Model Predictive Control - EL2700

Assignment 3 : Linear Quadratic and Gaussian Regulators

2023

Automatic Control
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Erik Berglund and Pedro Roque

Introduction

Dear student, welcome to Space! It is with great pleasure that we welcome you to the orbit testing facilities of the Astrobees. After a series of successful tests on the ground, we are now ready to go to full 6 Degrees-of-Freedom goodness and do further testing in zero gravity!

Since a fundamental functionality for the Astrobees is to be able to dock with the docking station, in this Assignment we will revisit the Assignment 1 setup, but we will improve both the model we are using, as well as the control design. In Figure 1, you can see the docked configuration of the Astrobee.

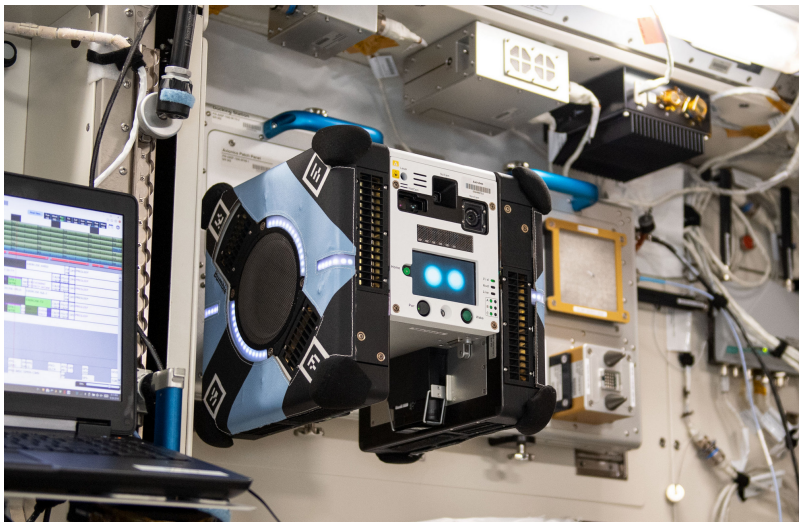

Figure 1: An Astrobee docked on the JEM module of the International Space Station (ISS). The Astrobees are able to charge, upload/download experimental data and communicate with the ground controllers through their docking ports.

The dynamics model of an Astrobee can be written with the Newton-Euler equations

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1a}$$

$$\dot{\mathbf{v}} = \frac{1}{m} R(\boldsymbol{\theta})^T \mathbf{f}, \tag{1b}$$

$$\dot{\boldsymbol{\theta}} = \Theta(\boldsymbol{\theta})\boldsymbol{\omega}, \text{ and} \tag{1c}$$

$$\dot{\boldsymbol{\omega}} = J^{-1}(\mathbf{t} - \boldsymbol{\omega} \times J\boldsymbol{\omega}), \tag{1d}$$

where $\mathbf{p} \in \mathbb{R}^3$ is the position, $\mathbf{v} \in \mathbb{R}^3$ the linear velocity, $\boldsymbol{\theta} \in \mathbb{R}^3$ the euler angles, and $\boldsymbol{\omega} \in \mathbb{R}^3$ the angular velocity in the body-frame. In particular, the euler angles in $\boldsymbol{\theta}$ are ϕ, corresponding to the roll angle, φ corresponding to the pitch angle, and ψ corresponding to the yaw angle, such that $\boldsymbol{\theta} = [\phi, \varphi, \psi]^T$. The rotation matrix $R(\boldsymbol{\theta})$ and attitude jacobian $\Theta(\boldsymbol{\theta})$ are defined as

$$R(\phi,\varphi,\psi) = \begin{bmatrix} c_\psi c_\varphi & c_\psi s_\varphi s_\phi - s_\psi c_\phi & c_\psi s_\varphi c_\phi + s_\psi s_\phi \\ s_\psi c_\varphi & s_\psi s_\varphi s_\phi + c_\psi c_\phi & s_\psi s_\varphi c_\phi - c_\psi s_\phi \\ -s_\varphi & c_\varphi s_\phi & c_\varphi c_\phi \end{bmatrix} \qquad \Theta = \begin{bmatrix} 1 & s_\phi t_\varphi & c_\phi t_\varphi \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi/c_\varphi & c_\phi/c_\varphi \end{bmatrix} \tag{2}$$

In case you are not familiar with Euler angles[1], this website provides an interactive tool for you to visualize them. Lastly, $m \in \mathbb{R}_{>0}$ represents the mass of the Astrobee (no air carriage this time), and $J \in \mathbb{R}^{3\times3}$ its positive definite diagonal inertia matrix. Lastly, the system input is a force $\mathbf{f} \in \mathbb{R}^3$ and torque $\mathbf{t} \in \mathbb{R}^3$.

About now you start to worry - "Ok, what am I supposed to do with this?". Worry not, we will start with a linear version of this system for our LQR, and will leave the nonlinear model for later in the course. We will start by linearized the dynamics in (1) to obtain a model of the format

$$\dot{\mathbf{x}} = A_c\mathbf{x} + B_c\mathbf{u}. \tag{3}$$

[1]Website: `https://danceswithcode.net/engineeringnotes/rotations_in_3d/demo3D/rotations_in_3d_tool.html`

This time, our state $\mathbf{x}$ concatenates $\mathbf{x} = \begin{bmatrix}\mathbf{p}^T & \mathbf{v}^T & \boldsymbol{\theta}^T & \boldsymbol{\omega}^T\end{bmatrix}^T$, while the system input $\mathbf{u}$ concatenates $\mathbf{u} = \begin{bmatrix}\mathbf{f}^T & \mathbf{t}^T\end{bmatrix}^T$. The matrices A_c and B_c are rather large, so we will omit writing them here. You can (and should!) however observe in the `Astrobee` class how they are obtained. As before, we will use Python and CasADi to complete task.

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 or above, but these instructions can also be completed on Windows and Mac as long as you have a working version of Python 3.6 or above. Before proceeding, install the dependencies by running the script `install_deps.py` script with Python 3.

To complete the task, carefully look into the Python files that are part of the assignment and the classes they implement. The code entry point is `task3.py`.

LQR Implementation

In this task, we will implement LQR controller for the reference tracking. The LQR controller is designed based on the discrete-time linearized dynamics

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t \tag{4}$$

where $\mathbf{x}_t \in \mathbb{R}^{12}$ and $\mathbf{u}_t \in \mathbb{R}^6$, being system state and control input respectively. The matrices A and B are provided in the `Astrobee` class, method `casadi_c2d`. The objective is to design an LQR controller that achieves the following criteria:

1. It moves the astrobee from the initial state $x_0 = \mathbf{0}$ to the reference state $\mathbf{p}^{ref} = [1, 0.5, 0.1]^T$, $\mathbf{v}^{ref} = [0, 0, 0]^T$, $\boldsymbol{\theta}^{ref} = [0.087, 0.077, 0.067]^T$ and $\boldsymbol{\omega}^{ref} = [0, 0, 0]^T$;
2. The first three inputs (the forces) may not exceed 0.85 N;
3. The last three inputs (the torques) may not exceed 0.04 Nm;
4. From 12 seconds until the simulation ends (20 seconds) ...
    - ... the distance to the reference must be less than $0.06m$;
    - ... the speed of the astrobee must be less than $0.03m/s$;
    - ... the Euler angles must differ from their targets by less than 10^{-7} radians.
    - ... the position overshoot must be less than 2cm for a safe docking maneuver.

Tracking a reference state $\mathbf{x}^{ref}$ with LQR is done by finding a corresponding equilibrium input u^{ref} such that

$$\mathbf{x}^{ref} = A\mathbf{x}^{ref} + Bu^{ref}. \tag{5}$$

With the incremented state and control variables ($\Delta\mathbf{x}_t = \mathbf{x}_t - \mathbf{x}^{ref}, \Delta u_t = u_t - u^{ref}$), the linear state space model of the system in (4) becomes

$$\Delta\mathbf{x}_{t+1} = A_d\Delta\mathbf{x}_t + B_d\Delta u_t. \tag{6}$$

With the state space model in (6), we can now calculate the optimal gain matrix L, provided by an LQR controller, such that the state feedback controller $u_t = -L\Delta\mathbf{x}_t$ minimizes the infinite horizon quadratic cost function given by

$$J = \sum_{t=0}^{\infty} \Delta\mathbf{x}_t^T Q \Delta\mathbf{x}_t + \Delta u_t^T R \Delta u_t$$

where $Q \succeq 0$ and $R \succ 0$ are symmetric and positive (semi-)definite matrices of appropriate dimensions. These matrices will be used to calculate your LQR feedback gain L. With stabilizing state feedback gain L, we can now find a u^{ref} in (5) of the form

$$u^{ref} = -L\mathbf{x}^{ref}$$

The optimal control to be applied to the original system can be calculated as

$$\begin{aligned} u_t &= -L\Delta\mathbf{x}_t + u^{ref} \\ &= -L\mathbf{x}_t. \end{aligned}$$

Q1: If a positive definite matrix Q is used in the design of an LQR, the system must be stabilizable for there to be a solution to the infinite horizon LQR problem with finite cost. Briefly explain why, and check numerically whether the linearized model of the astrobee is stabilizable *Hint:* use `numpy.linalg.eig()` to compute left eigenvectors of A_d.

Q2: To investigate what effect the tuning parameters have on the performance of the controller, we will now make a few experiments. Starting with `Q = np.eye(12)` and `R = np.eye(6)`, make the following changes to the matrices, one at a time:

- Multiply R by 10

- Add 100 to the velocity components of the diagonal of Q, `Q[3:6]` and `Q[9:]`
- Revert the velocity components to their initial value of 1 and add 100 to the position and attitude components of the diagonal of Q, `Q[0:3]` and `Q[6:9]`
- Increase all elements of Q by 100.

For each case, briefly describe and explain what happens.

Q3: Tune an LQR controller that satisfies the criteria stated in points 1-4.

LQG Implementation

We will now implement an LQG controller by combining the LQR controller with a Kalman filter. The only formal requirement for state estimation is that the system is observable. For the astrobee system, we now assume that we can't measure the velocity directly. Instead, we will estimate it by noisy measurements of the position. As for the attitude and angular velocity, we assume that they can be estimated perfectly. The output matrix for the position-velocity subsystem then becomes:

$$C_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{7}$$

We will use a Kalman filter provided by the *filterpy* library. The inputs to this block are:(i) State space model of the system with process noise (w_t) and measurement noise (v_t)

$$\begin{aligned} \mathbf{x}_{t+1} &= A_d \mathbf{x}_t + \begin{bmatrix} B_u & B_w \end{bmatrix} \begin{bmatrix} u_t \\ w_t \end{bmatrix} \\ y_t &= C_d \mathbf{x}_t + \begin{bmatrix} D_u & D_w \end{bmatrix} \begin{bmatrix} u_t \\ w_t \end{bmatrix} + v_t \end{aligned} \tag{8}$$

(ii) Initial guess of the state ($\mathbf{x}_0$); (iii) Process and measurement noise covariance matrices (Q_n , R_n). With the estimated state ($\hat{\mathbf{x}}_t$), we can formulate output feedback controller as:

$$u_t = -L\hat{\mathbf{x}}_t \tag{9}$$

Q4: Briefly explain what you expect the performance of the controller to be for different values of Q_n and R_n. You can manually adjust the measurement noise on line 61 of `simulation.py` to make the changes to the noise covariance matrices more noticeable.

Q5: In the previous part, the velocity was estimated from noisy position measurements. However, there was no noise in the process itself. In this part, we investigate what happens when this is the case. In the Astrobee code, line 258, uncomment the line `self.kf_activated = True` . Then run a few different simulations just as in the previous part and comment on the performance of the controller in the presence of process noise. You may also adjust the process noise on line 210 of the `Astrobee` class.

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 `task3.py`, which should provide all the results. Post all your questions on the Slack workspace `el2700mpc-ht22.slack.com`.

Good Luck!