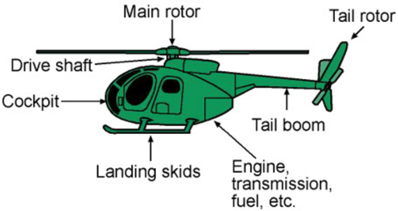
Quadcopter Dynamics, Simulation, and Control

Andrew Gibiansky
andrew.gibiansky@gmail.com

November 29, 2012

Helicopter



Swashplate

Helicopter Thrusts

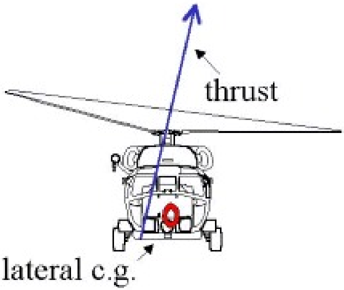
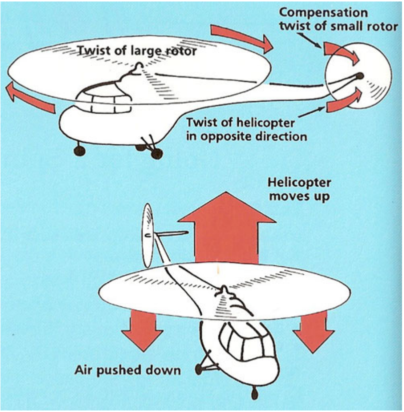


Figure 1

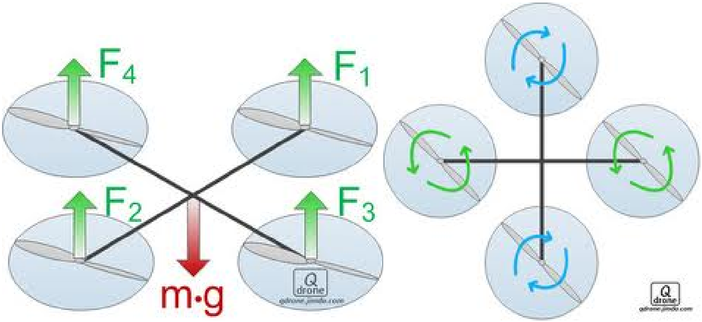


Swashplate mechanism provides control over extra degrees of freedom.

Quadcopters

Note the lack of swashplates!

Quadcopter Thrusts



Degrees of freedom provided by extra motors.

Quadcopters vs. Helicopters

- Quadcopters
  - Simpler mechanism: no swashplates.
  - Can be very small, yet controllable.
  - Very versatile.
  - Cheap!
- Helicopters
  - Much simpler to control (possible without electronic stabilization).
  - Have been scaled up to large sizes.

Frames

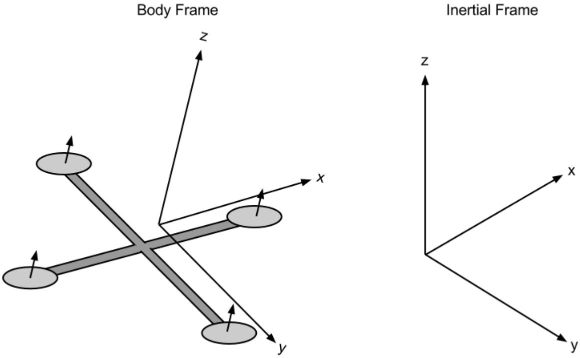


Figure: Quadcopter Body Frame and Inertial Frame

Forces and Torques

- Thrust force:

$$T_B = \sum_{i=1}^{4} T_i = k \begin{bmatrix} 0 \\ 0 \\ \sum \omega_i{}^2 \end{bmatrix}.$$

- Drag force:

$$F_D = \begin{bmatrix} -k_d \dot{x} \\ -k_d \dot{y} \\ -k_d \dot{z} \end{bmatrix}$$

- Motor torques:

$$\tau_B = \begin{bmatrix} Lk(\omega_1{}^2 - \omega_3{}^2) \\ Lk(\omega_2{}^2 - \omega_4{}^2) \\ b\left(\omega_1{}^2 - \omega_2{}^2 + \omega_3{}^2 - \omega_4{}^2\right) \end{bmatrix}$$

- Inertial Frame:

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B + F_D$$

- Body Frame:

$$m\ddot{x} = R^{-1} \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + T_B + R^{-1} F_D - \omega \times (m\dot{x})$$

Equations of Motion: Rotational

- Euler's Equations (Rigid Body Dynamics):

$$I\dot{\omega} + \omega \times (I\omega) = \tau$$

- Body Frame Dynamics:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}.$$

$$\dot{\omega} = \begin{bmatrix} \tau_\phi I_{xx}{}^{-1} \\ \tau_\theta I_{yy}{}^{-1} \\ \tau_\psi I_{zz}{}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}\omega_y\omega_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\omega_x\omega_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix}$$

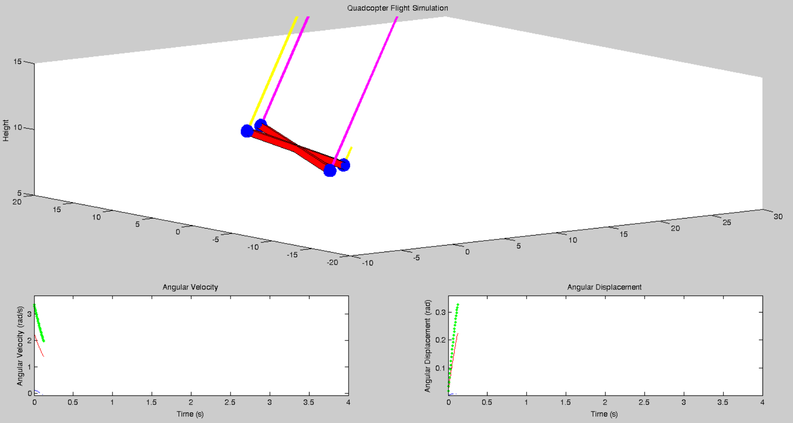
Simulation

```
% Simulation times, in seconds.
start_time = 0; end_time = 10; dt = 0.005;
times = start_time:dt:end_time;
N = numel(times);

% Initial simulation state.
x = [0; 0; 10]; xdot = zeros(3, 1); theta = zeros(3, 1);

% Simulate some disturbance in the angular velocity.
% The magnitude of the deviation is in radians / second.
deviation = 100;
thetadot = deg2rad(2 * deviation * rand(3, 1) - deviation);

% Step through the simulation, updating the state.
for t = times
    i = input(t);                                         % Get controller input
    omega = thetadot2omega(thetadot, theta);              % Convert (φ̇, θ̇, ψ̇) → ω̇
    a = acceleration(i, theta, xdot, m, g, k, kd);        % Compute linear acceleration
    omegadot = angular_acceleration(i, omega, I, L, b, k); % Compute angular acceleration
    omega = omega + dt * omegadot;                        % ω = ω + dt × ω̇
    thetadot = omega2thetadot(omega, theta);              % Convert ω → (φ̇, θ̇, ψ̇)
    theta = theta + dt * thetadot;                        % θ⃗ = θ⃗ + dt × (φ̇, θ̇, ψ̇)
    xdot = xdot + dt * a;                                 % ẋ = ẋ + dt × a
    x = x + dt * xdot;                                    % x = x + dt × ẋ
end
```



YouTube Movie

PD Control

- Control signal should be turned into a torque:

$$\tau = Iu(t) = I\dot{\omega}$$

- We can set torques:

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_d\dot{\phi} + K_p \int_0^T \dot{\phi}\,\mathrm{d}t\right) \\ -I_{yy}\left(K_d\dot{\theta} + K_p \int_0^T \dot{\theta}\,\mathrm{d}t\right) \\ -I_{zz}\left(K_d\dot{\psi} + K_p \int_0^T \dot{\psi}\,\mathrm{d}t\right) \end{bmatrix}$$

- Our electronic gyro outputs angular velocities, so we integrate them to get the angle. This integral is multiplied by the *proportional* gain.

Motor Angular Velocities

- We do not actually set torques, we set voltages over the motors. These correspond directly to the motor angular velocities ω_i.
- We can solve for the inputs $\gamma_i = {\omega_i}^2$:

$$\tau_B = \begin{bmatrix} Lk(\gamma_1 - \gamma_3) \\ Lk(\gamma_2 - \gamma_4) \\ b\,(\gamma_1 - \gamma_2 + \gamma_3 - \gamma_4) \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_d\dot{\phi} + K_p \int_0^T \dot{\phi}\,\mathrm{d}t\right) \\ -I_{yy}\left(K_d\dot{\theta} + K_p \int_0^T \dot{\theta}\,\mathrm{d}t\right) \\ -I_{zz}\left(K_d\dot{\psi} + K_p \int_0^T \dot{\psi}\,\mathrm{d}t\right) \end{bmatrix}$$

- For the last constraint, choose one to keep the quadcopter aloft:

$$T = \frac{mg}{\cos\theta\cos\phi}$$

PD Control

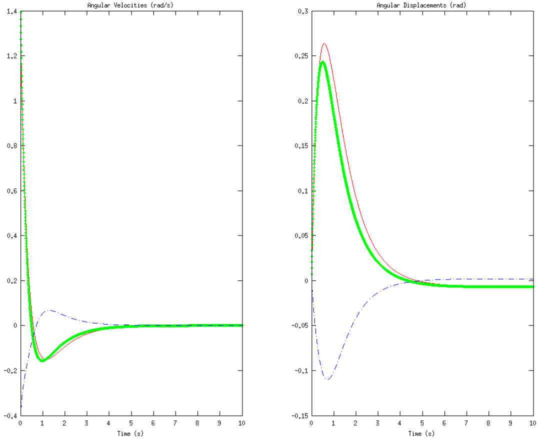


Figure: Left: Angular velocities. Right: angular displacements. ϕ, θ, ψ are coded as red, green, and blue. Note that there is some small steady-state error (approximately 0.3°).

- We can do better by adding an integral term to make this a PID controller.

$$e_\phi = K_d\dot\phi + K_p\int_0^T \dot\phi\,\mathrm{d}t + K_i\int_0^T\int_0^T \dot\phi\,\mathrm{d}t\,\mathrm{d}t$$
$$e_\theta = K_d\dot\theta + K_p\int_0^T \dot\theta\,\mathrm{d}t + K_i\int_0^T\int_0^T \dot\theta\,\mathrm{d}t\,\mathrm{d}t$$
$$e_\psi = K_d\dot\psi + K_p\int_0^T \dot\psi\,\mathrm{d}t + K_i\int_0^T\int_0^T \dot\psi\,\mathrm{d}t\,\mathrm{d}t$$

- To avoid integral wind-up, only start integrating the double-integral once the angle deviation is relatively small.

Integral Windup

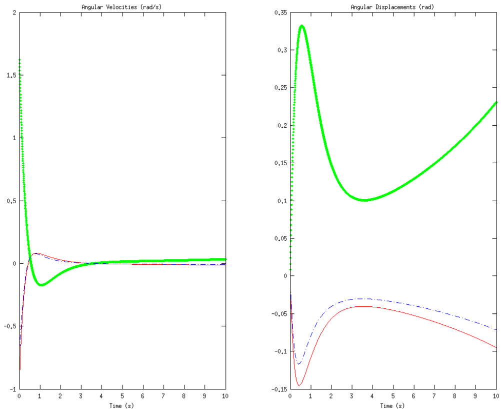


Figure: In some cases, integral wind-up can cause lengthy oscillations instead of settling. In other cases, wind-up may actually cause the system to become unstable, instead of taking longer to reach a steady state.

PID Control

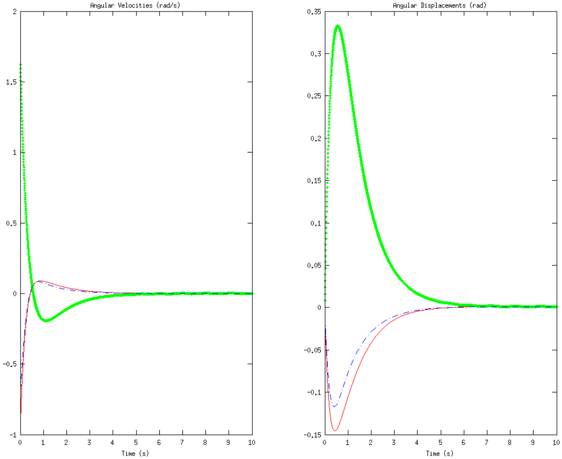


Figure: With a properly implemented PID, we achieve an error of approximately 0.06° after 10 seconds.

Automatic Gain Tuning

- Tuning the PID gains (K_i, K_p, and K_d) can be difficult.
- Quality of results depends on gain values and the ratios of the different gain values.
- "Best" gains might be different for different modes of operation.
- Requires expert intuition and a lot of time to tune them.
- We would like to do this automatically.

Extremum Seeking

- Define a cost function, defining the quality of a set of parameters:

$$J(\vec{\theta}) = \frac{1}{t_f - t_o} \int_{t_0}^{t_f} e(t, \vec{\theta})^2 \, \mathrm{d}t$$

- Use gradient descent to minimize this cost function in parameter-space:

$$\vec{\theta}(k+1) = \vec{\theta}(k) - \alpha \nabla J(\vec{\theta})$$

- Approximate gradient numerically:

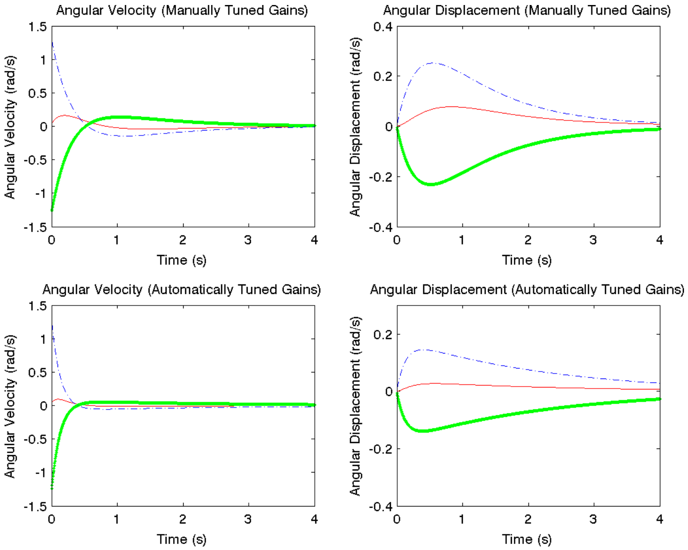
$$\nabla J(\vec{\theta}) = \left(\frac{\partial}{\partial K_p} J(\vec{\theta}), \frac{\partial}{\partial K_i} J(\vec{\theta}), \frac{\partial}{\partial K_d} J(\vec{\theta}) \right).$$

$$\frac{\partial}{\partial K} J(\vec{\theta}) \approx \frac{J(\vec{\theta} + \delta \cdot \hat{u}_K) - J(\vec{\theta} - \delta \cdot \hat{u}_K)}{2\delta}$$

Gradient Descent Tricks

- Adjust step size α as we go along, to become more precise as time goes on.
- Computing $e(t, \vec{\theta})$ means running a simulation with some random initial disturbance. Use many simulations, take the average. As we go along, average more simulations.
- Repeat many times to produce many local minima, then choose the best one and call it the "global minimum".
- Automatically choose a time to stop iterating (when we're no longer improving our average cost).

Manual Gains vs. Automatically Tuned Gains



Questions?

Thank you to Professors Donatello Materassi and Rob Wood for their help.