Quadrotor Control: Linear Quadratic Method

Equations of motion, including body moment of inertia and propeller gyroscopic effect. The first term is the body moment term affecting all three axes. Only the roll and pitch axes are influenced by propeller gyroscopic effect. The propeller axes are parallel with the yaw axis, so yaw is not effected by this.

Command Inputs

The drive inputs are modelled as $U_{1,2,3}$. Propeller rotational rates are $\Omega_{1,2,3,4}$.

U_1 is cross-body differential drive representing the propeller pair that drives body roll. We derived it as $b \cdot l \cdot \left({\Omega_2}^2 - {\Omega_4}^2 \right)$ where 'b' is the, "thrust factor" and, 'l' is the distance from the body center of of mass and the propeller axes.

U_2 is cross-body differential drive representing the propeller pair that drives body pitch. As with roll above it is $b \cdot l \cdot \left({\Omega_1}^2 - {\Omega_3}^2 \right)$.

U_3 is the yaw drive input. We derived this as $d \cdot \left({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_3}^2 - {\Omega_1}^2 \right)$ where 'd' is the propeller, "drag factor" and the sum of counter-clockwise rotations squared less the squared clockwise rotations models the four propellers as one single propeller with a net squared rotaitonal rate that yileds yaw rate proportional to the drag factor for a single propeller

Bold omega with no subscript($\mathbf{\Omega}$) is net propeller rotation, counter-clockwise minus clockwise:

$$\mathbf{\Omega} = \Omega_1 + \Omega_3 - \Omega_2 - \Omega_4$$

We derived the gyroscopic effect on roll and pitch. The sign difference on the $\mathbf{\Omega}$ is explained by the derivation.

When our controller calculates a desired command input $U_{1,2,3}$ it will apportion it out to the four propellers ratiometrically to achieve the required command effect. For example, if we need yaw the controller will determine the net difference in the counter-clockwis and clockwise propeller rates, then we will scale the cross-body pairs that drive roll and pitch to satisfy the $U_{1,2}$ command rquirement.

This last bit of command condiitoning will yield the specific motor drive commands. Logically $U_{1,2,3}$ are the result of the control law. Physically the motor drive voltages result in the actual output.

Ω is modelled as a, "command" input but it is actually a consequence of the above $U_{1,2,3}$ logic. The propeller gyroscopic effect is an effect of our desired drive, but it appears as a fourth, "command input".

Equations of Motion

Roll

$$\ddot{\phi} = \dot{\theta}\dot{\psi}\left(\frac{I_y - I_z}{I_x}\right) - \frac{J}{I_x}\dot{\theta}\,\mathbf{\Omega} + \frac{l}{I_x}U_1$$

Pitch

$$\ddot{\theta} = \dot{\phi}\dot{\psi}\left(\frac{I_z - I_x}{I_y}\right) + \frac{J}{I_y}\dot{\phi}\,\mathbf{\Omega} + \frac{l}{I_y}U_2$$

Yaw

$$\ddot{\psi} = \dot{\theta}\dot{\phi}\left(\frac{I_x - I_y}{I_z}\right) + \frac{1}{I_z}U_3$$

State Model

It's difficult to see how to linearize this equation with the rate product in the body moment term (first terms in the equations above. The Bouaddallah paper offers a trick. The paper models the states as...

$$X = \begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \Psi \\ \dot{\Psi} \end{bmatrix}$$

Then the body inertial effect is split between 2 entries in the state transition matrix A as follows. The, "split" requires the 2 in the denominator of the A terms. When calculate it out you'll get the equations above.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\dot{\psi}}{2} \cdot \left(\frac{I_y - I_z}{I_x} \right) & 0 & \frac{\dot{\theta}}{2} \cdot \left(\frac{I_y - I_z}{I_x} \right) \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{\dot{\psi}}{2} \cdot \left(\frac{I_z - I_x}{I_y} \right) & 0 & 0 & 0 & \frac{\dot{\phi}}{2} \cdot \left(\frac{I_z - I_x}{I_y} \right) \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{\dot{\theta}}{2} \cdot \left(\frac{I_x - I_y}{I_z} \right) & 0 & \frac{\dot{\phi}}{2} \cdot \left(\frac{I_x - I_y}{I_z} \right) & 0 & 0 \end{bmatrix}$$

The B matrix captures the gains on the control intputs modelled as descripd above:

$$B=\begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{l}{I_x} & 0 & 0 & \frac{J}{I_x}\cdot\dot{\theta} \\ 0 & 0 & 0 & 0 \\ 0 & \frac{l}{I_y} & 0 & \frac{J}{I_y}\cdot\dot{\phi} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} & 0 \end{bmatrix}$$

The 'C' vector yields the model outputs, which are the attitude angle and rates that we'll estimate from gyroscopes and accelerometers on the platform. In practice these will feedback to our reference input via a state, "estimator" whic will fuse sensor inputs from the platform and apply a Kalman filter to appropriately weigh inputs into the filter at rates higher than our closed loop control rate, making available an appropriate state estimate for our controller on each controller iteration.

$$C=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We have no direct feed-through of attitude command input to output so our 'D' Matrix is as follows

$$D=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This leaves us with our plant modelled in the familiar form...

$$\dot{\boldsymbol{X}}=\boldsymbol{A}\bullet\boldsymbol{X}+\boldsymbol{B}\bullet\boldsymbol{U}$$
$$\boldsymbol{Y}=\boldsymbol{C}\bullet\boldsymbol{X}+\mathbf{D}\bullet\boldsymbol{U}$$

Next Steps...

We can see above that to use this model we need to supply roll- and yaw-rate conditions to the A and B matrices in order to linearize the system. We'll do this successively throughout flight. Once done, if we want to regulate to, "horizontal" we can employ Matlab's, "LQR" function. We'll not do this in real-time, but instead compute a gain table for an operational range of roll- and pitch-rate.

We also might not want a, "regulator" to horizontal. We might want to track to a desired copter platform attitude in order to thrust away from horizontal to say intercept a target.

In either case, we're going to dig deeper into the Linear Quadratic Regulator or Tracking gain calculations an not simply rely on Matlab LQR() (or LQI() for the tracking problem) .