Quadrotor Controller Design Methodologies

Ethan LoCicero

January 5, 2024

1 Introduction

In this document, two controller design methodologies are laid out: linear quadratic regulator (LQR) and proportional-derivative (PD). Both (but mostly LQR) rely on a dynamic model of the drone. This model is based on the report by Lukkonnen and is simulated in MATLAB in UAVControllerPy/ControlDesign/Main.m and accompanying functions (especially NonlinearStatespace.m). The estimation procedure for most of the relevant parameters (mass, intertia, length, etc) are fairly straightforward. Motor and rotor parameters are more difficult to estimate, so their estimation procedure is outlined in the final section. All papers referenced herein are in the same folder as this document.

2 LQR Controller

The LQR controller solves the optimization problem

$$\min_{\delta u} \int_0^\infty \delta x^T Q \delta x + \delta u^T R \delta u dt \tag{1}$$

where t is time, $Q \geq 0$ and $R > 0$ are tune-able matrices that weight the designer's preferences, $\delta x = x - x_e$, where x is the state vector and x_e is the equilibrium state or state setpoint, and $\delta u = u - u_e$, where u is the input vector, and u_e is the equilibrium input or input setpoint. When δx is small, then the state is close to the desired setpoint (which means good performance and low flight error), and when δu is small, the input is close to the desired setpoint (which means a low energy demand from the controller). Furthermore, when Q is large, the penalty to state deviations is large, so the controller will drive the state to equilibrium faster. Likewise, when R is large, the penalty to the input is large, so the controller will work less hard and drive the state slower. These parameters can be chosen to shape the system response appropriately.

Equation 1 is difficult to solve for a general nonlinear system $\dot{x} = f(x, u)$, but it is simple to solve for a linearized system, $\dot{x} = Ax + Bu$. Given the dynamics matrices A and B and the chosen weight matrices Q and R, the controller is $\delta u = -K\delta x$, or equivalently,

$$u = u_e - K(x - x_e),$$

where K can be found with the MATLAB function $K = lqr(A, B, Q, R)$ (this solves an algebraic Riccati equation, but that's outside the scope of this document). This control design is implemented around likes 23-32 of the MATLAB code UAVControllerPy/ControlDesign/Main.m. For drone flight, it is implemented in Feedback.py under the function LQR.

Note that this controller can be tricky to tune, and can result in response patterns that are not as smooth as some other methods, like PD control. More importantly, when the states are not known exactly but are measured with some filter, the LQR design can easily become unstable.

3 PD

Proportional-derivative control is one of the simplest types of control when restricted to single-input-single-output systems. It simply has the form $\delta u = -K_P \delta x - K_D \delta \dot{x}$ where $\delta \dot{x} = \dot{x} - \dot{x}_e$, where $\dot{x}$ is the time rate of change of the state, $\dot{x}_e$ is the equilibrium value or setpoint, and K_P and K_D are tune-able parameters. Loosely, this says if x is less than it should be, apply a proportional positive force and vice versa, and if x is traveling too fast, reduce that force proportionally and vice versa. If x and u are scalars, then there are only two parameters to tune, which is easily done with Zeigler-Nichol's tuning method (increase K_P until boardline unstable, then increase K_D until sufficiently damped).

However, for multiple-input-multiple-output systems, it is much more challenging to tune a PD controller. To deal with this, we'll follow Mahony, Kumar, and Corke's method, which decouples the problem.

First, they provide PDs controller to set the commanded rate of change in the x and y directions:

$$\ddot{x}_{com} = K_{P,x}\delta x + K_{D,x}\delta \dot{x}$$
$$\ddot{y}_{com} = K_{P,y}\delta y + K_{D,y}\delta \dot{y}$$

These equations say that the further away the drone is from the desired xy position, the faster we will try to move towards that position, but if we're moving too fast, we'll slow down.

Next, we note that the acceleration in the xy plane depends on the roll and pitch angles, so we'll use the commanded xy accelerations to define the roll and pitch angle setpoints.

$$\phi_e = (\ddot{x}_{com} \sin\psi_e - \ddot{y}_{com} \cos\psi_e)/g$$
$$\theta_e = (\ddot{x}_{com} \cos\psi_e + \ddot{y}_{com} \sin\psi_e)/g$$

where g is gravity, and ψ_e is the yaw setpoint which is given in the initialization. Now that the desired roll pitch and yaw are known, we can design a PD controller for the attitude and z position. The roll depends only on the torque about the roll axis, and likewise for pitch and yaw, so these are independent equations:

$$\tau_\phi = -K_{P,\phi}\delta\phi - K_{D,\phi}\delta\dot{\phi}$$
$$\tau_\theta = -K_{P,\theta}\delta\theta - K_{D,\theta}\delta\dot{\theta}$$
$$\tau_\psi = -K_{P,\psi}\delta\psi - K_{D,\psi}\delta\dot{\psi}$$

The z position depends on the total lift of the motors minus gravity, so the PD controller for z requires an offset term to compensate for the weight. This results in:

$$F = mg - K_{P,z}\delta z - K_{D,z}\delta \dot{z}$$

We need to convert the lift F and torques tau_i into rotor speed. Creating the vector

$$\xi = \begin{bmatrix} L \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix},$$

the rotor speeds that generate the approprate torque and lift can be calculated as

$$\omega_{com} = \sqrt{\Gamma \xi}$$

where

$$\Gamma = \begin{bmatrix} k & k & k & k \\ 0 & -lk & 0 & lk \\ -lk & 0 & lk & 0 \\ b & -b & b & -b \end{bmatrix}^{-1}$$

is the matrix that satisfies $\xi = \Gamma^{-1}\omega_{com}^2$, k is the lift coefficient, b is the drag coefficient, and l is the distance from center of mass to center of rotor (this is just stacking together the equations for rotor speed to lift and torque). Lastly, rotor speed must be converted to pusewidth, which is done with

$$PW = V_{ff}(\omega_{com}) + K_{P,m}(\omega_{com} - \omega_{meas})$$

where V_{ff} is a nonlinear function of motor parameters defined in the next section, and the second term is a proportional controller that is meant to compensate for errors in the commanded and actual motor speed. However, we currently have no way of measuring the actual rotor speed in real time, so this term is omitted.

4 Rotor and Motor System Identification

First and simplest, from Luukkonen, $F = k\omega^2$. So the lift coefficient is estimated by strapping the motor to a scale, measuring the weight difference at several motor speeds, and then applying a linear regression to F vs ω^2, constrained to the origin.

Next, the motor dynamics are given in Bouabdallah2004 and Luukkonen as

$$\begin{aligned} v &= Ri + L\frac{di}{dt} + k_e\omega \\ J\dot{\omega} &= \tau_{motor} - \tau_{drag} \\ \tau_{motor} &= k_T i \\ \tau_{drag} &= b\omega^2 \end{aligned}$$

where i current, v voltage delivered to motors, ω motor speed, R resistance, L inductance, k_e back EMF constant, J is rotor inertia, τ_{motor} is the motor torque, k_T is the torque constant, τ_{drag} is the rotor drag torque, and b is the drag coefficient. Evaluating this at equilibrium gives

$$v = \frac{Rb}{k_T}\omega^2 + k_e\omega$$

Fitting a quadratic fixed at the origin to v vs ω provides estimates of k_e and $\frac{Rb}{k_T}$. However, voltage delivered to motors is not directly known. Instead, we know pulsewidth commands. Pulsewidth is converted to voltage by

$$v = \frac{v_{battery}}{PW_{max} - PW_{min}}(PW - PW_{min})$$

where $v_{battery}$ is the current battery power, which is measured during the experiments.

Next, R can be measured directly with an Ohmmeter (or indirectly with a voltmeter and ammeter). The terms k_e and k_T are algebraically related, because $k_e = V/\omega$ and $k_T = \tau_{motor}/i$. Electrical and mechanical power balance gives $\sqrt{3}vi = \frac{2\pi}{60}\omega\tau$, where the left hand side is rms electrical power and the right hand side is mechanical power in Nm/s. Rearranging gives

$$k_T = \frac{60\sqrt{3}}{2\pi}k_e$$

Then b is solved for from the known value of $\frac{Rb}{k_T}$. The estimated parameters are stored in System Identification/Drone Paramters.xlsx. They are calculated in System Identification/MotorCharacterization.m using data from System Identification/MotorTestData.m.