WashU McKelvey Engineering

Spring 2025 ESE 4481 Autonomous Aerical Vehicle Control Lab

CrazyFlie 2.1 Cascaded PID/PD Control System

A tale of frequency frustration

Ian Snider

Lab: *Chengyu Li, Nicole Ejedimu*

May 22, 2025

1 Objectives

The CrazyFlie 2.1 is a small 33 g quadcopter. We seek to design a control system to achieve the following objectives:

- Take off and hover
- Land?
- Danforth aerial supremacy

We design a pole-placement controller in the form,

$$u = K(r - \hat{x}) \tag{1}$$

$$v = M(u) \tag{2}$$

where,

- u is the controller output vector.
- K is the gain matrix mapping state errors to state inputs.
- $\hat{x}$ is the estimated state of the drone.
- r is the setpoint vector.
- M is the mixer function.
- v is the PWM vector (32-bit unsigned integers from 0 to 65,536).

The controller will be implemented to track a setpoint defined in the ENU frame as,

$$\begin{bmatrix} p_x & p_y & p_z & \dot{p}_x & \dot{p}_y & \dot{p}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0.5 & 0 & 0 & 0 \end{bmatrix}$$

where the states are in [m] and [m/s]. The controller is design to output the following in SI units,

$$u = \begin{bmatrix} Z & L & M & N \end{bmatrix}$$

where Z is the upward thrust [N], L is the rolling moment [N·m], M is the pitching moment [N·m], and N is the yawing moment [N·m]. The mixer will be designed to convert the above controller outputs to PWM using the following empirical quadratic relationship,

$$\text{thrust} = 0.091492681f \cdot \text{PWM}^2 + 0.067673604f \cdot \text{PWM} \tag{3}$$

The drone's inertia matrix in the body frame is given as the following,

$$I = \begin{bmatrix} 16.571710 & 0.830806 & 0.718277 \\ 0.830806 & 16.655602 & 1.800197 \\ 0.718277 & 1.800197 & 29.261652 \end{bmatrix} \cdot 10^{-6} \tag{4}$$

which is given in [kg·m^2]. Notably, the cross terms are much smaller than the diagonal terms.

2 Dynamics

To begin designing a control system, we must first understand the nonlinear dynamics. The combined nonlinear equations of motions describing the drone dynamics are shown below,

$$
\begin{aligned}
\begin{pmatrix}\dot{p}_x\\ \dot{p}_y\\ \dot{p}_z\end{pmatrix} &= \begin{pmatrix}\mathrm{c}_\theta\mathrm{c}_\psi & \mathrm{s}_\phi\mathrm{s}_\theta\mathrm{c}_\psi - \mathrm{c}_\phi\mathrm{s}_\psi & \mathrm{c}_\phi\mathrm{s}_\theta\mathrm{c}_\psi + \mathrm{s}_\phi\mathrm{s}_\psi\\ \mathrm{c}_\theta\mathrm{s}_\psi & \mathrm{s}_\phi\mathrm{s}_\theta\mathrm{s}_\psi + \mathrm{c}_\phi\mathrm{c}_\psi & \mathrm{c}_\phi\mathrm{s}_\theta\mathrm{s}_\psi - \mathrm{s}_\phi\mathrm{c}_\psi\\ -\mathrm{s}_\theta & \mathrm{s}_\phi\mathrm{c}_\theta & \mathrm{c}_\phi\mathrm{c}_\theta\end{pmatrix}\begin{pmatrix}u\\ v\\ w\end{pmatrix}\\
\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix} &= \begin{pmatrix}rv - qw\\ pw - ru\\ qu - pv\end{pmatrix} + \frac{1}{\mathsf{m}}\begin{pmatrix}0\\ 0\\ Z - \mathsf{m}g\end{pmatrix},\\
\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{pmatrix} &= \begin{pmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta}\end{pmatrix}\begin{pmatrix}p\\ q\\ r\end{pmatrix}\\
\begin{pmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{pmatrix} &= \begin{pmatrix}\Gamma_1 pq - \Gamma_2 qr\\ \Gamma_5 pr - \Gamma_6(p^2 - r^2)\\ \Gamma_7 pq - \Gamma_1 qr\end{pmatrix} + \begin{pmatrix}\Gamma_3 L + \Gamma_4 N\\ \frac{1}{J_y}M\\ \Gamma_4 L + \Gamma_8 N\end{pmatrix}
\end{aligned}
$$

The dynamics are comprised of 12 states representing the following quantities,

- p_x - x position in the earth frame [m]
- p_y - y position in the earth frame [m]
- p_z - z position in the earth frame [m]
- u - x velocity in the body frame [m]
- v - y velocity in the body frame [m]
- w - z velocity in the body frame [m]
- ϕ - roll angle [rads]
- θ - pitch angle [rads]
- ψ - yaw angle [rads]
- p - rolling rate [rads/s]
- q - pitching rate [rads/s]
- r - yawing rate [rads/s]

The CrazyFlie 2.1 is able to report estimates of the above states (although requiring a rotation transform for the u, v, w states) using a Kalman filter.

3 Linearization

That's a crazy nonlinear system, I guess this is where we give up. WRONG. To derive approximate dynamics, we find the Jacobian of the system and evaluate at the setpoint. The

resulting linearized A and B matrices are shown below,

$$A = \begin{pmatrix} 0 & 0 & 0 & \mathbf{1.0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1.0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1.0} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1.0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1.0} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1.0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1.0}{\mathsf{m}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -\frac{1.0\,J_{zz}}{{J_{xz}}^2 - 1.0\,J_{xx}\,J_{yy}} & 0 & -\frac{1.0\,J_{xz}}{{J_{xz}}^2 - 1.0\,J_{xx}\,J_{yy}} \\ 0 & 0 & \frac{1}{J_{yy}} & 0 \\ 0 & -\frac{1.0\,J_{xz}}{{J_{xz}}^2 - 1.0\,J_{xx}\,J_{yy}} & 0 & -\frac{1.0\,J_{xx}}{{J_{xz}}^2 - 1.0\,J_{xx}\,J_{yy}} \end{pmatrix} \tag{6}$$

The above linearized system can then be used to derive the trim controls. To simplify the analysis even further, we recognize that the cross terms for the angular rates are much smaller than the diagonal terms. We then neglect the cross terms and find that B becomes,

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1.0}{\mathsf{m}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{J_{xx}} & 0 & 0 \\ 0 & 0 & -\frac{1}{J_{yy}} & 0 \\ 0 & 0 & 0 & -\frac{1}{J_{zz}} \end{pmatrix} \tag{7}$$

3.1 Controllability of Design

The quadcopter control system we are designing is inherently underactuated. The drone only has 4 inputs: thrust in the z-direction and 3 torques. The full dynamics, however, consists of 12 states. We find the controllability matrix as,

$$W_c = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$$

From this computation, we find that $\text{rank}(W_c) = 8$. Thus, the linearized system is not fully controllable in all 12 states. However, the system is controllable in the most critical subspaces. In particular, the controllable states consists of vertical position, vertical velocity, attitudes, and angular rates. The horizontal positions are stabilizable by controlling the attitude and the horizontal velocities are stabilizable by controlling angular rate.

Linearization and the controllability analysis where complete using the Matlab code in Listing 1.

Listing 1: MATLAB linearization and controllability.

```
clc; clear; close all;

syms pn pe pd u v w phi theta psi p q r Z L M N mass g
syms J_xx J_xy J_xz J_yx J_yy J_yz J_zx J_zy J_zz

J = [J_xx, J_xy, J_xz; J_yx, J_yy, J_yz; J_zx, J_zy, J_zz];
gammas_vector = inertia_matrix(J);

[A, B] = make_dynamics_for_paper(u,v,w,phi,theta,psi,p,q,r,Z, L, M, N, mass, g, gammas_vector, J);
JA = vpa(jacobian(A, [pn pe pd u v w phi theta psi p q r]));
JA = vpa(subs(JA,[pn pe pd u v w phi theta psi p q r], [ 0 0 0.5 0 0 0 0 0 0 0 0 0]))
JB = vpa(subs(JB,[mass J_xx J_xy J_xz J_yx J_yy J_yz J_zx J_zy J_zz], [33 16.571710 0.830806 0.718277 0.830806
     16.655602 1.800197 0.718277 1.800197 29.261652]))
W_c = ctrb(JA, JB)
rank(W_c)
function [xdotA, xdotB] = make_dynamics_for_paper(u, v, w, phi, theta, psi, p, q, r, fz, l, m, n, mass, g, gamma,
      inertia)
    position = [w*(sin(phi)*sin(psi) + cos(phi)*cos(psi)*sin(theta)) - v*(cos(phi)*sin(psi) - cos(psi)*sin(phi)*
         sin(theta)) + u*cos(psi)*cos(theta);...
                v*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta)) - w*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*
                     sin(theta)) + u*cos(theta)*sin(psi);...
                w*cos(phi)*cos(theta) - u*sin(theta) + v*cos(theta)*sin(phi)];

    velocityA = [r*v - q*w;...
                p*w - r*u;...
                q*u - p*v];

    velocityB = [0;0;fz/mass - g];

    angle = [p + r*cos(phi)*tan(theta) + q*sin(phi)*tan(theta);...
             q*cos(phi) - r*sin(phi);...
             (r*cos(phi)/cos(theta)) + (q*sin(phi)/cos(theta))];

    [rateA, rateB] = angular(p,q,r,l,m,n,gamma,inertia);

    xdotA = [position; velocityA; angle; rateA];
    xdotB = [zeros(3,1); velocityB; zeros(3,1); rateB];
end

% angular rate
```

```
function [A,B] = angular(p,q,r,l,m,n,gamma,inertia)
    Jy = inertia(2,2);
    g1 = gamma(1);
    g2 = gamma(2);
    g3 = gamma(3);
    g4 = gamma(4);
    g5 = gamma(5);
    g6 = gamma(6);
    g7 = gamma(7);
    g8 = gamma(8);

    A = [g1*p*q - g2*q*r;...
        g5*p*r - g6*(p^2 - r^2);...
        g7*p*q - g1*q*r];

    B = [g3*l + g4*n;...
        (1/Jy)*m;...
        g4*l + g8*n];

end

% rotational dynamics
function gammas_vector = inertia_matrix(J)
    Jx = J(1,1);
    Jy = J(2,2);
    Jz = J(3,3);
    Jxz = J(1,3);

    gamma = Jx*Jy - Jxz^2;

    gamma_1 = Jxz*(Jx - Jy + Jz)/gamma;
    gamma_2 = (Jz*(Jz - Jy) + Jxz^2)/gamma;
    gamma_3 = Jz/gamma;
    gamma_4 = Jxz/gamma;
    gamma_5 = (Jz - Jx)/Jy;
    gamma_6 = Jxz/Jy;
    gamma_7 = ((Jx - Jy)*Jx + Jxz^2)/gamma;
    gamma_8 = Jx/gamma;

    gammas_vector = [gamma_1, gamma_2, gamma_3, gamma_4, gamma_5, gamma_6, gamma_7, gamma_8];
end
```

4 Trim Controls

The trim controls are then expressed as,

$$\ddot{p}_z = R_p^{ENU}\dot{w} = R_p^{ENU}\left(\frac{Z}{\mathsf{m}} - g\right) \tag{8}$$

$$\ddot{\phi} = \dot{p} = \frac{L}{J_{xx}} \tag{9}$$

$$\ddot{\theta} = \dot{q} = \frac{M}{J_{yy}} \tag{10}$$

$$\ddot{\psi} = \dot{r} = \frac{N}{J_{zz}} \tag{11}$$

The units for Z, L, M, and N are then [N], [N·m], [N·m], [N·m], and [N·m].

5 Cascaded PID/PD Control System

The cascaded PID/PD control system we seek to design involves PID in the outer loop to prevent drift due to position errors and a faster PD control system in the inner loop to control attitude errors. The cascaded PID/PD control system is a good balance between computational efficiency, ease of gain tuning, and robustness. The ease of gain tuning is an especially important metric because I have never tuned a quadcopter control system. The cascaded PID/PD separates fast attitude control from slower position control, which can make tuning the gains a bit more tractable. The cascaded approach is used for the rolling and pitching moments. However, single PD loops are used for the thrust and yawing moment. This variation in the control design is advisable because the yaw and thrust are naturally decoupled from the rolling and pitching moments. For Z thrust control, the steady-state error is mitigated by gravity, so integral control is not necessary.

Additionally, for a controlled indoor environment, such as Green 1157, disturbances are small and low-frequency, so a cascaded PID/PD control design should at least somewhat work for stable hovering. Cascaded PID/PD can struggle to keep up with aggressive maneuvers due to potential inner loop lagging. Fortunately, hovering at 0.5 m should not involve aggressive maneuvers. Slow drift could actually be a huge issue for the CrazyFlie 2.1 due to its pathetic battery life and questionable state estimation, which is the primary reason for the integral control.

Finally, model-based optimization is generally a recommended strategy when computationally permitted. If I had wanted to suffer slightly more, I would've replaced the inner PD loop with LQR for optimal attitude control...next time!

5.1 Control Design

The single PD controller for the Z thrust is expressed as,

$$\begin{aligned} Z &= mg + k_{pz}e_z + k_{dz}\dot{e}_z \\ &= mg + k_{pz}(p_{zc} - \hat{p}_z) + k_{dz}\dot{\hat{p}}_z \end{aligned}$$

where k_{pz} and k_{dz} are the proportional and derivative gains respectively. Similarly, the single PD controller for the N yawing moment is expressed as,

$$\begin{aligned} N &= k_{p\psi}e_\psi + k_{d\psi}\dot{e}_\psi \\ &= k_{p\psi}\hat{\psi} + k_{d\psi}\hat{r} \end{aligned}$$

where $k_{p\psi}$ and $k_{d\psi}$ are the proportional and derivative gains respectively. The cascaded PID/PD controller for the L rolling moment is expressed as,

$$\begin{aligned} L &= k_{p\phi}e_\phi + k_{d\phi}\dot{e}_\phi \\ &= k_{p\phi}(\phi_c - \hat{\phi}) + k_{d\phi}\hat{p} \\ &= k_{p\phi}\left(k_{py}\hat{p}_y + k_{iy}\int \hat{p}_y dy + k_{dy}\dot{\hat{p}}_y - \hat{\phi}\right) + k_{d\phi}\hat{p} \end{aligned}$$

where k_{py}, k_{iy}, k_{dy} are the proportional, integral, and derivative gains for the outer PID loop and $k_{p\phi}$ and $k_{d\phi}$ are the proportional and derivative gains for the inner PD loop. Similarly, the cascaded PID/PD controller for the M pitching moment is expressed as,

$$\begin{aligned} M &= k_{p\theta} e_\theta + k_{d\theta} \dot{e}_\theta \\ &= k_{p\theta}(\theta_c - \hat{\theta}) + k_{d\theta}\hat{q} \\ &= k_{p\theta}\left(k_{px}\hat{p}_x + k_{ix}\int \hat{p}_x dx + k_{dx}\dot{\hat{p}}_x - \hat{\theta}\right) + k_{d\theta}\hat{q} \end{aligned}$$

where k_{px}, k_{ix}, k_{dx} are the proportional, integral, and derivative gains for the outer PID loop and $k_{p\theta}$ and $k_{d\theta}$ are the proportional and derivative gains for the inner PD loop.

5.2 Gain Selection

Gain relationships are derived from the closed-loop transfer functions. For the thrust, these gains are selected in the following form,

$$k_{pz} = m\omega_{nz}^2 \qquad k_{dz} = m2\zeta_z\omega_{nz}$$

where ω_n is the bandwidth and ζ is the damping coefficent. Gains for the torques, either single loop or inner loop, are selected as follows,

$$\begin{aligned} k_{p\phi} &= J_{xx}\omega_{n\phi}^2 & k_{d\phi} &= J_{xx}2\zeta_\phi\omega_{n\phi} \\ k_{p\theta} &= J_{yy}\omega_{n\theta}^2 & k_{d\theta} &= J_{yy}2\zeta_\theta\omega_{n\theta} \\ k_{p\psi} &= J_{zz}\omega_{n\psi}^2 & k_{d\psi} &= J_{zz}2\zeta_\psi\omega_{n\psi} \end{aligned}$$

Gains for the outer PD loops are selected in the following form,

$$\begin{aligned} k_{px} &= \omega_{nx}^2/g & k_{ix} &= \omega_{nx}^3/g & k_{dx} &= 2\zeta_x\omega_{nx}/g \\ k_{py} &= \omega_{ny}^2/g & k_{iy} &= \omega_{ny}^3/g & k_{dy} &= 2\zeta_y\omega_{ny}/g \end{aligned}$$

Gains are then selected by tuning the bandwidth and damping coefficient (within reasonable bounds dictated by hardware limitations).

5.3 Control System Architecture

Figures 2, 3, 4, & 5 contain block diagrams of the control system architecture. Note, that after each computation of the input signal, a motor transfer function of the form,

$$H(z) = \frac{7.234537 \times 10^{-8}}{1 - 0.9695404z^{-1}} \tag{12}$$

must be applied. This discrete time transfer function normalizes the thrust, so a scaling factor must also be applied. Additionally, the transfer function is sufficiently approximated in continuous time as,

$$H(s) = \frac{10}{s + 10} \tag{13}$$

Bode diagrams of the motor transfer functions are shown in Fig. 1.

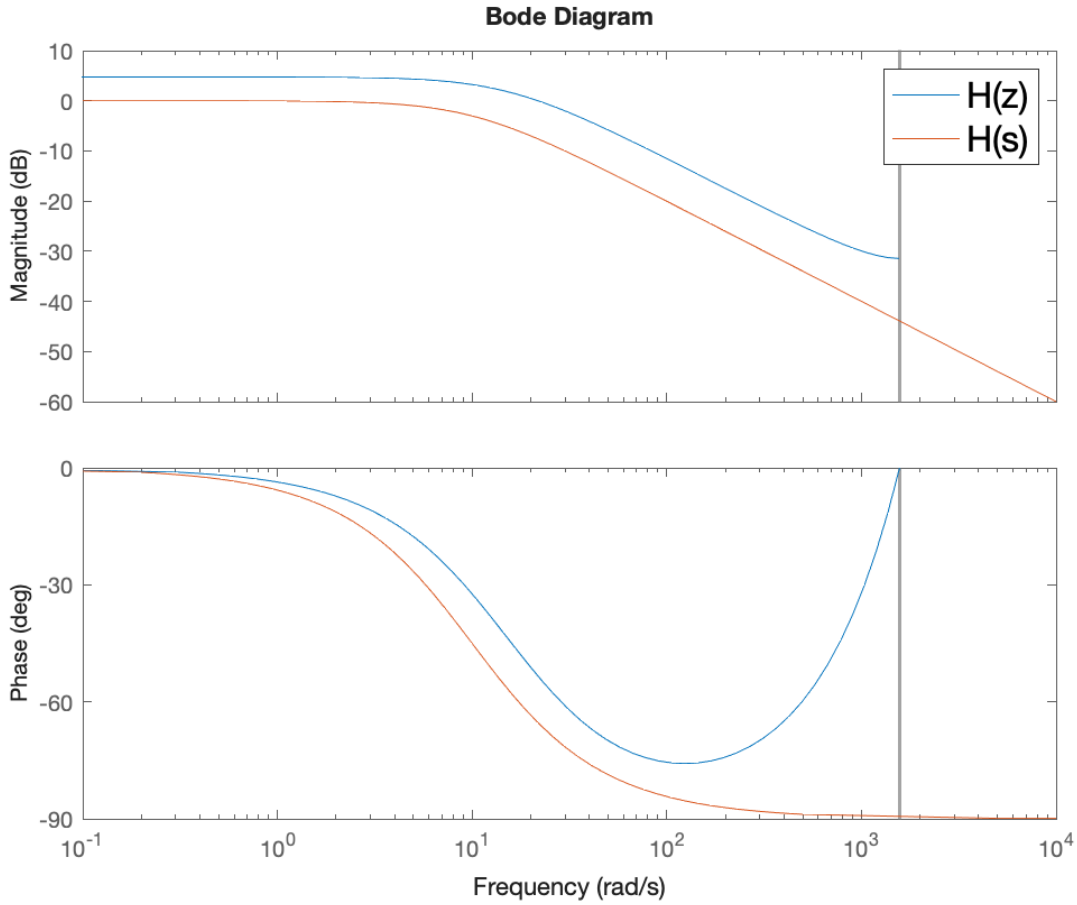


Figure 1: Bode comparison of H(z) and H(s) approximation.

Importantly, Fig. 1 shows that the $H(z)$ and $H(s)$ approximation both correspond to a phase of 45° at approximately 10 rad/s and achieve similar gain magnitudes. Figure 2 contains the block diagram for the Z single PD loop.

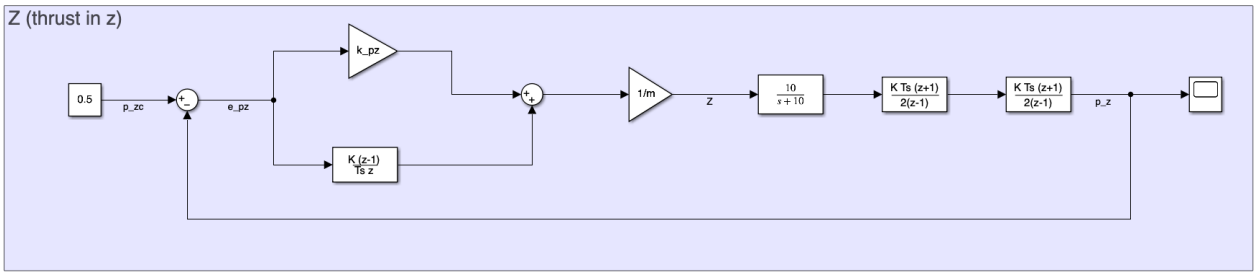


Figure 2: Z (thrust) single PD loop block diagram.

A constant input signal of $p_{zc} = 0.5$ is applied to achieve tracking of the setpoint. Tustin's transform is used for discrete-time integration. Figure 3 contains the block diagram for the L cascaded PID/PD controller.

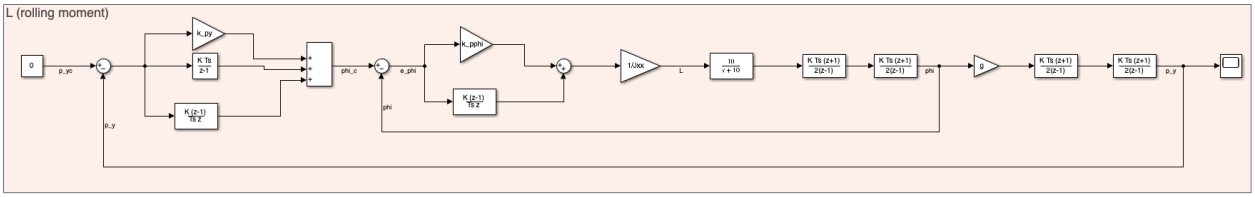


Figure 3: L (rolling moment) cascaded PID/PD block diagram.

The rolling moment controller tracks a setpoint command of $p_{yc} = 0$. The small angles approximation of $\sin\phi \approx \phi$ is used to achieve $\ddot{p}_y = g\phi$ when transferring from the inner to outer loop. Figure 4 contains the block diagram for the M cascaded PID/PD controller.

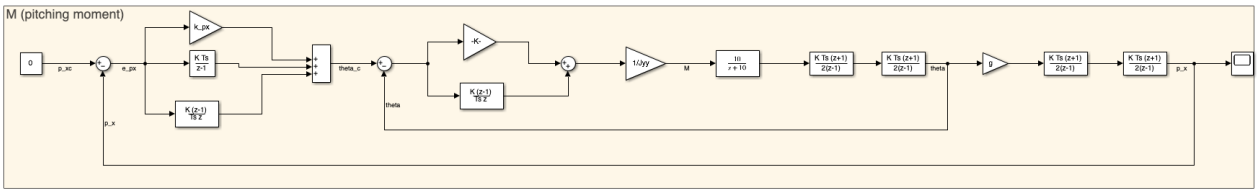


Figure 4: M (pitching moment) cascaded PID/PD block diagram.

The pitching moment controller tracks a setpoint command of $p_{xc} = 0$. Again, the small angles approximation of $\sin\theta \approx \theta$ is used to achieve $\ddot{p}_x = g\theta$ when transferring from the inner to outer loop. Fig. 5 contains the block diagram for the N single PD loop.

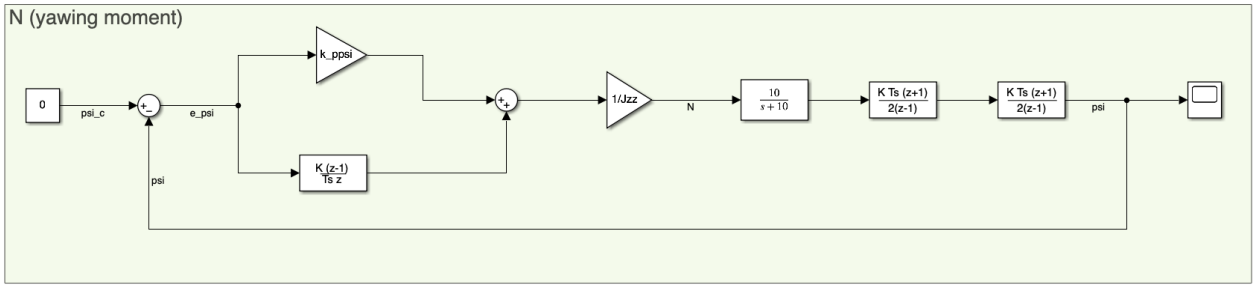


Figure 5: N (yawing moment) single PD loop block diagram.

The yawing moment controller tracks a setpoint command of $\psi_c = 0$. The cascaded PID/PD controller was implemented in Matlab using the code in Listing 2. Note that some modifications were added to simulate derivative filtering in addition to the above architecture.

Listing 2: MATLAB cascaded PID/PD controller.

```
function [Z, L, M, N, integral_x, integral_y] = PID_controller(integral_x, integral_y, x, ref, m, g, filtered)
    Jxx = 16.571710 * 1e-6;
    Jyy = 16.655602 * 1e-6;
    Jzz = 29.261652 * 1e-6;
    dt = 0.002; % 500 Hz sampling rate

    % Z (force in z)
    zeta_z = 1.0;
    w_n_z = 3.0;
    % Gains
    k_pz = m*w_n_z^2;
    k_dz = m*2*zeta_z*w_n_z;

    % L ----------------------------------
    zeta_roll = 1.2;
    w_n_roll = 3.0;

    % inner loop gains
    k_pphi = Jxx*w_n_roll^2;
    k_dphi = Jxx*2*zeta_roll*w_n_roll;
    % outer loop gains
    zeta_y = 1.8;
    w_n_y = 0.4;

    k_py = 0.2;
    k_iy = 0.1;
    k_dy = 0.1;

    % M ----------------------------------
    zeta_pitch = 1.5;
    w_n_pitch = 3.0;
    % Gains
    k_ptheta = Jyy*w_n_pitch^2;
    k_dtheta = Jyy*2*zeta_pitch*w_n_pitch;
    % outer loop gains
    zeta_x = 1.8;
    w_n_x = 0.4;

    k_px = 0.2;
    k_ix = 0.1;
    k_dx = 0.1;
```

```
42
43      % N ----------------------------------
44      zeta_yaw = 1.2;
45      w_n_yaw = 2.5;
46      % Gains
47      k_ppsi = Jzz*w_n_yaw^2;
48      k_dpsi = Jzz*2*zeta_yaw*w_n_yaw;
49
50
51
52      % errors
53      e_x = ref(1) - x(1);
54      e_y = ref(2) - x(2);
55      e_z = ref(3) - x(3);
56      e_vx = ref(4) - x(4);
57      e_vy = ref(5) - x(5);
58      e_vz = ref(6) - x(6);
59
60      integral_x = integral_x + e_x*dt;
61      integral_y = integral_y + e_y*dt;
62
63      N_filter = 10; % derivative filtering
64      alpha = N_filter*dt / (1 + N_filter*dt);
65      filtered.e_vx_filtered = (1 - alpha)*filtered.e_vx_filtered + alpha *e_vx;
66      filtered.e_vy_filtered = (1 - alpha)*filtered.e_vy_filtered + alpha *e_vy;
67
68      % commanded angles
69      theta_c = k_px*e_x + k_ix*integral_x + k_dx*filtered.e_vx_filtered;
70      phi_c = -(k_py*e_y + k_iy*integral_y + k_dy*filtered.e_vy_filtered);
71
72      % Z thrust
73      Z = m*g + k_pz*e_z + k_dz*e_vz;
74
75      % Current states
76      phi = x(7);
77      theta = x(8);
78      psi = x(9);
79      p = x(10);
80      q = x(11);
81      r = x(12);
82
83      % attitude errors
84      e_phi = phi_c - phi;
85      e_theta = theta_c - theta;
86      e_psi = 0 - psi;
87
88      % angular rate errors
89      e_p = 0 - p;
90      e_q = 0 - q;
91      e_r = 0 - r;
92
93      % torque'n it
94      L = k_pphi*e_phi + k_dphi*e_p;
95      M = k_ptheta*e_theta + k_dtheta*e_q;
96      N = k_ppsi*e_psi + k_dpsi*e_r;
97  end
```

6 Simulated Stability Analysis

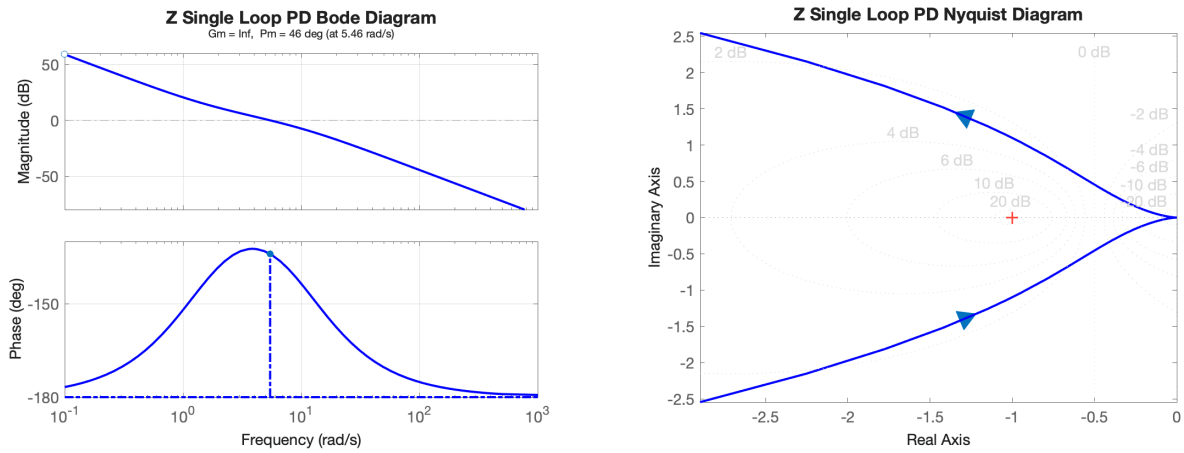
Tuning the cascaded PID/PD controller is still somewhat confusing and frustrating. We need to ensure that the bandwidth of the inner loop is much greater than the bandwidth of the outer loop to avoid phase lag. The inner loop must react quickly to attitude changes, while the outer loop should have a more delayed response to deal with steady-state position errors. Additionally, a few modifications to the outer loop PIDs were made to achieve the desired stability margins >6 dB of gain margin and $> 45°$ of phase margin.

6.1 Z (Thrust) Controller

The open-loop transfer function at the plant input is expressed as,

$$L_{oz} = \frac{k_{dz}s + k_{pz}}{ms^2} \cdot \frac{10}{s+10} \tag{14}$$

The additional $10/(s+10)$ term is the motor transfer function. Figure 6 contains the stability margins and Nyquist diagram for the Z single PD loop controller.



(a) Z Bode diagram.

(b) Z Nyquist diagram.

Figure 6: Z single loop PD frequency analysis.

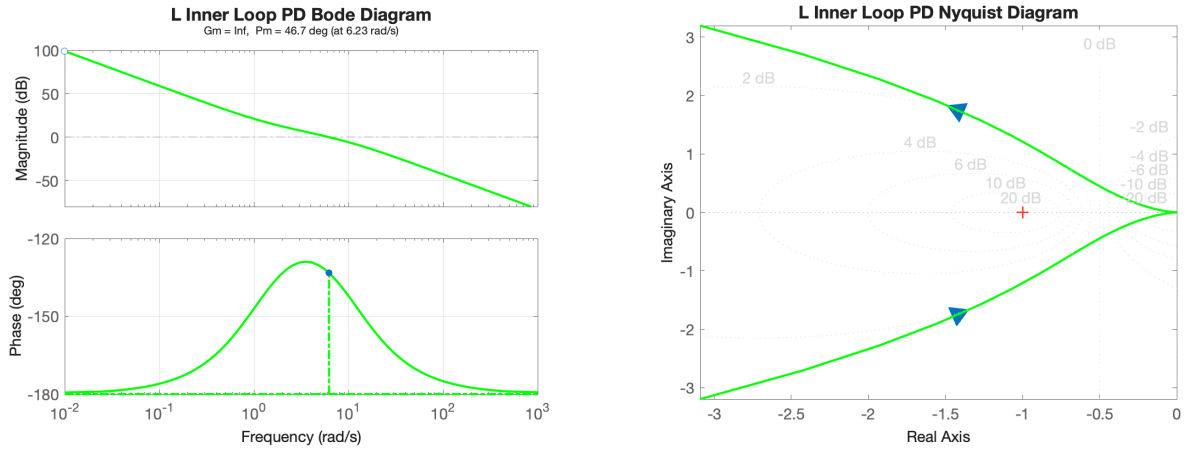
The resulting gain margin for the controller is $-\infty$ which implies that the gain could be infinitely decreased without affecting stability. The infinite gain margin also means that an incorrect gain could still stabilize the system. The resulting phase margin is 46°, which is just above the desired margin of $> 45°$. These values were achieved with $\omega_n = 3$ rads/s and $\zeta = 1.0$ (critical damping).

6.2 L (Rolling Moment) Controller

The open-loop transfer function for the inner loop at the plant input is expressed as,

$$L_{o\phi,\text{inner}} = \frac{k_{d\phi}s + k_{p\phi}}{J_{xx}s^2} \cdot \frac{10}{s+10} \tag{15}$$

The PD expression will essentially be the same for all PD loops aside from the gain adjustment. Figure 7 contains the stability margins and Nyquist diagram for the L inner loop PD controller.



(a) L inner loop Bode diagram.

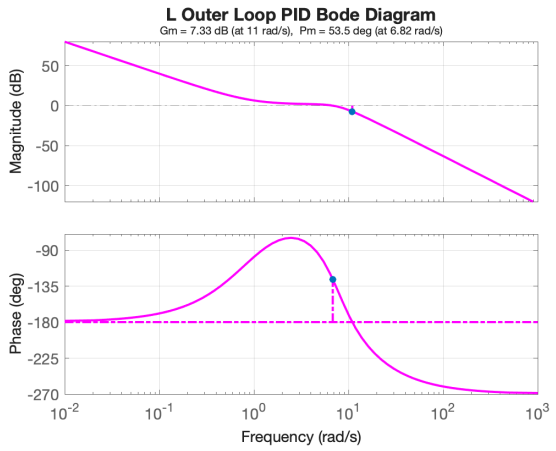
(b) L inner loop Nyquist diagram.

Figure 7: L inner loop PD frequency analysis.

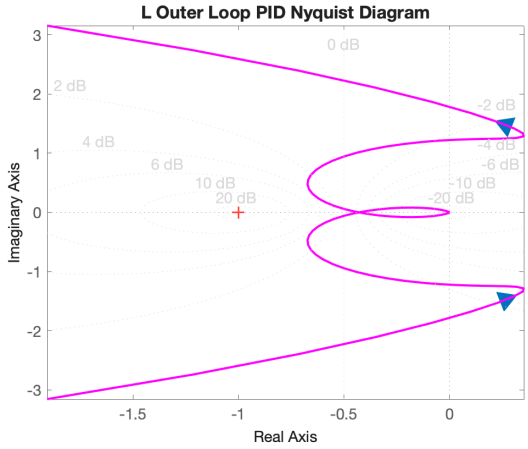
The gain margin is again $-\infty$ and the phase margin is just above the desired $> 45°$. These values were achieved with $\omega_n = 3$ rads/s and $\zeta = 1.2$ (overdamped). The open-loop transfer function for the outer loop at the plant input is expressed as,

$$L_{o\phi,\text{outer}} = g\frac{k_{dy}s^2 + k_{py}s + k_{iy}}{s(1+s/N)} \cdot L_{o\phi,\text{inner}}(I + L_{o\phi,\text{inner}})^{-1} \tag{16}$$

here, N is a filtering coefficient added for applying a low-pass filter to the derivative term. This derivative filter was needed due to instability from amplifying errors in the PID. Figure 8 contains the stability margins and Nyquist diagram for the L outer loop PID controller.



(a) L outer loop Bode diagram.



(b) L outer loop Nyquist diagram.

Figure 8: L outer loop PID frequency analysis.

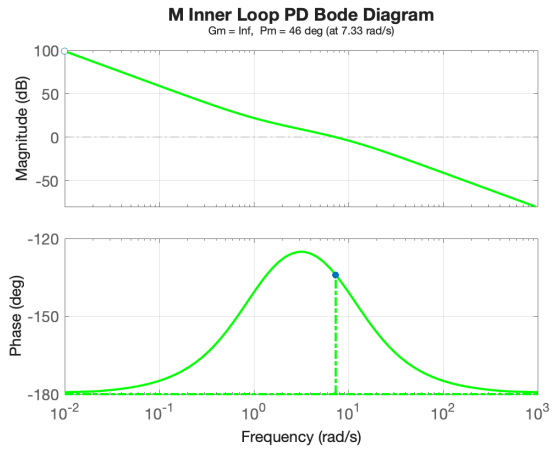
The resulting gain margin for the PID is 7.33 dB which is just above the desired > 6 dB. The phase margin for the PID is 53.3°. These margins are volatile and sensitive to small changes in ω_n and ζ which means these values may not translate to the hardware very well. Additionally, tuning via setting a bandwidth and damping coefficient became cumbersome, so the gains for k_{py}, k_{iy}, and k_{dy} were manually tuned.

6.3 M (Pitching Moment) Controller

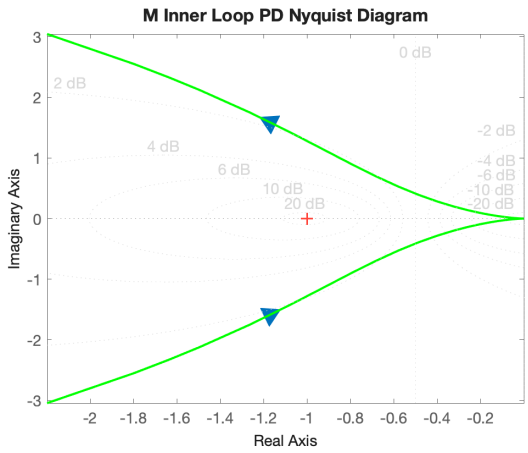
The open-loop transfer function for the inner loop at the plant input is expressed as,

$$L_{o\theta,\text{inner}} = \frac{k_{d\theta}s + k_{p\theta}}{J_{yy}s^2} \cdot \frac{10}{s+10} \tag{17}$$

Figure 9 contains the stability margins and Nyquist diagram for the M inner loop PD controller.



(a) M inner loop Bode diagram.



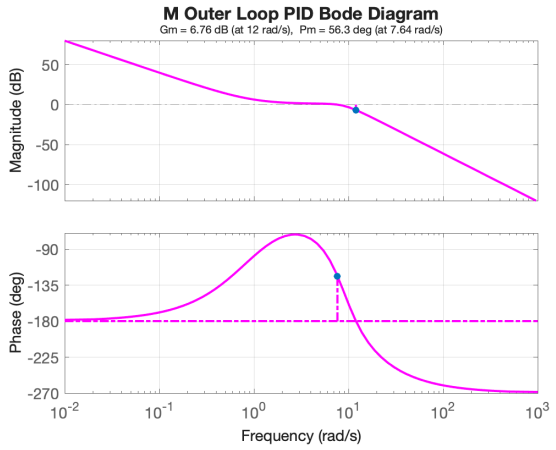
(b) M inner loop Nyquist diagram.

Figure 9: M inner loop PD frequency analysis.

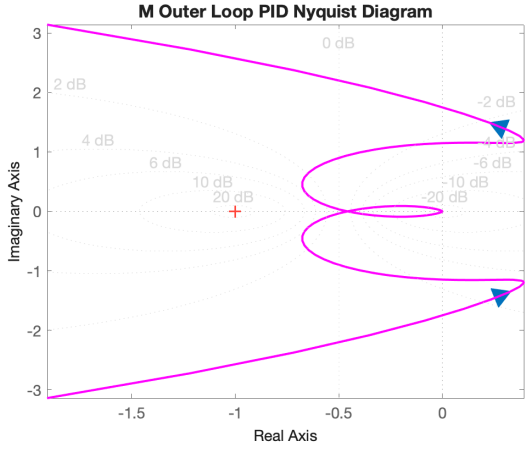
The gain margin is again $-\infty$ and the phase margin is just above the desired $> 45°$. These values were achieved with $\omega_n = 3$ rads/s and $\zeta = 1.2$ (overdamped). Using the same bandwidth and damping for both the rolling and pitching moments helps to maintain symmetry in the system. The difference in the moments of inertia is close to negligible. The open-loop transfer function for the M outer loop at the plant input is expressed as,

$$L_{o\theta,\text{outer}} = g\frac{k_{dx}s^2 + k_{px}s + k_{ix}}{s(1 + s/N)} \cdot L_{o\theta,\text{inner}}(I + L_{o\theta,\text{inner}})^{-1} \tag{18}$$

Here, we once again add a low-pass derivative filter to reduce amplification of PID instabilities. Figure 10 contains the stability margins and Nyquist diagram for the M outer loop PID controller.



(a) M outer loop Bode diagram.



(b) M outer loop Nyquist diagram.

Figure 10: M outer loop PID frequency analysis.

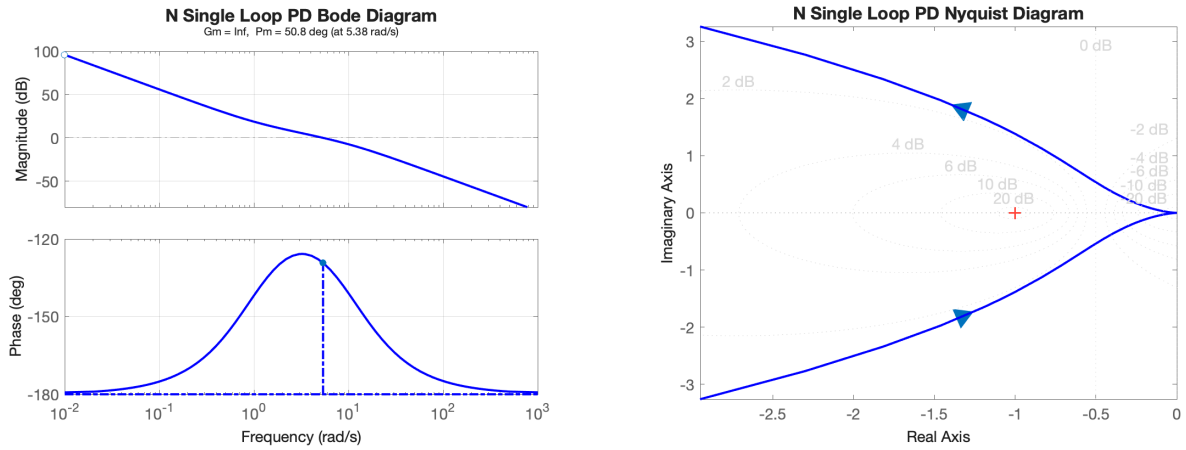
The gain margin for the PID is 6.76 dB which just barely clears the desired margin. The phase margin is 56.3°. These values were once again achieved by manually tuning the position error gains k_{px}, k_{ix}, and k_{dx}.

6.4 N (Yawing Moment) Controller

The open-loop transfer function at the plant input is expressed as,

$$L_{o\psi} = \frac{k_{d\psi}s + k_{p\psi}}{J_{zz}s^2} \cdot \frac{10}{s+10} \tag{19}$$

Figure 11 contains the stability margins and Nyquist diagram for the N single PD loop controller.



(a) N Bode diagram.

(b) N Nyquist diagram.

Figure 11: N single loop PD frequency analysis.

The resulting gain margin for the controller is, again, $-\infty$. The resulting phase margin is 50.8°, which achieves the desired margin of $> 45°$. These values were achieved with $\omega_n = 2.5$ rads/s and $\zeta = 1.2$ (overdamping). The complete stability analysis Matlab code is contained in Listing 3.

Listing 3: MATLAB tuning process.

```
clc; clear; close all;

m = 0.033; % mass
g = 9.81;
Jxx = 16.571710 * 1e-6;
Jyy = 16.655602 * 1e-6;
Jzz = 29.261652 * 1e-6;



% Z (force in z)
```

```
zeta_z = 1.0;
w_n_z = 3.0;
% Gains
k_pz = m*w_n_z^2
k_dz = m*2*zeta_z*w_n_z

% OL TF (plant input)
s = tf('s');
OLz = (k_dz*s + k_pz)/(m*s^2) * 10/(s+10);

[Gm, Pm] = margin(OLz);
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLz);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'b')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('Z Single Loop PD Bode Diagram',fontsize=16)
figure()
nyquist(OLz);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'b')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('Z Single Loop PD Nyquist Diagram',fontsize=16)

%%
z = tf('z',1/500);
H = 65536*(7.2345374e-8)/(1 - 0.9695401*z^-1)/0.09;
figure()
bode(H)
hold all;
s = tf('s');
G = 10/(s + 10);
bode(G)
legend('H(z)','H(s)',fontsize=16)

%%
% L (rolling moment)
clc;
zeta_roll = 1.2;
w_n_roll = 3.0;

% inner loop gains
k_pphi = Jxx*w_n_roll^2
k_dphi = Jxx*2*zeta_roll*w_n_roll

% OL TF (inner loop)
s = tf('s');
OLL = (k_pphi + k_dphi*s)/(Jxx*s^2) * 10/(s+10);

[Gm, Pm] = margin(OLL);
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLL);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'g')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('L Inner Loop PD Bode Diagram',fontsize=16)
```

```
figure()
nyquist(OLL);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'g')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('L Inner Loop PD Nyquist Diagram',fontsize=16)

%%
clc;
% outer loop gains
zeta_y = 0.8;
w_n_y = 1.4;


% k_py = w_n_y^2 / g
% k_iy = w_n_y^3 / (10*g)
% k_dy = 2*zeta_y*w_n_y / g
k_py = 0.2
k_iy = 0.1
k_dy = 0.1
N = 10;

% OL TF (outer loop)
CLL = minreal(OLL*inv(1 + OLL));
PID_tf = g*(k_dy*s^2 + k_py*s + k_iy) / (s*(1 + s/N)); % derivative filtering was necessary
OLy = PID_tf*CLL/s;

[Gm, Pm] = margin(OLy);
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLy);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'm')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('L Outer Loop PID Bode Diagram',fontsize=16)
figure()
nyquist(OLy);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'm')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('L Outer Loop PID Nyquist Diagram',fontsize=16)


%%

% M (pitching moment)
clc;
zeta_pitch = 1.5;
w_n_pitch = 3.0;
% Gains
k_ptheta = Jyy*w_n_pitch^2
k_dtheta = Jyy*2*zeta_pitch*w_n_pitch

% OL TF
s = tf('s');
OLM = (k_ptheta + k_dtheta*s)/(Jyy*s^2) * 10/(s + 10);


[Gm, Pm] = margin(OLM);
```

```
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLM);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'g')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('M Inner Loop PD Bode Diagram',fontsize=16)
figure()
nyquist(OLM);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'g')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('M Inner Loop PD Nyquist Diagram',fontsize=16)

%%
clc;
% outer loop gains
zeta_x = 0.9;
w_n_x = 3.0;

% k_px = w_n_x^2 / (g)
% k_ix = w_n_x^3 / (g)
% k_dx = 2*zeta_x*w_n_x / (g)
k_px = 0.2
k_ix = 0.1
k_dx = 0.1
N = 10;

% OL TF (outer loop) (g = 9.81)
CLM = minreal(OLM*inv(1 + OLM));
% OLx = g*(k_px*s + k_ix + k_dx*s^2)/s^3 * CLM/s;
PID_tf = g*(k_dx*s^2 + k_px*s + k_ix) / (s*(1 + s/N)); % derivative filtering was necessary
OLx = PID_tf*CLM/s;


[Gm, Pm] = margin(OLx);
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLx);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'm')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('M Outer Loop PID Bode Diagram',fontsize=16)
figure()
nyquist(OLx);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'm')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('M Outer Loop PID Nyquist Diagram',fontsize=16)


%%

% N (yawing moment)
clc;
zeta_yaw = 1.2;
w_n_yaw = 2.5;
```

```
% Gains
k_ppsi = Jzz*w_n_yaw^2
k_dpsi = Jzz*2*zeta_yaw*w_n_yaw


% OL TF
s = tf('s');
OLN = (k_ppsi + k_dpsi*s)/(Jzz*s^2) * 10/(s+10);

[Gm, Pm] = margin(OLN);
disp('Gain Margin (dB):'); disp(20*log10(Gm));
disp('Phase Margin (deg):'); disp(Pm);
figure();
margin(OLN);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'b')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('N Single Loop PD Bode Diagram',fontsize=16)
figure()
nyquist(OLN);
grid on;
box on;
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'b')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('N Single Loop PD Nyquist Diagram',fontsize=16)
```

7 Simulated Step Response

Figure 12 contains the simulated 0.5 m step response of the drone.

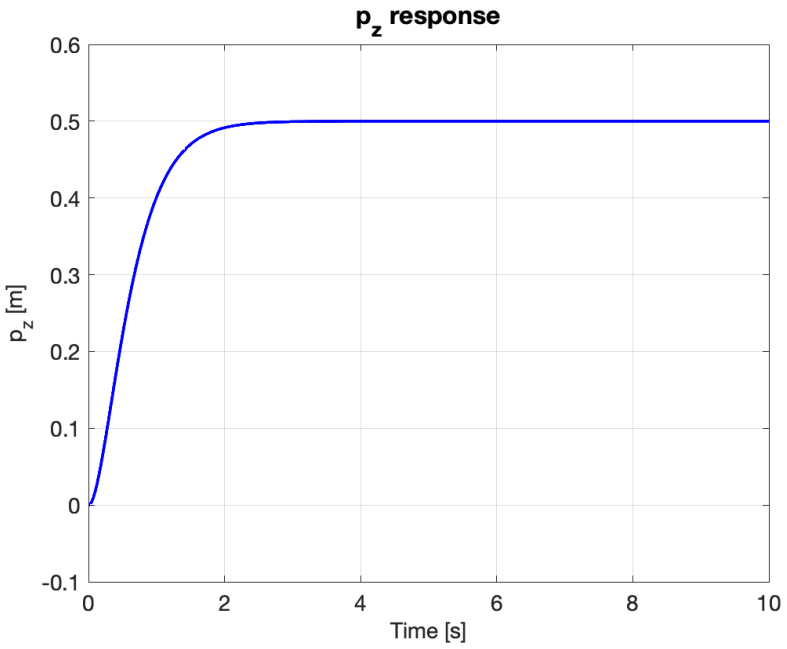
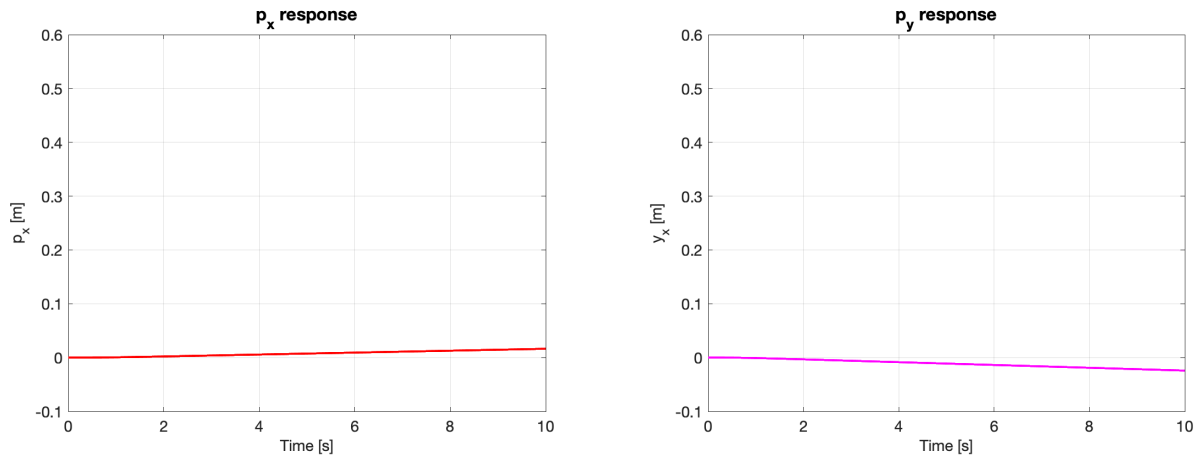


Figure 12: Simulated p_z step response of the drone.

The step response was simulated at 0.5 m using the cascaded PID/PD control system with the same gains from the stability analysis. The initial CrazyFlie 2.1 state estimation was imperfect, often with 0.5° to 1° of deviation. To simulate the effect of these imperfections, a small disturbance was applied to the initial conditions. Figure 13 contains the resulting p_x and p_y drift from deviations in the initial condition.



(a) p_x response. (b) p_y response.

Figure 13: Horizontal drift due to deviation in initial conditions.

Figure 13 demonstrates the unsuccessful control of the p_x and p_y drift. Adding and tuning the PID significant improved the drift rate, but was unable to completely mitigate the steady-state error. The Matlab step response program is contained in Listing 4.

Listing 4: MATLAB step response analysis.

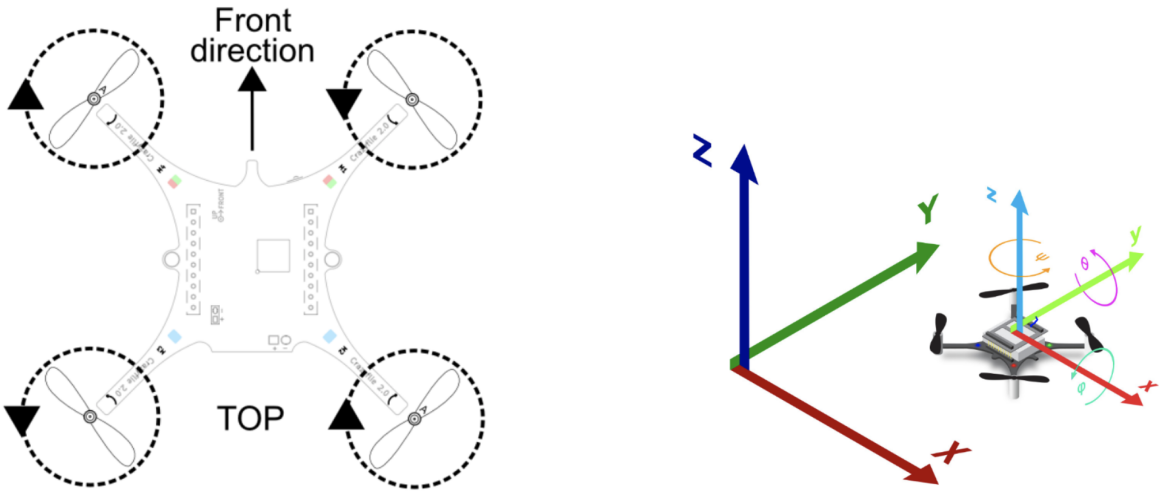
```
% tune gains and define trim controls
clc; clear; close all;
dt = 1/500; % 500 Hz
T = 10;
N = T/dt;

x = zeros(12,1) + [0;0;0;0;0;0;0.01;0.01;0.02;0;0;0];

ref = [0; 0; 0.5; 0; 0; 0]; % [x y z vx vy vz]
mass = 0.033;
g = 9.81;

integral_x = zeros(1,N+1);
integral_y = zeros(1,N+1);
filtered = struct('e_vx_prev', 0, 'e_vy_prev', 0, ...
                  'e_vx_filtered', 0, 'e_vy_filtered', 0);

t = 0:dt:T;
z_log = zeros(1,N+1); % make log
x_log = zeros(1,N+1); % make log
y_log = zeros(1,N+1); % make log
phi_log = zeros(1,N+1); % make log
theta_log = zeros(1,N+1); % make log
```

```
psi_log = zeros(1,N+1); % make log
pzdot_log = zeros(1,N+1); % make log
Z_log = zeros(1,N+1); % make log
L_log = zeros(1,N+1); % make log
M_log = zeros(1,N+1); % make log
N_log = zeros(1,N+1); % make log
% simulate
for i = 1:1:N

    % storing
    x_log(i) = x(1);
    y_log(i) = x(2);
    z_log(i) = x(3);
    phi_log(i) = x(7);
    theta_log(i) = x(8);
    psi_log(i) = x(9);


    [Z, L, M, N, integral_x(i), integral_y(i)] = PID_controller(integral_x(i), integral_y(i), ...
                                                                x, ref, mass, g, filtered);
    Z_log(i) = Z;
    L_log(i) = L;
    M_log(i) = M;
    N_log(i) = N;

    % needed states
    phi = x(7);
    theta = x(8);
    psi = x(9);
    p = x(10);
    q = x(11);
    r = x(12);

    % BODY FRAME VELOCITIES
    R = rotation_matrix(phi, theta, psi);
    uvw = R'*x(4:6);


    % make dynamics
    xdot = make_dynamics(uvw(1), uvw(2), uvw(3), phi, theta, psi, ...
                         p, q, r, Z, L, M, N);

    % oilurr
    x = x + xdot*dt;
end

x_log(end) = x(1);
y_log(end) = x(2);
z_log(end) = x(3);


phi_log(end) = x(7);
theta_log(end) = x(8);
psi_log(end) = x(9);


figure();
plot(t, z_log, 'b','LineWidth',2);
xlabel('Time [s]');
ylabel('p_z [m]');
ylim([-0.1,0.6])
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'b')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
```

```
title('p_z response',FontSize=16);
grid on;
box on;

figure()
plot(t, x_log,'r', 'LineWidth',2);
xlabel('Time [s]');
ylabel('p_x [m]');
ylim([-0.1,0.6])
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'r')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('p_x response',fontsize=16);
grid on;
box on;

figure()
plot(t, y_log,'m','LineWidth',2);
xlabel('Time [s]');
ylabel('y_x [m]');
ylim([-0.1,0.6])
xlim([0,10])
set(findall(gcf, 'type', 'line'), 'linewidth',2, 'color', 'm')
set(findall(gcf, '-property', 'FontSize'), 'FontSize', 14);
title('p_y response',fontsize=16);
grid on;
box on;

function R = rotation_matrix(phi, theta, psi)
    Rz = [cos(psi), -sin(psi), 0;
          sin(psi), cos(psi), 0;
          0, 0, 1];

    Ry = [cos(theta), 0, sin(theta);
          0, 1, 0;
          -sin(theta), 0, cos(theta)];

    Rx = [1, 0, 0;
          0, cos(phi), -sin(phi);
          0, sin(phi), cos(phi)];

    R = Rz*Ry*Rx;
end
```

8 The Mixer

The CrazyFlie 2.1 behavior conventions are shown Fig. 14.



(a) Propeller conventions.

(b) Drone frames.

Figure 14: CrazyFlie 2.1 conventions.

The mixer is then derived by analyzing the behavior of the drone. Essentially, the desired behavior is as follows,

$$\begin{aligned}
\text{M1 thrust} &= \text{yaw} - \text{pitch} - \text{roll} + \text{z thrust} \\
\text{M2 thrust} &= \text{pitch} - \text{roll} - \text{yaw} + \text{z thrust} \\
\text{M3 thrust} &= \text{roll} - \text{pitch} - \text{yaw} + \text{z thrust} \\
\text{M4 thrust} &= \text{roll} - \text{pitch} - \text{yaw} + \text{z thrust}
\end{aligned}$$

That is to say, when we apply a Z thrust, all motors should actuate to achieve positive thrust. When we apply an L roll, M1 and M2 should actuate positively and M3 and M4 should actuate negatively to achieve a positive roll. When we apply an M pitch, M2 and M3 should actuate positively and M1 and M4 should actuate negatively to achieve a positive pitch. Finally, when we apply an N yaw, M1 and M3 should actuate positively and M2 and M4 should actuate negatively to achieve a positive yaw. The normalized mixer matrix is then expressed as follows,

$$\text{Mixer} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{20}$$

The thrust is then solved for by taking the inverse of the Mixer,

$$\begin{bmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ n_4^2 \end{bmatrix} = \text{Mixer}^{-1} \begin{bmatrix} Z \\ L \\ M \\ N \end{bmatrix} \rho C_T D^5 \tag{21}$$

$$\text{where,} \quad \text{Mixer}^{-1} = 0.25 \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \tag{22}$$

and n^2 is the propeller rev/s. Note, that in general, the propeller thrust equation is expressed in terms of the thrust coefficient, C_T, as,

$$T = \rho C_T D^5 n^2 \tag{23}$$

The density ρ, diameter D^5, and C_T, however, are contained within the given empirical normalized PWM equation from Eq. 3. We solve the for the motor PWM signal by simply using the quadratic formula. Note, however, that we must apply 16 bit unsigned integer of 65,536 to properly scale the PWM signal to the motor.

9 CrazyFlie 2.1 Implementation

The mixer and cascaded PID/PD control system were implemented on the CrazyFlie 2.1 firmware with moderate success. We implemented the mixer and controller by editing the `power_distribution_quadrotor.c` and `controller_pp.c`. The CrazyFlie 2.1 was configured for pole-placement. Several modifications beyond the explained control system architecture were added to the code in an attempt to achieve successful hovering. The Z thrust was clamped to prevent saturation, which was a huge problem during early testing of the control system. Angle wrapping was added to mitigate for the excessive initial yaw state errors. Finally, trim constants were also added to the rolling, pitching, and yawing moments due to experimental observations. There is was likely a significant imperfection in the quadcopter IMU that was used. Additionally, the integral terms were constrained to prevent integral wind-up.

9.1 Mixer

The mixer was implemented to the `power_distribution_quadrotor.c` by adding the C functions contained in Listing 5.

Listing 5: Functions added to power_distribution_quadrotor.c.

```
1 // convert thrust [N] to pwm
2 float thrustToPWM(float thrust) {
```

```
    // just in case there is for some reason negative thrust
    // if (thrust <= 0) return 0;

    // quadratic solution for the pwm
    // float discriminant = pwmToThrustB*pwmToThrustB + 4*pwmToThrustA*thrust;
    // return (-pwmToThrustB + sqrtf(discriminant))/(2*pwmToThrustA);
    const float thrust_max = 0.159f; // Max thrust before PWM=1.0
    if (thrust <= 0) return 0.0f;
    if (thrust > thrust_max) thrust = thrust_max; // Saturate
    float discriminant = pwmToThrustB*pwmToThrustB + 4*pwmToThrustA*thrust;
    return (-pwmToThrustB + sqrtf(discriminant)) / (2*pwmToThrustA);
}
static void powerDistributionForceTorque(const control_t *control, motors_thrust_uncapped_t* motorThrustUncapped)
      {
  // TODO: put mixer code below
  // const float mixer[4][4] = {
  //       {1.0f, 1.0f, 1.0f, 1.0f}, // Thrust
  //       {-1.0f, -1.0f, 1.0f, 1.0f}, // Roll
  //       {1.0f, -1.0f, -1.0f, 1.0f}, // Pitch
  //       {1.0f, -1.0f, 1.0f, -1.0f}  // Yaw
  //   };

  const float mixer_inv[4][4] = {
      {0.25f, -0.25f, 0.25f, 0.25f},
      {0.25f, -0.25f, -0.25f, -0.25f},
      {0.25f, 0.25f, -0.25f, 0.25f},
      {0.25f, 0.25f,  0.25f, -0.25f}
  };
  float Z = control->thrustSi;
  float L = control->torqueX;
  float M = control->torqueY;
  float N = control->torqueZ;
  // float Z = 0.4f; // [N]
  // float L = 0.0f;
  // float M = 0.0f;
  // float N = 0.0f;

  float n[4]; // motor thrusts

  for (int motor = 0; motor < STABILIZER_NR_OF_MOTORS; motor++) {
        n[motor] = mixer_inv[motor][0]*Z + mixer_inv[motor][1]*L + mixer_inv[motor][2]*M + mixer_inv[motor][3]*N;

        // convert thrust to normalized PWM
        float pwm_normalized = thrustToPWM(n[motor]);

        // scale PWM
        uint16_t pwm_absolute = (uint16_t)(pwm_normalized*UINT16_MAX);
        motorThrustUncapped->list[motor] = pwm_absolute;

        // safety clamp
        motorThrustUncapped->list[motor] = constrain(motorThrustUncapped->list[motor],0,UINT16_MAX);
    }
}
```

9.2 Control System

The cascaded PID/PD controller was implemented to the pole-placement `controller_pp.c` code by adding the C functions contained in Listing 6.

Listing 6: Functions added to controller_pp.c.

```c
typedef struct {
    float integral_x;
    float integral_y;
    float prev_e_vx;  //  previous velocity error
    float prev_e_vy;
    float filtered_dx; // low-pass derivative filtering
    float filtered_dy;
} PID_State;
void controllerPP(control_t *control, const setpoint_t *setpoint,
                                      const sensorData_t *sensors,
                                      const state_t *state,
                                      const stabilizerStep_t stabilizerStep) {
    // limits controller rate to 500Hz.
    if (!RATE_DO_EXECUTE(UPDATE_RATE, stabilizerStep)) {
        return;
    }
    // TODO: put your controller code below
    static PID_State pid = {0};
    if (setpoint->mode.z == modeDisable) {
        // Disarm motors when the drone is on the ground/landed
        control->thrustSi = 0.0f;
        control->torqueX = 0.0f;
        control->torqueY = 0.0f;
        control->torqueZ = 0.0f;

        pid.integral_x = 0.0f;
        pid.integral_y = 0.0f;
    } else {
        // // TODO: send command to mixer below
        // Mass and gravity
        const float m = 0.033f;  // Crazyflie mass in kg
        const float g = 9.81f;
        //const float pi = 22.0f/7.0f;
        const float dt = 0.002f; // 500 Hz update for integral control
        const float N = 10.0f;  // filter coefficient for derivative
        const float alpha = N*dt / (1.0f + N*dt); // time constant exp smoothing

        const float roll_trim = 0.01f; // 0.02f works pretty well
        const float pitch_trim = 0.03f; // 0.02f works pretty well
        const float yaw_trim = 0.02f; // 0.02f works pretty well
        // const float roll_trim = 0.0f; // 0.02f works pretty well
        // const float pitch_trim = 0.0f; // 0.02f works pretty well
        // const float yaw_trim = 0.0f; // 0.02f works pretty well


        // Controller constants
        const float Jxx = 16.571710e-6f;
        const float Jyy = 16.655602e-6f;
        const float Jzz = 29.261652e-6f;

        const float zeta_z = 0.7f;
        // const float zeta_x = 0.8f;
        // const float zeta_y = 0.8f;

        const float zeta_roll = 2.0;
        const float zeta_pitch = 1.7f;
        const float zeta_yaw = 1.5f;

        const float w_n_z = 3.0f; // rads/s
        // const float w_n_x = 3.0f;
        // const float w_n_y = 3.0f;

        const float w_n_roll = 6.0f; // a bit of asymmetry was needed (roll oscillations)
```

```
64          const float w_n_pitch = 8.0f;
65          const float w_n_yaw = 6.0f;
66
67          // Position gains
68          const float k_px = 0.2f;
69          const float k_py = 0.2f;
70          const float k_pz = m*w_n_z*w_n_z;
71
72          // Velocity gains
73          const float k_dx = 0.25f;
74          const float k_dy = 0.25f;
75          const float k_dz = m*2.0f*zeta_z*w_n_z*w_n_z;
76
77          // Attitude gains
78          const float k_pphi = Jxx*w_n_roll*w_n_roll;
79          const float k_ptheta = Jyy*w_n_pitch*w_n_pitch;
80          const float k_ppsi = Jzz*w_n_yaw*w_n_yaw;
81
82          // Rate gains
83          const float k_dphi = Jxx*2.0f*zeta_roll*w_n_roll;
84          const float k_dtheta = Jyy*2.0f*zeta_pitch*w_n_pitch;
85          const float k_dpsi = Jzz*2.0f*zeta_yaw*w_n_yaw;
86
87          // adding PID to outer loop cascades for roll and pitch
88          // const float k_ix = w_n_x*w_n_x*w_n_x/(10.0f*g);
89          // const float k_iy = w_n_y*w_n_y*w_n_y/(10.0f*g);
90          const float k_ix = 0.05;
91          const float k_iy = 0.05;
92
93
94
95          // Position errors
96          float e_x = 0.0f - state->position.x;
97          float e_y = 0.0f - state->position.y;
98          float e_z = 0.5f - state->position.z;
99
100         // integral
101         pid.integral_x += e_x*dt;
102         pid.integral_y += e_y*dt;
103
104         // anti-windup
105         pid.integral_x = constrain(pid.integral_x, -0.5f, 0.5f);
106         pid.integral_y = constrain(pid.integral_y, -0.5f, 0.5f);
107
108
109         // Velocity errors
110         float e_vx = 0.0f - state->velocity.x;
111         float e_vy = 0.0f - state->velocity.y;
112         float e_vz = 0.0f - state->velocity.z;
113
114         // derivative filtering
115         pid.filtered_dx = (1.0f - alpha)*pid.filtered_dx + alpha*e_vx;
116         pid.filtered_dy = (1.0f - alpha)*pid.filtered_dy + alpha*e_vy;
117
118
119         // Commanded angles (convert position error to desired attitude)
120         float theta_c = -(k_px*e_x + k_ix*pid.integral_x +k_dx*pid.filtered_dx); //pitch command
121         float phi_c = k_py*e_y + k_iy*pid.integral_y + k_dy*pid.filtered_dy;  // roll command
122
123         // Total thrust (Z)
124         control->thrustSi = constrain(m*g + k_pz*e_z + k_dz*e_vz, 0.1f, 0.5f); // prevent saturated
125
126
```

```
        // Attitude errors
        float e_phi = atan2f(sinf(phi_c - state->attitude.roll),cosf(phi_c - state->attitude.roll));
        float e_theta = atan2f(sinf(theta_c - state->attitude.pitch),cosf(theta_c - state->attitude.pitch));
        float e_psi = atan2f(sinf(0 - state->attitude.yaw),cosf(0 - state->attitude.yaw));


        // Angular rate errors
        float e_p = 0.0f - sensors->gyro.x;
        float e_q = 0.0f - sensors->gyro.y;
        float e_r = 0.0f - sensors->gyro.z;

        // Torques (L, M, N)
        control->torqueX = k_pphi*e_phi + k_dphi*e_p - roll_trim;
        control->torqueY = -(k_ptheta*e_theta + k_dtheta*e_q) + pitch_trim;
        control->torqueZ = -(k_ppsi*e_psi + k_dpsi*e_r) + yaw_trim;

        // // Temporary test code
        // control->thrustSi = 0.0f;
        // control->torqueX = 0.0f;
        // control->torqueY = 0.0f;
        // control->torqueZ = 0.0f;

    }
    control->controlMode = controlModeForceTorque; // use custom mixer
}
```

10 Results & Discussion

Table 1 contains the final gains and parameters from the Matlab simulation and CrazyFlie 2.1 test flight.

Table 1: Simulation and experimental gains.

Parameter/Gain	Simulation	CrazyFlie 2.1
k_{pz} [N·rads^2/m]	0.297	0.297
k_{dz} [N·rads·s/m]	0.198	0.198
$k_{p\phi}$ [N·m·rads^2]	1.4194×10^{-4}	5.96×10^{-4}
$k_{d\phi}$ [N·m·rads·s]	1.1932×10^{-4}	2.98×10^{-4}
$k_{p\theta}$ [N·m·rads^2]	1.499×10^{-4}	1.066×10^{-3}
$k_{d\theta}$ [N·m·rads·s]	1.499×10^{-4}	3.198×10^{-4}
$k_{p\psi}$ [N·m·rads^2]	1.8289×10^{-4}	1.053×10^{-3}
$k_{d\psi}$ [N·m·rads·s]	1.7557×10^{-4}	5.267×10^{-4}
k_{px} [rads^2/m]	0.2	0.2
k_{ix} [rads^3/m·s]	0.1	0.05
k_{dx} [rads·s/m]	0.1	0.25
k_{py} [rads^2/m]	0.2	0.2
k_{iy} [rads^3/m·s]	0.1	0.05
k_{dy} [rads·s/m]	0.1	0.25
N_{filter} [rads/s]	10	10

It's unlikely that system identification is completely necessary for each individual quadcopter, but given the small size and cheap components there may be some variation between models that affects the motor transfer function. Such discrepancies could result in the significant tuning differences between the Matlab simulation and experimental implementation. There are also, of course, numerous assumptions that were made to simplify the dynamics such as neglecting inertia matrix cross terms. While these terms were small compared to the diagonal terms, they were likely not completely negligible.

Most challenging, however, was tuning the PID to ensure that the inner loop bandwidth with approximately 5 to 10× faster than the outer loop bandwidth. The drone was able to hover at 0.5 m, but there were significant oscillations in the p_x and p_y states. Additionally, there were aggressive roll oscillations occurring at approximately 5 Hz. These oscillations were attempted to be reduced by increasing the damping coefficient and decreasing the inner loop bandwidth, but reducing the bandwidth too much caused a phase lag in the cascaded design. Removing the oscillations caused the quadcopter to drift significantly.

For a future implementation, I would choose an LQR controller to find the optimal position error gains for the outer loop. Additionally, tuning an LQR controller likely would have been simpler and easier, but where is the fun in that? The CrazyFlie 2.1 hardware would likely be able to handle the computational expense of LQR. LQR would likely reduce the position steady-state error that plagued the control system. Position steady-state errors were amplified significantly by oscillations in roll and pitch and would likely have needed faster actuation to make recovery maneuvers. Limiting the bandwidth of the PID by cascading the controller in an outer loop was likely not the best idea.

The control design evolved over the course of the project. Originally, my team simply wanted to implement a cascaded P controller. This design was later expanded to cascaded PD control. The cascaded PD control was the first instance of successful hovering. However, I personally, was still dissatisfied with the position error drift and overambitiously decided to convert the PD outer loop to PID the night before the project deadline. The drone, was able to fly, but significant tuning improvements could still be made to the drone. Lesson learned.