Modeling Documentation

From a dynamics perspective, the quadrotor is essentially composed to two systems: the rigid free body frame and the motor-rotor elements.

From Bouabdallah, Noth, and Siegwart, “PD vs LQ control techniques applied to an indoor micro quadrotor” *ETH Library*, 2004, the motor dynamics are

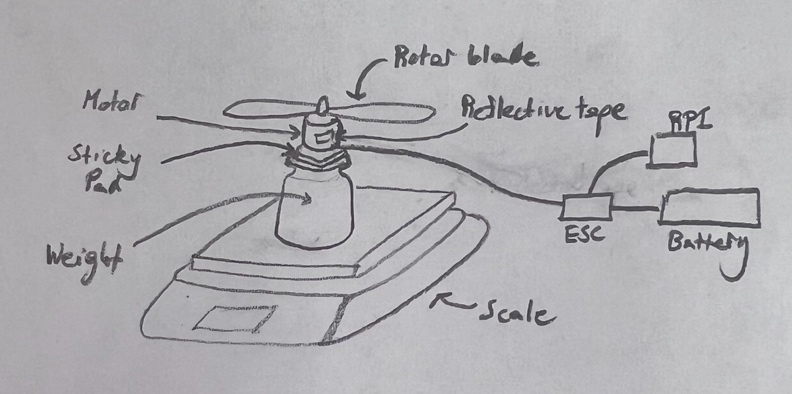
(The last equation is actually from Lukkonen, “Modeling and control of a quadcopter” *Aalto University Independent Research Project*, 2011), where *v* is the voltage applied to the motor, *i* is the current applied to the motor, *w* is the motor speed, *R* is the internal electrical resistance of the motor, *L* is the inductance of the motor*,*  is the back EMF constant, *J* is the motor inertia*,* is the torque constant of the motor, and *b* is the drag coefficient of the rotor blade. It is assumed that the inductance and inertia are small, so these equations are approximately steady-state on the time-scale of the drone. Therefore,

Also, from Lukkonen,

Where *F* is the lift force of the motor/rotor and k is the lift coefficient of the rotor blade. There are 5 parameters to estimate. The motor resistance, *R*, can be measured experimentally using a multimeter, as explained in: https://www.youtube.com/watch?v=4GgE01acZ6E\&ab\\_channel=RCexplained

The torque constant, , is directly related to the back EMF constant, , according to

Which, according to the document “Measuring Motor Parameters” (source unknown), comes from the electrical and mechanical power balance. (The source document uses mechanical power in lb-inches, so there is an extra conversion term, whereas here, the units are Newton-meters). Therefore, the only remaining parameters are , *b*, and *k*. These can be estimated together using a single set of experiments to collect data that fits the two equations relating voltage to rotor speed to lift force. The experimental setup, depicted below, requires a single motor mounted with a sticky pad to a weight, which is placed on a scale. Reflective tape is applied to the motor casing for rpm measurements by a laser tachometer. The motor is powered by a the battery, ESC, and RPI, which are placed off the scale. The RPI code should be set up to manually input pulsewidth commands and display the battery voltage. Record the pulsewidth, the battery voltage, the rotation rate (with a laser tachometer) and the lift force (difference in the scale weight). Repeat the experiment at multiple battery voltages and pulsewidths and with each different motor. Then, in matlab, find the linear fit of lift force vs rotor speed squared, which gives *k*. Then calculate the voltage applied to the motor according to . This comes from the PW to voltage relation being linear, with PW=1100 giving v=0 and PW=1900 giving v=v\_battery. Now, we can find a quadratic fit of the form for the voltage vs rotor speed. Calculate from , then .



This has been implemented in the matlab script MotorCharacterization.m. The currently estimated values are

R: 0.17

k: 1.2879e-07

ke: 0.0006563

kT: 0.010855

b: 8.2123e-09

Next, we must model the rigid body dynamics. For the attitude dynamics, we use Lukkonen’s Euler-Lagrange equations, and for the x,y,z position dynamics, we use Lukkonen’s Newton-Euler equations.. The attitude dynamics are governed by Equation 20:

Where is the vector of roll, pitch, yaw angles (their symbolic convention for roll and pitch may be flipped from ours), is the vector of corresponding torques, is the Jacobian matrix given in Equation 16, and is the Coriolis matrix given in Equation 19. **For ,** **don’t use Lukkonen’s Equation 8.** The sign conventions differ based on the orientation of the blades. The way we have it set up, use Gamma altered from Mahony’s Equation 8 as defined in the Controller.py code.

The x,y,z position dynamics are governed by Equation 10:

(Remember, the convention of theta and phi might be switched, so be careful), where *m* is the mass of the drone, *g* is gravity, and

Where is the rotor speed of the *i-th* motor, and *k* is the lift coefficient of the blade.

The model is implemented in the folder “Control Design”, where the system is linearized, an LQR controller is designed for the linearized system, and then the closed-loop is simulated with both the linear and nonlinear system.

**This model has not been validated against the physical system. If you want to do model-based design, it’s recommended that you implement the PD controller described in “Control Documentation” in the simulation, and then tune the model so that the behavior is consistent. May also be worth re-measuring parameters, as the hardware configuration has changed somewhat since the last measurements.**