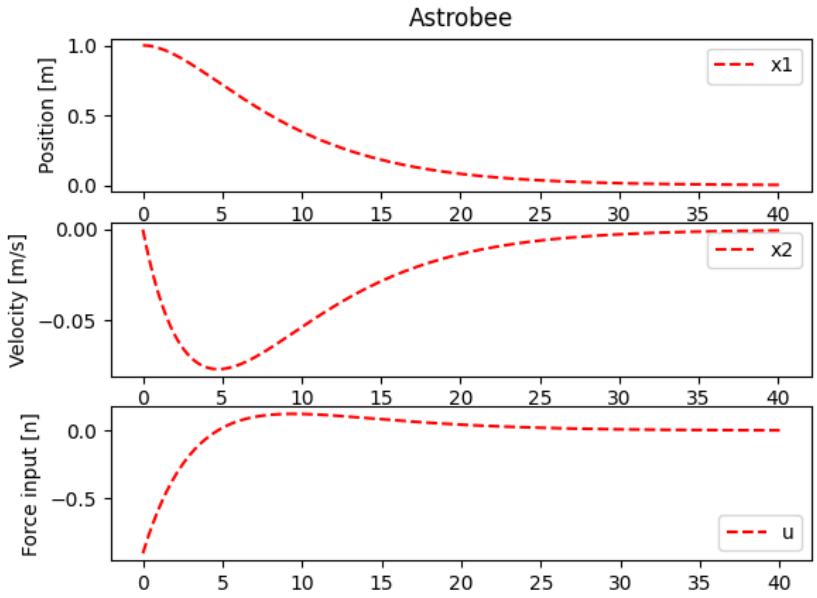

Figure 1.4: Control input and system response with closed loop

1.5 Question 5: Integral Action Design

After introducing a disturbance to the system, we analyzed the control input and the corresponding system response. The first image, shown in Figure 1.5, illustrates the system's response and control input when a disturbance is introduced. As can be seen, the system does not achieve the desired performance due to the stationary error caused by the disturbance.

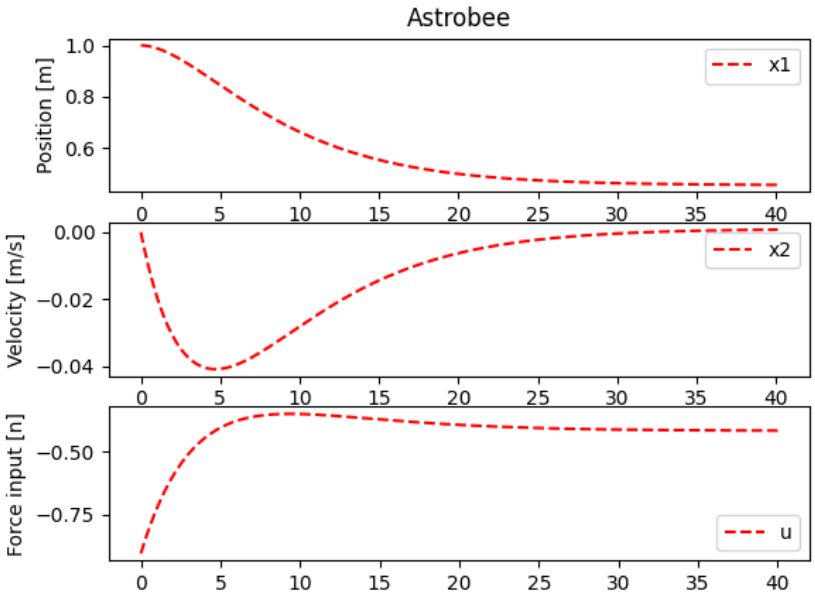

Figure 1.5: Control input and system response with disturbance introduced.

To address this issue, we introduced integral action into our controller. By iteratively tuning the value of K_i, we were able to satisfy the constraints and significantly reduce the steady-state error. The final value after tuning K_i was determined to be $K_i = 0.038$. The second image, shown in Figure 1.6, depicts the system response and control input after adding integral action. The integral action effectively compensates for the disturbance, resulting in improved performance and a reduction in the stationary error.

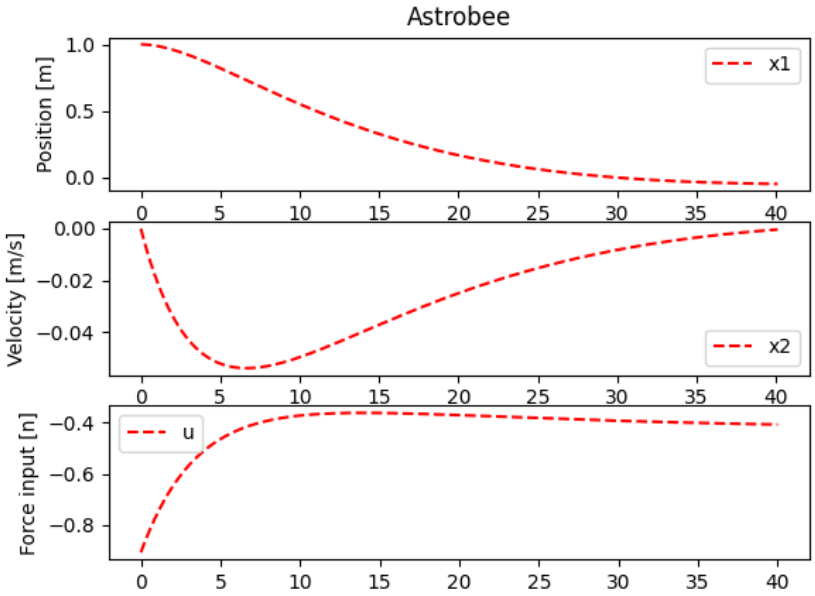

Figure 1.6: Control input and system response after adding integral action.

As demonstrated by the two figures, the introduction of integral action significantly

enhances the system's ability to reach the desired reference point, even in the presence of disturbances. This iterative tuning of K_i was essential to achieving the desired system performance while adhering to the imposed constraints.

1.6 Extra: Two-Axis Dynamics Model

In this extra part of the assignment, we extend the model from a single-axis translation to a two-axis translation. The state variables p_X, v_X, p_Y, and v_Y represent the position and velocity along the X-axis and Y-axis, respectively. The control inputs are forces f_X and f_Y applied along these axes.

The dynamics of the system can be represented using the following state-space model:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

The matrices A, B, C, and D for the two-axis model are defined as follows:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ \frac{1}{m_G} & 0 \\ 0 & 0 \\ 0 & \frac{1}{m_G} \end{bmatrix}$$

Here, m_G represents the combined mass of the Astrobee and the air-carriage.

The dynamics of one axis have not changed significantly from the previous analysis. However, by extending the model to two axes, we can now evaluate the system's behavior in a more realistic scenario. Below are the simulation results for the two-axis dynamics under different conditions.

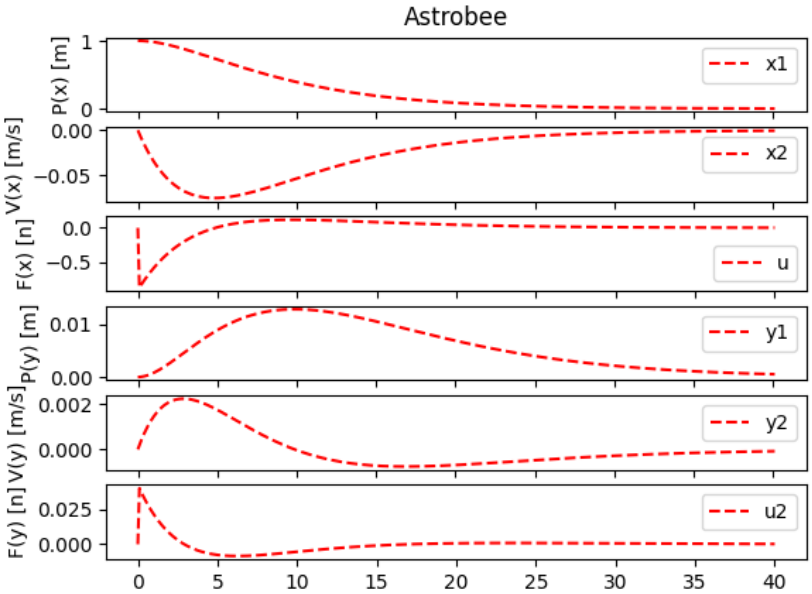

Figure 1.7: Simulation of two-axis dynamics without disturbance.

The Figure 1.7 shows the simulation results for the two-axis dynamics without any disturbance introduced. The dynamics along one axis remain consistent with the previous single-axis analysis. By analyzing the obtained graphs, we can observe that when an input is applied along the y-axis, the movement along this axis is on the order of centimeters, as expected, since the primary motion should occur along the x-axis. Additionally, the force applied along the y-axis is significantly lower than the one applied on the x-axis, further confirming that the displacement along y data is consistent.