Quad-rotor P&S

Exercise 4: PID and LQR Implementation Continued

Learning objectives relevant for this exercise sheet:

v) Experienced the challenges of tuning a PID and LQR controller for achieving stable hover of a quad-rotor, both in simulation and on the real-world system,

vi) Be able to write C++ code for implementing a PID and LQR controller.

The following steps are a brief outline for continuing from the PID implementation you started in the previous exercise class, and then implement an LQR controller.

A) PID Control for altitude and yaw

Continue with the respective tasks from the previous exercise sheet. The goal of this task is to implement a PID controller for altitude and yaw (i.e., for α and z) and tune the P, I, and D gains accordingly.

- For tuning the P, I, and D gains you can consider using either: (1) the Ziegler-Nichols oscillation method for getting an initial guess, (2) the gains you find using simulation, or (3) directly tune that gains on the real system (slowly increase P until oscillation starts, then slowly add a small amount of D to dampen the oscillation, and then slowly add a small amount of I to correct any steady state errors).

B) LQR Control for (x, y) position

Recall that the linearised equations of motion indicate a complete decoupling between x position and pitch angle from y position and roll angle, i.e.,

$$\begin{bmatrix} \dot{p}_x \\ \ddot{p}_x \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & g \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ \dot{p}_x \\ \beta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \omega_y \end{bmatrix}, \qquad \begin{bmatrix} \dot{p}_y \\ \ddot{p}_y \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -g \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_y \\ \dot{p}_y \\ \gamma \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \omega_x \end{bmatrix},$$

Thus we can use the pitch rate ω_y to control a body frame x position error, and the roll rate ω_x to control a body frame y position error.

The goal of this task is to design an LQR controller for each of the x and y separately.

- The LQR controller synthesis method is briefly described in the additional material at the end of this exercise sheet.
- Can you see a link between the gains of the LQR state-feedback controller and the P and D gains of a PID controller? What property/feature of the dynamics and state vector allow you to see this link?

C) Add an integrator for the (x, y) position

Once you have completed the implementation of your LQR controller for (x, y) position from the previous task, observe the steady state offset in the tracking of an (x, y) setpoint. Try adjusting the position of the battery by a few millimeters and observe the steady state offset again. Does the position of the battery noticeably change the steady state offset? How does the sensitivity compared with what you simulated previously?

The goal of this task is to add an integrator to each of the x and y position controllers to remove the steady state offset.

- The LQR controllers you designed in the previous task use position and angle as the state and compute a angular rate in the body frame as the input to be applied by the Crazyflie. What type of offset are you correcting for if your integrator also computes an angular rate adjustment?
- Thinking about the test of adjusting the battery position, what type of model error should we correct for by adding an integrator on the x and y position errors? Recalling the body frame torque actuators introduced in the script, namely $\tau_x^{(\mathrm{B})}$ and $\tau_y^{(\mathrm{B})}$, implement an integrator that compensates for the steady state error by adjusting the motor commands directly.
- It is recommended to connect your code to the buttons in the GUI for switching quickly between the three options that you implement:
  (1) **No integrator**
  (2) Integrator that applies **angular rates** in the body frame to correct the steady-state offset
  (3) Integrator that applies **torques** in the body frame to correct the steady-state offset
- To connect your code to the GUI buttons, in the file `StudentControllerService.cpp` search for the function `customCommandReceivedCallback`, and there you see a switch case that allows you to trigger different code depending on which button was pressed. It is recommended that you use one button for each of the following:
  (1) One button to toggle the "angular rates integrator" on and off
  (2) One button to toggle the "torques integrator" on and off
  (3) One button to reset both integrators.

Pre-class task for next session

D) Custom Project

For the final experiment-based session you will be given the freedom to implement your own ideas using your implemented (x, y, z, α) tracking controller as a basis. Take some time to brainstorm with your partner a goal that you would like to achieve, some guidance and group brainstorming will be conducted during the class.

A key element for maximising the utilisation of your time in the final experiment session is to prepare the required code in the time between now and the final experiment session.

Additional info - LQR Controller Synthesis:

As a reminder, the acronym **LQR** stands for **L**inear **Q**uadratic **R**egulator. The name itself specifies the setting to which this controller design method applies:

- the **dynamics** of the system are **linear**,
- the **cost function** to be minimized is **quadratic**,
- the **controller** sythesised will **regulate** the states **to zero**.

The following information summarises the steps and equations for synthesising an infinite-horizon LQR controller. The resulting controller is termed a "linear state-feedback controller", and commonly denoted by a matrix "K". The state-feedback matrix K has one row for each input to the plant, and one column for each state of the plant.

- The continuous-time and discrete-time linear system dynamics are denoted respectively as,

$$\dot{x} = Ax + Bu\,, \qquad\qquad x_{k+1} = A_{\mathrm{D}} x_k + B_{\mathrm{D}} u_k\,.$$

  To convert the continuous-time system matrices, A and B, to discrete-time system matrices, A_{D} and B_{D}, use the MATLAB function `c2d`, specifying *zero order hold* as the discretisation method.

- You will need to choose the Q and R **cost function** matrices to achieve the desired flight performance of your vehicle. The infinite-horizon LQR cost function is:

$$J_\infty \;=\; \int_0^\infty \Big(x(t)^\top\, Q\, x(t) \;+\; u(t)^\top\, R\, u(t) \Big)\, dt$$

- For choosing the Q and R cost function matrices you should think about the units of the different elements of the state vector. For example, perhaps you wish to penalise a 10 centimeter deviation in x position the same amount as a 5 degree deviation in yaw.
- Use the MATLAB functions `care` and `dare` to compute solutions for the continuous and discrete time algebraic Riccati equation respectively. Read the help documentation to be sure you understand what these functions compute.
- The infinite-horizon LQR design equations for a **continuous-time** linear-time-invariant (LTI) system are (taken directly from the Control Systems I lecture notes):

$$0 \;=\; -\,A^\top P_\infty \;-\; P_\infty A \;+\; P_\infty B R^{-1} B^\top P_\infty \;-\; Q \tag{1a}$$

$$u(t) \;=\; -\,K_\infty\, x(t) \tag{1b}$$

$$K_\infty \;=\; R^{-1} B^\top P_\infty \tag{1c}$$

  where (1a) is the Riccati equation to solve for P_∞, (1b) is the control law to implement, and (1c) is the state-feedback gain matrix.

- The infinite-horizon LQR design equations for a **discrete-time** LTI system are:

$$0 \;=\; -\,P_{\infty,\mathrm{D}} \;+\; A_{\mathrm{D}}^\top P_{\mathrm{D},\infty} A_{\mathrm{D}} \;-\; A_{\mathrm{D}}^\top P_{\mathrm{D},\infty} B_{\mathrm{D}} \left(B_{\mathrm{D}}^\top P_{\mathrm{D},\infty} B_{\mathrm{D}} + R \right)^{-1} B_{\mathrm{D}}^\top P_{\mathrm{D},\infty} A_{\mathrm{D}} \;+\; Q \tag{2a}$$

$$u_{\mathrm{D}}(k) \;=\; -\,K_{\mathrm{D},\infty}\, x_{\mathrm{D}}(k) \tag{2b}$$

$$K_{\mathrm{D},\infty} \;=\; \left(B_{\mathrm{D}}^\top P_{\mathrm{D},\infty} B_{\mathrm{D}} + R \right)^{-1} B_{\mathrm{D}}^\top P_{\mathrm{D},\infty} A_{\mathrm{D}} \tag{2c}$$

  where the subscript $(\cdot)_{\mathrm{D}}$ indicates that the quantities are for a discrete-time system.

- Both the `care` and `dare` function can return a state-feedback gain matrix, be careful of the sign convention if you use this matrix.

Additional info - PID Control for (x, y) position:

If you wish to also implement a PID controller for (x, y) position, this is the relevant task description.

Based on the learnings and outcomes from implementing a PID controller for altitude and yaw: design, implement and, tune a PID controller for controlling the x and y position.

Recall that the linearised equations of motion, provided in the "additional info" of exercise sheet 2, indicate a complete decoupling between x position and pitch angle from y position and roll angle, i.e.,

$$\begin{bmatrix} \dot{p}_x \\ \ddot{p}_x \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & g \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ \dot{p}_x \\ \beta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \omega_y \end{bmatrix}, \qquad \begin{bmatrix} \dot{p}_y \\ \ddot{p}_y \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -g \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_y \\ \dot{p}_y \\ \gamma \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \omega_x \end{bmatrix},$$

Thus we can use the pitch rate ω_y to control a body frame x position error, and the roll rate ω_x to control a body frame y position error.

We again have that the dynamics of the pitch (or roll) angle is sufficiently faster than the dynamics of the x (or y) position, and thus we can also reasonably control the position error with a nested controller. The "outer control" loop takes an x position error and requests a pitch, β, angle to correct for this, then the "inner loop" takes the error to this requested pitch, β, angle and requests an angular rate about the body frame y-axis, ω_y, to correct for this. This architecture is shown in Figure 1,

- Before testing out your chosen P, I, and D gains on the real system, you should create this architecture in your simulation and see what sign conventions are required and to get an indication of the controller gains and behaviour.

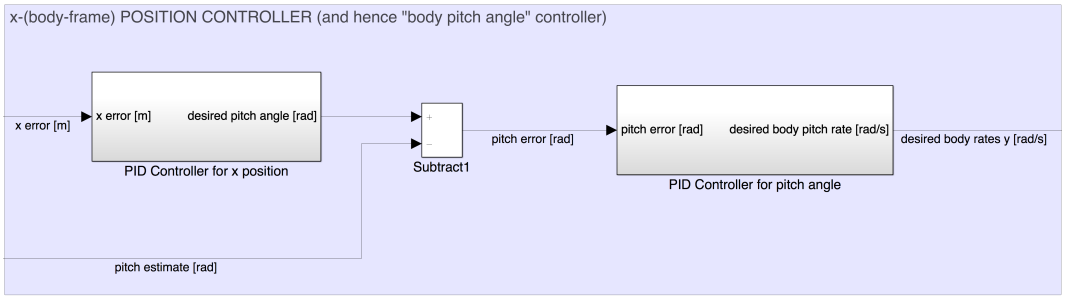

Figure 1: Showing the explained nested architecture for controlling the body frame x position error through the actuator of angular rate about the body frame y-axis ω_y.