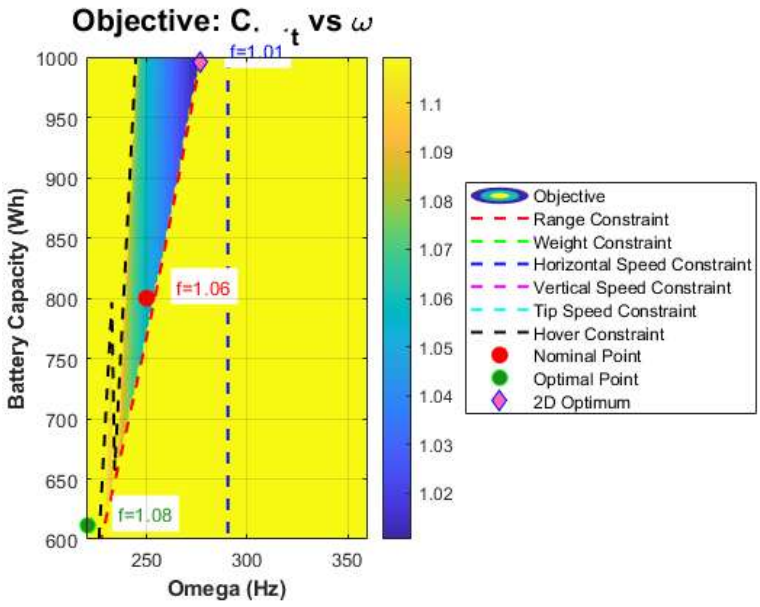
Contents


```
%Tyler & Ben

clear;
clc;
close all;
```

Nominal & Optimal Design Variables(50/50)

```
alpha_nom = 4;
d_nom = 0.7;
NR_nom = 6;
omega_nom = 250;
C_batt_nom = 800;

alpha_opt= 2.79;
d_opt = 0.8526;
NR_opt = 4;
omega_opt = 220.765;
C_batt_opt = 611.3964;

% Grid settings
n = 100;
alpha_range = linspace(1, 8, n);
d_range = linspace(0.65, 1, n);
omega_range = linspace(220, 360, n);
C_batt_range = linspace(600, 1000, n);
NR_range = linspace(3, 10, n);




colors = {'r', 'g', 'b', 'm', 'c', 'k'};



%%Plot 1: omega vs C_batt
[X, Y] = meshgrid(omega_range, C_batt_range);
Z = zeros(size(X));
gmat = zeros([size(X), 6]);

for i = 1:numel(X)
    x = [alpha_nom, d_nom, NR_nom, X(i), Y(i)];
    [fval, ~, ~] = objective_fun(x);
    [g, ~] = nonlincon(x);
    [row, col] = ind2sub(size(X), i);
    gmat(row, col, :) = g;

    if all(g <= 0)
        Z(i) = fval;
    else
        Z(i) = Inf;
    end
end

%Find 2D minimum
[min_val, idx] = min(Z(:));

omega_2D = X(idx);
C_batt_2D = Y(idx);


%Plotting
figure;
contourf(X, Y, Z, 30, 'LineColor', 'none'); hold on; colorbar
```

```
finite_Z = Z(isfinite(Z));
caxis([min(finite_Z), prctile(finite_Z, 95)]);
xlabel('Omega (Hz)', 'FontWeight', 'bold');
ylabel('Battery Capacity (Wh)', 'FontWeight', 'bold');
title('Objective: C_{batt} vs \omega ', 'FontSize', 16, 'FontWeight', 'bold');

for k = 1:6
    contour(X, Y, gmat(:,:,k), [0 0], '--', 'LineWidth', 1.5, 'LineColor', colors{k});
end

plot(omega_nom, C_batt_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
plot(omega_opt, C_batt_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);
plot(omega_2D, C_batt_2D, 'bd', 'MarkerFaceColor', [1 0.4 0.7], 'MarkerSize', 8);


plot(omega_nom, C_batt_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
[f_nom, ~, ~] = objective_fun([alpha_nom, d_nom, NR_nom, omega_nom, C_batt_nom]);
text(omega_nom + 15, C_batt_nom + 10, sprintf('f=%.2f', f_nom), ...
     'Color', 'r', 'FontSize', 10, 'BackgroundColor', 'w');

plot(omega_opt, C_batt_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);
[f_opt, ~, ~] = objective_fun([alpha_opt, d_opt, NR_opt, omega_opt, C_batt_opt]);
text(omega_opt + 15, C_batt_opt + 10, sprintf('f=%.2f', f_opt), ...
    'Color', [0.13, 0.55, 0.13], 'FontSize', 10, 'BackgroundColor', 'w');


plot(omega_2D, C_batt_2D, 'bd', 'MarkerFaceColor', [1 0.4 0.7], 'MarkerSize', 8);
[f_2D, ~, ~] = objective_fun([alpha_nom, d_nom, NR_nom, omega_2D, C_batt_2D]);
text(omega_2D + 15, C_batt_2D + 10, sprintf('f=%.2f', f_2D), ...
    'Color', 'b', 'FontSize', 10, 'BackgroundColor', 'w');




legend('Objective', ...
       'Range Constraint','Weight Constraint','Horizontal Speed Constraint','Vertical Speed Constraint','Tip Speed Constraint','Hover Constraint', ...
       'Nominal Point', 'Optimal Point', '2D Optimum', 'Location', 'eastoutside');
grid on;
```



Plot 2: d vs NR

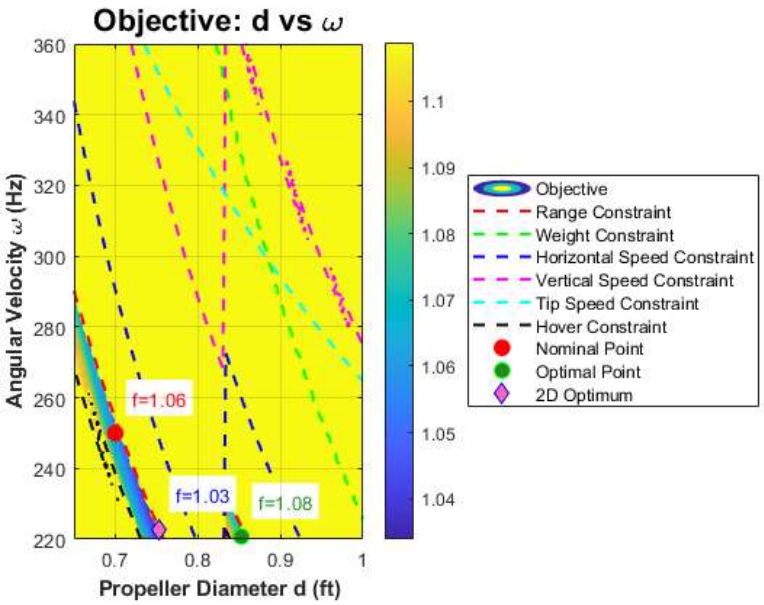
```
[X, Y] = meshgrid(d_range, NR_range);
Z = zeros(size(X));
gmat = zeros([size(X), 6]);

%Loop through grid
for i = 1:numel(X)
    x = [alpha_nom, X(i), Y(i), omega_nom, C_batt_nom];
    [fval, ~, ~] = objective_fun(x);
    [g, ~] = nonlincon(x);
    [row, col] = ind2sub(size(X), i);
    gmat(row, col, :) = g;
```

```
    if all(g <= 0)
        Z(i) = fval;
    else
        Z(i) = Inf;
    end
end

%Find 2D minimum
[min_val, idx] = min(Z(:));

d_2D = X(idx);
NR_2D = Y(idx);


%Plotting
figure;
contourf(X, Y, Z, 30, 'LineColor', 'none'); hold on; colorbar
finite_Z = Z(isfinite(Z));
caxis([min(finite_Z), prctile(finite_Z, 95)]);
xlabel('Propeller Diameter d (ft)', 'FontWeight', 'bold');
ylabel('Number of Rotors NR', 'FontWeight', 'bold');
title('Objective: NR vs d', 'FontSize', 16, 'FontWeight', 'bold');

for k = 1:6
    contour(X, Y, gmat(:,:,k), [0 0], '--', 'LineWidth', 1.5, 'LineColor', colors{k});
end

plot(d_nom, NR_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
plot(d_opt, NR_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);

plot(d_2D, NR_2D, 'bd', 'MarkerFaceColor',      'b', 'MarkerSize', 8);



plot(d_nom, NR_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
[f_nom, ~, ~] = objective_fun([alpha_nom, d_nom, NR_nom, omega_nom, C_batt_nom]);
text(d_nom + 0.02, NR_nom + 0.2, sprintf('f=%.2f', f_nom), ...
    'Color', 'r', 'FontSize', 10, 'BackgroundColor', 'w');

plot(d_opt, NR_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);
[f_opt, ~, ~] = objective_fun([alpha_opt, d_opt, NR_opt, omega_opt, C_batt_opt]);
text(d_opt + 0.02, NR_opt + 0.2, sprintf('f=%.2f', f_opt), ...
    'Color', [0.13, 0.55, 0.13], 'FontSize', 10, 'BackgroundColor', 'w');

plot(d_2D, NR_2D, 'bd', 'MarkerFaceColor', [1 0.4 0.7], 'MarkerSize', 8);
[f_2D, ~, ~] = objective_fun([alpha_nom, d_2D, NR_2D, omega_nom, C_batt_nom]);
text(d_2D + 0.02, NR_2D + 0.2, sprintf('f=%.2f', f_2D), ...
    'Color', 'b', 'FontSize', 10, 'BackgroundColor', 'w');


legend('Objective', ...
       'Range Constraint','Weight Constraint','Horizontal Speed Constraint','Vertical Speed Constraint','Tip Speed Constraint','Hover Constraint', ...
       'Nominal Point', 'Optimal Point', '2D Optimum', 'Location', 'eastoutside');
grid on;
```

Objective Function 1

```
function [f time noise] = objective_fun(x)

    alpha = x(1);    %Tilt Angle(0 degrees means angled straight up)
    d = x(2);        %Rotor Diameter (ft)
    NR = x(3);       %Number of rotors (Discrete variable)
    omega = x(4);    %Angular Velocity (hz)
    C_batt = x(5);   %Battery Capacity (Wh)
    payload_weight = 14.4; %Assumes 3 pizzas and a 2 liter soda
    Wbattery = 0.013 * C_batt;  % battery weight per Wh
    Wempty = 8;  %Empty Weight of Drone
    A = 2.0;    %Cross Sectional Area of Drone
    L = 2.5;    %Length of Drone
    W = 2.5;    %Width of Drone
    g = 32.174; %Gravity
    h = 200;    %Operational Height, needs to clear trees
    D = 10 * 5280; %Farthest possible range
    rho = 0.00223582; %Blacksburg air density

    Wprop = calc_prop_weight(d, NR); %Computes the weight of the prop
    Cd = get_cd(d, omega); %Compute the coefficient of drag
    Ct = get_ct(d, omega); %Computes the coefficient of thrust
    [T_total, T_vertical, T_forward] = get_thrust(NR, Ct, rho, omega, d, alpha); %Gets thrust components
```

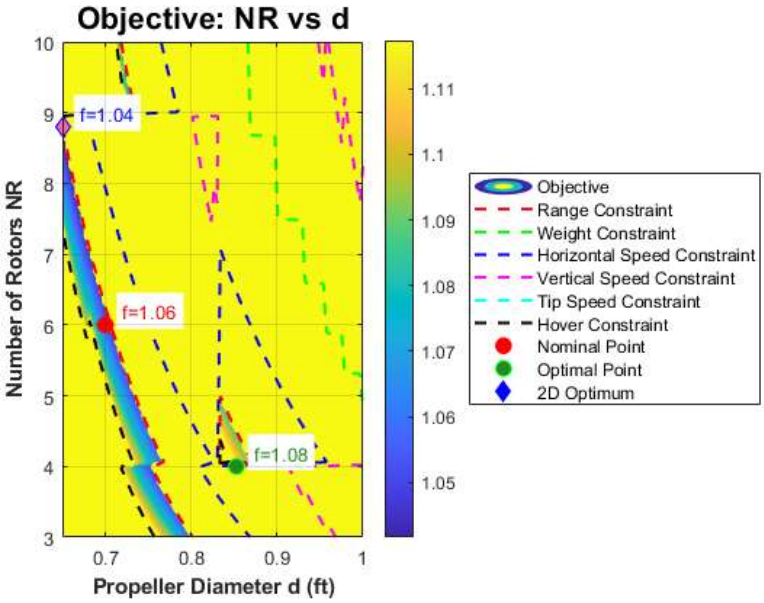
```
    [motor_count, motor_weight] = motor_count_weight(NR, omega, T_total, d); %Determines the motors needed
    W_total = get_total_weight(Wempty, payload_weight, Wbattery, Wprop + motor_weight); %Computes drones weight


    spacing = get_spacing(NR, d, W, L); %Determines spacing between rotors(s)
    T_vertical = account_for_coupling(T_vertical, spacing); %Vertical thrust reduction from coupling
    T_forward = account_for_coupling(T_forward, spacing); %Horizontal thrust reduction from coupling
    eta = calculate_efficiency(Ct, Cd); %Efficiency from momentum theory
    [V_forward, V_up] = compute_velo(T_forward, Cd, rho, A, g, h, T_vertical, W_total); %Computes velocity components

    alpha_climb = 0; %Alpha is zero for climb and descent
    grav = 32.174;
    [T_total_0] = get_thrust(NR, Ct, rho, omega, d, alpha_climb); %Thurst is computed for vertical segment
    T_vertical_0 = account_for_coupling(T_total_0, spacing); %Vertical is accounted for coupling
    T_forward_0 = 0;
    [~, Vu_0] = compute_velo(T_forward_0, Cd, rho, A, grav, h, T_vertical_0, W_total); %Computes velocity for climb

    time = (h / Vu_0) + (h / sqrt(2 * grav * h)) + (D / V_forward);



    time = time / 365; %Normalize by time to deliver;
    noise = 10*log10(NR*(pi*d*omega)^2)/65; %normalize noise

    w_noise = 0.5; %Weighting
    w_time = 0.5; %Weighting


    f = w_noise*noise + w_time*time; %normalized weighted objective function


end
```



Constraints

```
function [g, ceq] = nonlincon(x)
   alpha = x(1);    %Tilt Angle(0 degrees means angled straight up)
    d = x(2);        %Rotor Diameter (ft)
    NR = x(3);       %Number of rotors (Discrete variable)
    omega = x(4);    %Angular Velocity (hz)
    C_batt = x(5);   %Battery Capacity (Wh)
    payload_weight = 14.4; %Assumes 3 pizzas and a 2 liter soda
    Wbattery = 0.013 * C_batt;  % battery weight per Wh
    Wempty = 8;  %Empty Weight of Drone
    A = 2.0;    %Cross Sectional Area of Drone
    L = 2.5;    %Length of Drone
    W = 2.5;    %Width of Drone
    grav = 32.174; %Gravity
    h = 200;    %Operational Height, needs to clear trees
    D = 10 * 5280; %Farthest possible range
    rho = 0.00223582; %Blacksburg air density

    Wprop = calc_prop_weight(d, NR); %Computes the weight of the prop
```

```
    Cd = get_cd(d, omega); %Compute the coefficient of drag
    Ct = get_ct(d, omega); %Computes the coefficient of thrust
    [T_total, T_vertical, T_forward] = get_thrust(NR, Ct, rho, omega, d, alpha); %Gets thrust components
    [motor_count, motor_weight] = motor_count_weight(NR, omega, T_total, d); %Determines the motors needed
    W_total = get_total_weight(Wempty, payload_weight, Wbattery, Wprop + motor_weight); %Computes drones weight

    spacing = get_spacing(NR, d, W, L); %Determines spacing between rotors
    T_vertical = account_for_coupling(T_vertical, spacing); %Computes vertical thrust
    T_forward = account_for_coupling(T_forward, spacing); %Computes forward thrust
    eta = calculate_efficiency(Ct, Cd); %Computes efficiency from momentum theory
    [V_forward, V_up] = compute_velo(T_forward, Cd, rho, A, grav, h, T_vertical, W_total); %Compute velocity components
    [P_hover, P_forward] = compute_Power(T_vertical, rho, A, eta, T_forward, V_forward); %Compute Power
    pow_req = compute_range_power(21 * 5280, V_forward, P_hover, P_forward); %Determine power required to complte mission
    ts = get_tip_speed(d, omega); %Get Tip speed


    alpha_climb = 0; %Alpha is zero for climb and descent
    g = 32.174;
    [T_total_0] = get_thrust(NR, Ct, rho, omega, d, alpha_climb); %Thurst is computed for vertical segment
    T_vertical_0 = account_for_coupling(T_total_0, spacing); %Vertical is accounted for coupling
    T_forward_0 = 0;
    [~, Vu_0] = compute_velo(T_forward_0, Cd, rho, A, g, h, T_vertical_0, W_total); %Computes velocity for climb

    max_speed = 146.667;
    g = zeros(6,1);
    g(1) = (pow_req - C_batt)/C_batt;         %Ensures there is enough power to complete entire mission
    g(2) = (W_total - 55)/55;                 %FAA weight limit
    g(3) = (V_forward - max_speed)/max_speed;     %FAA speed limit constraint(Forward)
    g(4) = (Vu_0 - max_speed)/max_speed;          %FAA speed limit constraint(Vertical)
    g(5) = ((ts / 1108.44) - 0.75)/0.75;      %Avoid Transonic
    g(6) = (W_total - T_vertical)/T_vertical; %Ensure that the aircraft will hover during dash

    ceq = [];
end
```

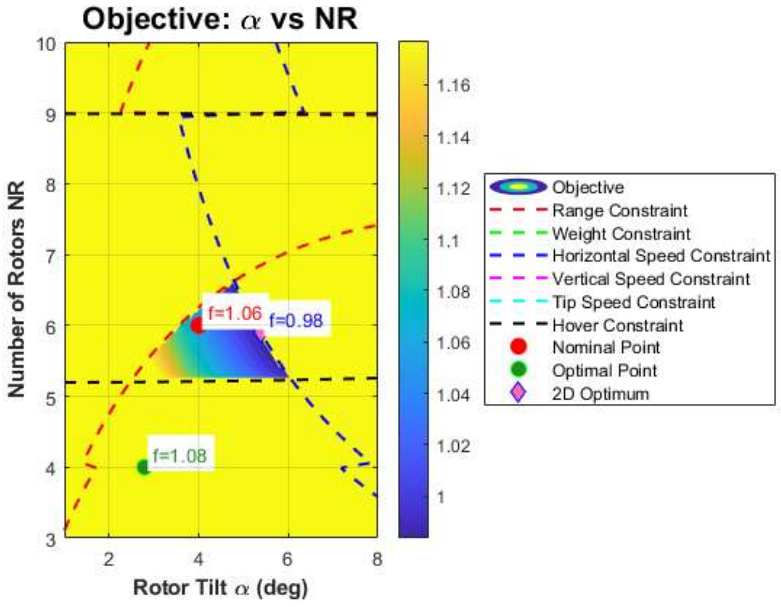



```
Warning: Contour not rendered for constant ZData
```

Plot 3: alpha vs NR Bens additional plots

```
[X, Y] = meshgrid(alpha_range, NR_range);
Z = zeros(size(X));
gmat = zeros([size(X), 6]);

% Loop through grid
for i = 1:numel(X)
    x = [X(i), d_nom, Y(i), omega_nom, C_batt_nom];
    [fval, ~, ~] = objective_fun(x);
    [g, ~] = nonlincon(x);
    [row, col] = ind2sub(size(X), i);
    gmat(row, col, :) = g;
```

```
    if all(g <= 0)
        Z(i) = fval;
    else
        Z(i) = Inf;
    end
end

[min_val, idx] = min(Z(:));
alpha_2D = X(idx);
NR_2D = Y(idx);

% Plotting
figure;
contourf(X, Y, Z, 30, 'LineColor', 'none'); hold on; colorbar
finite_Z = Z(isfinite(Z));
caxis([min(finite_Z), prctile(finite_Z, 95)]);
xlabel('Rotor Tilt \alpha (deg)', 'FontWeight', 'bold');
ylabel('Number of Rotors NR', 'FontWeight', 'bold');
title('Objective: \alpha vs NR', 'FontSize', 16, 'FontWeight', 'bold');

for k = 1:6
    contour(X, Y, gmat(:,:,k), [0 0], '--', 'LineWidth', 1.5, 'LineColor', colors{k});
end

plot(alpha_nom, NR_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
plot(alpha_opt, NR_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);
plot(alpha_2D, NR_2D, 'bd', 'MarkerFaceColor', [1 0.4 0.7], 'MarkerSize', 8);

[f_nom, ~, ~] = objective_fun([alpha_nom, d_nom, NR_nom, omega_nom, C_batt_nom]);
text(alpha_nom + 0.2, NR_nom + 0.2, sprintf('f=%.2f', f_nom), ...
    'Color', 'r', 'FontSize', 10, 'BackgroundColor', 'w');

[f_opt, ~, ~] = objective_fun([alpha_opt, d_opt, NR_opt, omega_opt, C_batt_opt]);
text(alpha_opt + 0.2, NR_opt + 0.2, sprintf('f=%.2f', f_opt), ...
    'Color', [0.13, 0.55, 0.13], 'FontSize', 10, 'BackgroundColor', 'w');

[f_2D, ~, ~] = objective_fun([alpha_2D, d_nom, NR_2D, omega_nom, C_batt_nom]);
text(alpha_2D + 0.2, NR_2D + 0.2, sprintf('f=%.2f', f_2D), ...
    'Color', 'b', 'FontSize', 10, 'BackgroundColor', 'w');

legend('Objective', ...
       'Range Constraint','Weight Constraint','Horizontal Speed Constraint','Vertical Speed Constraint','Tip Speed Constraint','Hover Constraint', ...
       'Nominal Point', 'Optimal Point', '2D Optimum', 'Location', 'eastoutside');
grid on;
```



Plot 4: d vs omega

```
[X, Y] = meshgrid(d_range, omega_range);
Z = zeros(size(X));
gmat = zeros([size(X), 6]);

% Loop through grid
for i = 1:numel(X)
```

```
    x = [alpha_nom, X(i), NR_nom, Y(i), C_batt_nom];
    [fval, ~, ~] = objective_fun(x);
    [g, ~] = nonlincon(x);
    [row, col] = ind2sub(size(X), i);
    gmat(row, col, :) = g;

    if all(g <= 0)
        Z(i) = fval;
    else
        Z(i) = Inf;
    end
end

[min_val, idx] = min(Z(:));
d_2D = X(idx);
omega_2D = Y(idx);

% Plotting
figure;
contourf(X, Y, Z, 30, 'LineColor', 'none'); hold on; colorbar
finite_Z = Z(isfinite(Z));
caxis([min(finite_Z), prctile(finite_Z, 95)]);
xlabel('Propeller Diameter d (ft)', 'FontWeight', 'bold');
ylabel('Angular Velocity \omega (Hz)', 'FontWeight', 'bold');
title('Objective: d vs \omega', 'FontSize', 16, 'FontWeight', 'bold');

% ✅ Replace this loop:
for k = 1:6
    try
        contour(X, Y, gmat(:,:,k), [0 0], '--', 'LineWidth', 1.5, 'LineColor', colors{k});
    catch
        % skip invalid constraint plots
    end
end

plot(d_nom, omega_nom, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8);
plot(d_opt, omega_opt, 'go', 'MarkerFaceColor', [0.13, 0.55, 0.13], 'MarkerSize', 8);
plot(d_2D, omega_2D, 'bd', 'MarkerFaceColor', [1 0.4 0.7], 'MarkerSize', 8);

[f_nom, ~, ~] = objective_fun([alpha_nom, d_nom, NR_nom, omega_nom, C_batt_nom]);
text(d_nom + 0.02, omega_nom + 10, sprintf('f=%.2f', f_nom), ...
    'Color', 'r', 'FontSize', 10, 'BackgroundColor', 'w');

[f_opt, ~, ~] = objective_fun([alpha_opt, d_opt, NR_opt, omega_opt, C_batt_opt]);
text(d_opt + 0.02, omega_opt + 10, sprintf('f=%.2f', f_opt), ...
    'Color', [0.13, 0.55, 0.13], 'FontSize', 10, 'BackgroundColor', 'w');

[f_2D, ~, ~] = objective_fun([alpha_nom, d_2D, NR_nom, omega_2D, C_batt_nom]);
text(d_2D + 0.02, omega_2D + 10, sprintf('f=%.2f', f_2D), ...
    'Color', 'b', 'FontSize', 10, 'BackgroundColor', 'w');

legend('Objective', ...
       'Range Constraint','Weight Constraint','Horizontal Speed Constraint','Vertical Speed Constraint','Tip Speed Constraint','Hover Constraint', ...
       'Nominal Point', 'Optimal Point', '2D Optimum', 'Location', 'eastoutside');
grid on;
```

Supporting Functions (Tyler & Ben)

```
function Pow = compute_range_power(R, V_forward, P_hover, P_forward)
    %Computes power needed to complete an entire mission
    Pow = ((P_hover + P_forward) * R) / (3600 * V_forward);
end

function cd = get_cd(d, omega)
    %Estimates the drag ceofficient by interpolating through openProp results

    d_vals = [0.2, 0.35, 0.5, 0.65, 0.8];
    omega_vals = [20, 40, 60, 80, 100];
    [D, W] = meshgrid(d_vals, omega_vals);
    cd_vals = [0.008, 0.012, 0.017, 0.022, 0.027;
               0.012, 0.017, 0.023, 0.028, 0.034;
               0.017, 0.023, 0.030, 0.035, 0.042;
               0.021, 0.028, 0.035, 0.042, 0.048;
               0.025, 0.032, 0.040, 0.048, 0.055];
    F = fit([D(:), W(:)], cd_vals(:), 'poly22');
    cd = F(d, omega);
end

function ct = get_ct(d, omega)
```

```
    %Estimates the thrust ceofficient by interpolating through openProp results
    d_vals = [0.2, 0.35, 0.5, 0.65, 0.8];
    omega_vals = [20, 40, 60, 80, 100];
    [D, W] = meshgrid(d_vals, omega_vals);
    ct_vals = [0.020, 0.030, 0.045, 0.060, 0.070;
               0.030, 0.045, 0.065, 0.080, 0.095;
               0.045, 0.065, 0.085, 0.105, 0.120;
               0.060, 0.080, 0.105, 0.125, 0.140;
               0.070, 0.095, 0.120, 0.140, 0.160]; %openprop vals
    F = fit([D(:), W(:)], ct_vals(:), 'poly22'); %Interpolates values
    ct = F(d, omega);
end

function Weight = get_total_weight(Wempty, Wpayload, Wbattery, Wprop)
    %Calculates total weight of the drone
    Weight = Wempty + Wpayload + Wbattery + Wprop;
end

function [T_total, vert_thrust, for_thrust] = get_thrust(Nr, Ct, rho, omega, d, alpha)
    %Computes thrust components for the entire drone
    T_total = Nr * Ct * rho * omega^2 * d^4;
    vert_thrust = T_total * cosd(alpha);
    for_thrust = T_total * sind(alpha);
end

function torque_per_rotor = compute_torque(T_total, d, NR)
    %Computes torque per rotor
    torque_total = T_total * d / (2 * pi);
    torque_per_rotor = torque_total / NR;
end

function [motor_count, motor_weight] = motor_count_weight(NR, omega, T_total, d)
    %Determines if another motor is needed if it cannot provide enough
    %torque
    torque_per_rotor = compute_torque(T_total, d, NR);
    max_motor_torque = 0.8; % ft-lb per motor
    motors_per_rotor = ceil(torque_per_rotor / max_motor_torque);
    motor_count = (motors_per_rotor * NR)+NR;% Need to account for the motor to adjust the pitch

    motor_weight = motor_count * 0.4;
end

function prop_weight = calc_prop_weight(d, NR)
    %Calculates the weight of the propeller
    density_plastic = 0.04; %ln/in^3
    thickness = 0.25; %in
    chord_width = 1.5; %in
    radius_in = (d / 2) * 12;
    blade_area = chord_width * radius_in;
    blade_volume = blade_area * thickness;
    weight_per_blade = blade_volume * density_plastic;
    prop_weight = weight_per_blade * 2 * NR; %Two rotors per blade
end

function [v_forward, v_up] = compute_velo(T_horizontal, Cd, rho, A, g, h, T_vertical, W_total)
    %Computes upward and forward velocities

    v_forward = sqrt((2 * T_horizontal) / (Cd * rho * A));
    if T_vertical > W_total
        v_up = sqrt(2 * g * h * (T_vertical / W_total - 1));
    else
        v_up = 0;
    end
end

function spacing = get_spacing(Nr, d, W, L)
    %Calculates the amount of spacing between rotors
    n_rows = ceil(sqrt(Nr * W / L));
    n_cols = ceil(Nr / n_rows);
    spacing_L = L / n_cols;
    spacing_W = W / n_rows;
    spacing = min(spacing_L, spacing_W) - d;
end

function thrust_out = account_for_coupling(thrust_in, spacing)
    %Estimates the effects of coupling if spacing is too small
    if spacing <= 0
        loss = 0.5;
    elseif spacing < 0.25
        loss = 0.1;
```

```
    else
        loss = 0;
    end
    thrust_out = thrust_in * (1 - loss);
end

function [P_hover, P_forward] = compute_Power(T_vertical, rho, A, eta, T_horizontal, V_forward)
    %Computes power in in both directions
    motor_efficiency = 0.8;
    mech_hover   = (T_vertical^(3/2)) / (sqrt(2 * rho * A) * eta);
    mech_forward = (T_horizontal * V_forward) / eta;
    P_hover   = mech_hover / motor_efficiency;
    P_forward = mech_forward / motor_efficiency;
end

function eta = calculate_efficiency(ct, cd)
    %Calculates efficiency from momentum theory
    eta = (ct^(3/2)) / cd;
end

function ts = get_tip_speed(d, omega)
    %Calculates tip speed
    ts = pi * d * omega;
end
```

Published with MATLAB® R2024a