Model Predictive Control - EL2700

Assignment 1 : State Feedback Control Design

2023

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Pedro Roque

Introduction

Space robotics is currently receiving a lot of attention from the research community. The need for safe autonomous systems operating in challenging environments is a topic that drives a lot of researchers towards developing efficient control systems, planning strategies, and safe behaviors. In this course, we will be working with a Space Robot that is currently operating on the International Space Station - the NASA Astrobee, see Figure 1. The robot moves by expelling air from two plenum cavities on its sides through vents with controllable aperture, and perceives its environment using onboard RGBD cameras. You can find more information about these robots in NASA's Astrobee website `https://www.nasa.gov/astrobee`.

In this assignment, we will design a discrete-time linear state-feedback controller for our system. To this end, you will compute a discrete-time model which describes the state evolution between sampling instances, and use this model to design a linear state feedback controller which achieves the desired control performance.

Although several of the tasks in this assignment can be carried out in Matlab, you will be working with Python 3 and the CasADi[1] framework. In this way, we can use the same framework for designing both simple linear controllers and advanced nonlinear model-predictive control laws. We encourage you to get familiar with the CasADi framework early in the course - there are a lot of useful examples[2] widely available on the internet, as well as a Cheat Sheet available on Canvas on the Assignment 1 page.

Figure 1: An Astrobee free-flying on the Space Station. The NASA Astrobee's are Guest Science Platforms for experimenting with robots operating in space and testing novel technologies aboard the space station. Ultimately, they will contribute towards autonomous space stations, inhabited and maintained by robotic agents.

[1] Website: `https://web.casadi.org/`

[2] CasADi examples: `https://github.com/casadi/casadi/tree/master/docs/examples/python` and `https://github.com/casadi/casadi/wiki/examples`

Software Preparation

Before proceeding, let's make sure that you have all the software needed for this assignment. We recommend you to use Ubuntu 20.04. Alternatively, a Virtual Machine for VMWare 16 and above is available in Canvas - more information in the Syllabus tab. This virtual machine contains a Ubuntu 20 installation, ready for all the assignments and project. If you wish to install the dependencies on your system, follow these steps:

1. Install Python 3[3] and Pip 3 in your system;
2. Install a Python editor, such as Visual Studio Code, Spyder, or PyCharm. Also feel free to use any other tool of your liking;
3. Install the dependencies by running the `install_deps.py` script with Python 3. Furthermore, for installation of the `slycot` library, please refer to Appendix 1.1.

Design Task

In order to prepare our Astrobee's to be deployed in Space, we first need to do a series of ground tests to make sure that our system is ready to be deployed 400km above Earth. These ground-tests are typically performed at the granite table, where the robot has 3 Degrees of Freedom (DoF) that resemble operations in zero-gravity (remember that inside the space-station there's also atmosphere). Figure 2 shows this testing facility.

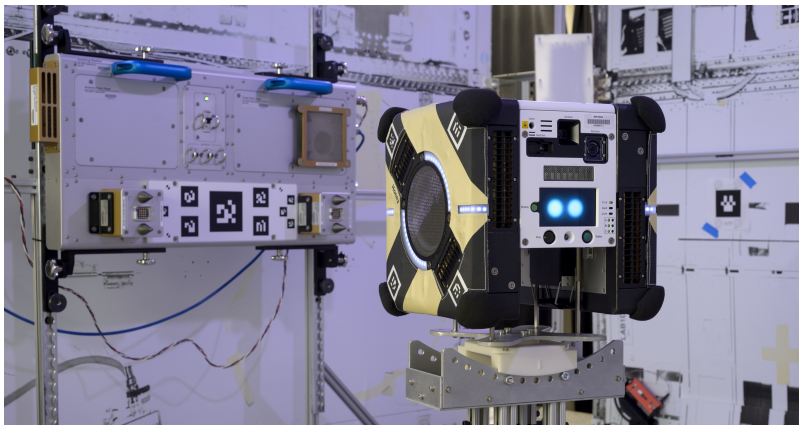

Figure 2: An Astrobee on the granite table testing facility. The robot free-floats on top of an air-carriage. This air carriage transports the robot on three degrees of freedom with no friction with the table. The robot is free to move along two translation axis and one rotation axis. To the left of the robot is a docking bay with two docking ports.

To complete the task, carefully look into the Python files that are part of the assignment and the classes that they implement. The code entry point is `task1.py`, but you will have to complete parts of the `Astrobee` and `Controller` classes.

Introduction to the design task In this assignment we will focus on the docking maneuver with the docking-station, which the robot uses to upload/download data and charge its batteries. Since this is the first task we are running on our freshly built system, we will focus the movement along one axis of the robot, assuming the robot was previously aligned with its docking port.

Design of the state feedback controller For designing the state feedback controller, we will use a linearized model that describes the motion of the Astrobee along one translation axis

$$\dot{\mathbf{x}} = A_c\mathbf{x} + B_c u \tag{1}$$

[3]Python 3.8 or later: `https://www.python.org/downloads/`

where the state $\mathbf{x} = \begin{bmatrix} p_X & v_X \end{bmatrix}^T$ concatenates the position on the X-axis and $u = F_x$ is a force along the body X-axis of the robot. The matrices A_c and B_c are given by

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

where m_G is the mass of the Astrobee and air-carriage, combined. Accordingly:

$$\text{Astrobee mass:} \qquad m_A = 9.6 \text{ [kg]} \tag{2}$$

$$\text{Air-carriage mass:} \qquad m_C = 11.3 \text{ [kg]} \tag{3}$$

$$\text{Combined ground test mass:} \qquad m_G = m_A + m_C = 20.9 \text{ [kg]} \tag{4}$$

Q1: Implement the model in the function `one_axis_ground_dynamics`.

Q2: As you may have noticed, the continuous-time model is a double integrator. You have discretized such a system (with a slightly different B-matrix) in Exercise 1.15 (c). Compare the numerical values returned by `casadi_c2d` with the results of the analytically discretized model - are they the same?

Now, observe that we can write the discrete-time model as

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + Bu_k \qquad y_k = C\mathbf{x}_k + Du_k \tag{5}$$

Q3: Compute the transfer function of the continuous-time model from the control input u to the output y, considering $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $D = 0$. How many poles and zeros does the system have? Where are they located? How many poles and zeros do you expect for the discrete-time model, and where should they be located? Was your intuition right?

Note: For this question, you can use Matlab or any other tool you are comfortable with. In Python, the `control` library provides the same functionality as Matlab (classes `StateSpace`, `TransferFunction` and method `control.pzmap`), but requires `slycot` to be installed - already available on the Virtual Machine. The method `poles_zeros` in the `Astrobee` class implements this functionality. For an installation in your system, please check the Appendix 1.1 for more information.

Q4: Based on the discretized linear model in (5), derive a state feedback controller of the form

$$u_k = -L(\mathbf{x}_k - \mathbf{x}_{ref}) \tag{6}$$

where L is the state feedback gain, designed by placing the poles of the closed loop poles at the desired location using the function `place` in the function `get_closed_loop_gain`. Implement this controller in the function `control_law`. Make sure that the resulting performance is sufficient to achieve 90% of the reference under 25 seconds, with a maximum control input of 0.85N, considering $\mathbf{x}_0 = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}^T$ and $\mathbf{x}_{ref} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}^T$ (in this case, assume that you want to be at least at 0.1[m] in 25 seconds). You should **not overshoot**, or you will hit the dock!

Q5: After several ground tests, we realized that the thrusters on Astrobee do not fully reproduce the desired force input on the system as when they are new, resulting on stationary error. This effect is simulated in the function `set_disturbance`. To fix this, we design an integral action according to

$$u_k = -L\mathbf{x}_k - K_i i_k \qquad i_{k+1} = i_k + h \cdot (p_k - p_{ref}) \tag{7}$$

where p_k is p_{X_k} and p_{ref} a desired position reference. Implement the integral action (7) in the functions `update_integral` and `control_law`, under the `Controller` class. Design the feedback and integral gains so that the Astrobee moves 90% of the distance between the initial position and the reference within 30 seconds. Function `activate_integral_action` can then be used to set an integral gain and system sampling time. **Note:** constraints in Q4 also apply here.

Extra: Extend the model in (5) to a 2-axis translation, considering the state variables p_Y, v_Y as the position and velocity along the Y-axis, and an extra control input force f_Y. Make sure to modify the needed parts in the provided code to accommodate the new model dimensions.

To complete this design project, you should upload a zip file containing the provided code, properly completed. The task grading is based on a successful run of `task1.py`, which should provide all the results. Post all your questions on the Slack workspace `el2700mpc-ht22.slack.com`.

Good Luck!

1 Appendix

1.1 Slycot Installation

Slycot installation is only recommended on Ubuntu systems. To successfully install `slycot`, run the following commands:

```
sudo apt install gfortran
sudo apt install liblapack3 liblapack-dev
sudo apt install libblas3 libblas-dev
pip install slycot
```

If any errors arise, **don't hesitate** to reach Pedro Roque on **Slack** via direct message.