ME 586: Biology-Inspired Robotics

University of Washington, Winter 2022, Prof. Sawyer B. Fuller

Problem Set 2*(revised 2022.01.28)*

Goals: familiarize you with basic control of 2D flight dynamics and control including the optimal "LQR" Linear Quadratic Regulator.

Reading: Optimization-Based Control by Richard M. Murray, Chapter 3, section 3.5 and 3.6: Linear Quadratic Regulators (the beginning of Chapter 3 provides a derivation).

1. **PD control of flight dynamics in 2D**
The idea of this exercise is to become familiar with controlling the 2D flight dynamics of a hovering aircraft like the crazyflie helicopter or flying insect. Skeleton code that simulates the dynamics of the robot has been implemented in a Jupyter notebook on the software_examples folder on the course website called `me586_2d_crazyflie_simulator_PD.ipynb`. It implements a planar, 2D simulation of the full 3D dynamics of a hovering aircraft, which follow the Newton-Euler equations of motion

$$\begin{aligned}\Sigma \boldsymbol{f} &= m\dot{\boldsymbol{v}}\\ \Sigma \boldsymbol{\tau}' &= \mathbf{J}\dot{\boldsymbol{\omega}}' + \boldsymbol{\omega}' \times \mathbf{J}\boldsymbol{\omega}'\\ \dot{\mathbf{R}} &= \mathbf{R}\boldsymbol{\omega}'^{\times}\\ \dot{\boldsymbol{p}} &= \boldsymbol{v}\end{aligned}$$

where $\boldsymbol{f}$ is any force acting on the vehicle, $\boldsymbol{\tau}'$ is any torque acting on the vehicle, $\boldsymbol{v}$ is the velocity and $\boldsymbol{p}$ is the position of the center of mass of the robot in world coordinates, $\boldsymbol{\omega}'$ is the angular velocity of the robot, $\mathbf{J}$ is the vehicle's moment of inertia, and $\mathbf{R}$ is the rotation matrix that relates vectors given in body-attached coordinates to world coordinates. For any vector $\boldsymbol{v}$ expressed in world coordinates, $\boldsymbol{v}'$ is the same vector expressed in body-attached coordinates. They are related by $\boldsymbol{v} = \mathbf{R}\boldsymbol{v}'$. (Using the special property of rotation matrices that $\mathbf{R}^{-1} = \mathbf{R}^T$, we can also go in the other direction: $\boldsymbol{v}' = \mathbf{R}^T\boldsymbol{v}$). The quantity $\boldsymbol{\omega}'^{\times}$ is a 3×3 matrix that performs the cross product operation $\boldsymbol{\omega}'\times$.

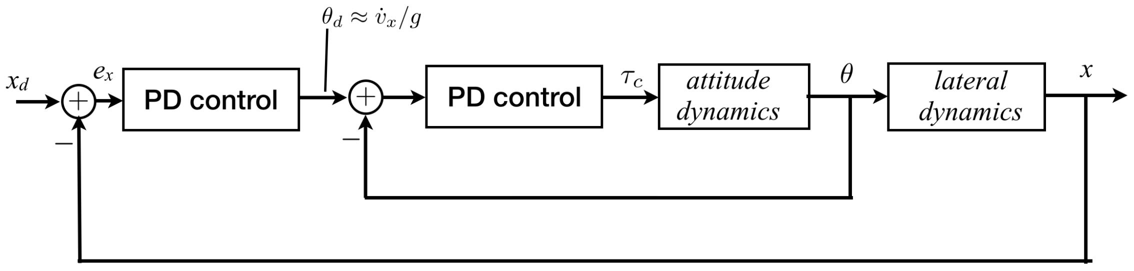

Figure 1: PD control of lateral dynamics using nested loops: a fast inner loop regulates attitude; a slower outer loop regulates lateral position. A separate loop (not shown) regulates altitude.

In the simulation these equations are considered in a constrained, two-dimensional plane for simplicity. The resuling equations are given by

$$\begin{aligned}\Sigma \boldsymbol{f} &= \begin{bmatrix}0\\-mg\end{bmatrix} + \mathbf{R}\begin{bmatrix}0\\f_z\end{bmatrix} - b\mathbf{R}\boldsymbol{v} = m\dot{\boldsymbol{v}}\\ \Sigma \boldsymbol{\tau}_y &= \tau_y = (\mathbf{J}\dot{\boldsymbol{\omega}}' + \boldsymbol{\omega}' \times \mathbf{J}\boldsymbol{\omega}')_y\\ \dot{\theta}_y &= \omega_y\\ \dot{\boldsymbol{p}} &= \boldsymbol{v}\\ \mathbf{R} &= \begin{bmatrix}\cos\theta_y & \sin\theta_y\\ -\sin\theta_y & \cos\theta_y\end{bmatrix}\end{aligned}$$

where the inputs are $\boldsymbol{u} = [f_z,\, \tau_y]^T$, the thrust force from the rotors and torque on the aircraft from the rotors/wings, respectively. Here, $\boldsymbol{v} = [\dot{x},\, \dot{z}]^T$, $\boldsymbol{\omega}' = [0,\, \omega_y',\, 0]^T$, $\boldsymbol{p} = [x,\, z]^T$, and b is an aerodynamic drag coefficient.

In `me586_2d_crazyflie_simulator_PD.ipynb`, you are to implement one type of control in the form of nested PD (proportional-derivative) control loops (Figure 1 above). In this notebook, the input is defined as the deviation from equilibrium, that is, $\tilde{\boldsymbol{u}} = [f_z - mg,\, \tau_y]^T$. In the cell entitled "controllers" hand-tune K_p and K_d (proportional and derivative) gains, respectively, for each controller and submit a notebook showing the trajectory of your tuned controller.

Remarks:

- In each of the three PD controller functions, `attitude_controller`, `lateral_controller`, and `altitude_controller`, the states you will need to implement that controller have already been extracted from the state vector (e.g. x-position and x-velocity for the lateral controller). Typically, a PD controller acts on the position (or angle) *error* but only acts on the *absolute velocity* according to $u = K_p(x_d - x) - K_d\dot{x}$ where x is the variable you are controlling.
- A reasonable strategy is to start by finding the gains of the inner-most loop, the attitude loop. Adjust its gains until you have good attitude performance, leaving the other controllers to output zero. For each controller, first find a proportional gain that isn't so high that you see erratic behavior like tumbling, but isn't so low that it doesn't respond. A good value results in oscillations around your desired position (you can think of proportional control as being "spring-like").
- Then, to damp out the oscillations, add a derivative term, whose purpose is to add damping. Repeat for your altitude and lateral controllers.

2. **Linearization of flight dynamics**
Now we will create a linear state-space formulation of the dynamics that we can use for LQR controller design.

   (a) Consider a reduced system in which the position is not part of the state, that is, $\boldsymbol{q} = [\theta_y,\, \omega_y,\, \dot{x},\, \dot{z}]^T$. Express these two-dimensional planar dynamics as a nonlinear differential equation in the form $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$, where $\boldsymbol{u} = [f_z,\, \tau_y]^T$. The term $(\mathbf{J}\dot{\boldsymbol{\omega}}' + \boldsymbol{\omega}' \times \mathbf{J}\boldsymbol{\omega}')_y$ is the y-component of the quantity in the parentheses. Please write out your answer component-wise, so that on each line of your answer you explicitly state how each state is changing with time, e.g. $\dot{\theta}_y = ...$; $\dot{\omega}_y = ...$ etc. in the original order of states.
   Tip: you can write Latex-style formulas in a Jupyter notebook "markdown" cell (i.e. a text cell) by surrounding your equation with \$s as follows: \$ <equation> \$.

   (b) Assuming that thrust perfectly balances out gravity ($f_z = mg$) and that the inertia matrix is diagonal ($\mathbf{J} = diag[J_{xx},\, J_{yy},\, J_{zz}]$), find the equilibrium point ($\boldsymbol{q}_e$, $\boldsymbol{u}_e$) corresponding to upright hover for the reduced dynamics. Then, linearize the dynamics about that point and express the system as a state-space linear system $\dot{\tilde{\boldsymbol{q}}} = A\tilde{\boldsymbol{q}} + B\tilde{\boldsymbol{u}}$, where $\tilde{\boldsymbol{q}} = \boldsymbol{q} - \boldsymbol{q}_e$ and $\tilde{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_e$.

   (c) Now consider the full 2D system that includes position, parameterized by the state vector $\boldsymbol{q} = [\theta_y,\, \omega_y,\, x,\, \dot{x},\, z,\, \dot{z}]^T$. This system has an infinitude of equilibria, one for all possible x and z positions. The linearization for the full 2D system for all of these positions is identical; show that it is given by

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ g & 0 & 0 & -b/m & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -b/m \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 1/J_{yy} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1/m & 0 \end{bmatrix}$$

3. **LQR control**
In a Linear Quadratic Regulator, the entire state is used for feedback. We will use the linearization derived above for control. In the subsequent problem set, we will use it for state estimation. An LQR regulator has the form of a state feedback controller

$$\boldsymbol{u} = -K\boldsymbol{q},$$

where $\boldsymbol{u}$ is the control input that minimizes the quadratic cost

$$J = \int_0^\infty (\boldsymbol{q}^T Q \boldsymbol{q} + \boldsymbol{u}^T R \boldsymbol{u}) dt.$$

Note that $\boldsymbol{u}$ here refers to deviation from equilibrium (which was $\tilde{\boldsymbol{u}}$ in previous problems). We will use an LQR controller to stabilize the full, nonlinear dynamics of the system. A python notebook is provided, `me586_2d_crazyflie_simulator_LQR.ipynb` that has skeleton code you can use for analysis and simulation.

   (a) Using the Python notebook provided, add a cell that shows that the full linearized system (question 2c above) can be stabilized by performing the reachability (also known as "controllability") test.

   (b) Finish implementation of the LQR controller by filling in missing elements (denoted anywhere there is a comment with three `###`'s.)

   (c) Now explore how changing weights affects the behavior of the system, increasing or decreasing weights until you get reasonable performance. Typically we know control effort costs and leave those fixed—they are the amount of energy the motors consume—and our state costs are the "knob" we tune. (Hint: start by incrementing weights by factors of 10 until you start seeing changes). Provide a notebook with the plots that show the following effects (you may need to copy and paste code to run simulations with different LQR weights) and briefly (in a sentence or two) explain:

      i. What happens when your x weight is much higher than your z-weight?

      ii. What happens when you have a high velocity weight relative to position weight?