Contents


```
clear;
clc;
close all
%Tyler and Ben
%Since genetic algorithm solvers do not use gradient based methods, finding
%the lagrange multipliers cannot be done. Therefore, we will use fmincon
%while starting from the point that we got from our evolutionary algorithms

x0 = [2.79; 0.8526; 4; 220.765; 611.3964];  % [alpha, d, NR, omega, C_batt]

lb = [1; 0.2; 3; 20; 100];      % Lower bounds
ub = [89; 3.0; 10; 400; 1000];  % Upper bounds

%fmincon Options
options = optimoptions('fmincon', ...
    'Display', 'iter', ...
    'Algorithm', 'interior-point', ...
    'SpecifyConstraintGradient', false, ...
    'SpecifyObjectiveGradient', false);


intcon = 3;  % NR is integer

%
[x_opt, fval, exitflag, output, lambda] = fmincon(@objective_fun, x0, [], [], [], [], lb, ub, @nonlincon, options);


x_opt(intcon) = round(x_opt(intcon));

% === Display Results ===
fprintf('\nOptimal Design Variables:\n');
fprintf('Tilt Angle (deg): %.2f\n', x_opt(1));
fprintf('Prop Diameter (ft): %.2f\n', x_opt(2));
fprintf('Number of Rotors: %d\n', x_opt(3));
fprintf('Omega (Hz): %.2f\n', x_opt(4));
fprintf('Battery Capacity (Wh): %.2f\n', x_opt(5));

% === Display Lagrange Multipliers ===
fprintf('\nLagrange Multipliers:\n');
disp(lambda.ineqnonlin)
```

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 = .3; %Weighting
w_time = .7; %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
[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

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
```

Supporting Functions (Tyler)

```
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
```

```
                                            First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       7    1.128979e+00    9.302e-01    1.308e-01
Objective function returned Inf; trying a new point...
    1      14    1.256304e+00    2.995e-01    8.002e-01    1.623e-01
    2      20    1.085326e+00    0.000e+00    5.103e-01    1.850e+00
    3      26    1.027560e+00    0.000e+00    1.016e-01    8.222e-01
    4      32    1.011855e+00    0.000e+00    1.000e-01    2.191e-01
    5      38    1.004590e+00    0.000e+00    2.826e-02    9.761e-02
    6      44    9.915503e-01    0.000e+00    2.698e-02    1.568e-01
    7      50    9.868780e-01    0.000e+00    7.638e-02    6.711e-02
    8      56    9.859193e-01    0.000e+00    5.739e-03    9.625e-02
    9      62    9.855524e-01    0.000e+00    2.866e-02    3.714e-02
   10      68    9.854709e-01    0.000e+00    2.950e-02    2.241e-02
   11      74    9.851361e-01    0.000e+00    4.030e-02    1.190e-01
   12      80    9.490297e-01    9.075e-02    3.257e-01    5.941e-01
   13      87    9.574096e-01    6.015e-02    2.140e-02    4.099e-01
   14      93    9.683691e-01    1.538e-02    7.306e-02    4.175e-01
   15      99    9.711841e-01    2.291e-02    8.269e-02    8.975e-02
   16     105    9.791667e-01    0.000e+00    1.531e-03    5.849e-02
   17     111    9.791755e-01    0.000e+00    2.329e-04    1.220e-02
```

```
   18     117    9.788077e-01    0.000e+00    1.053e-02    3.076e-02
   19     123    9.787016e-01    0.000e+00    1.008e-03    1.801e-02
   20     129    9.787117e-01    0.000e+00    1.050e-04    2.821e-03
   21     135    9.787115e-01    0.000e+00    8.400e-05    9.865e-05
   22     141    9.787115e-01    0.000e+00    8.400e-05    4.787e-04
   23     147    9.787113e-01    0.000e+00    8.401e-05    2.393e-03
   24     153    9.787103e-01    0.000e+00    8.401e-05    1.197e-02
   25     159    9.787052e-01    0.000e+00    8.401e-05    5.984e-02
   26     165    9.786809e-01    0.000e+00    8.402e-05    2.992e-01
   27     171    9.785820e-01    0.000e+00    1.206e-04    1.496e+00
   28     177    9.786641e-01    0.000e+00    4.596e-03    7.485e+00
   29     186    9.784754e-01    0.000e+00    5.880e-03    4.545e+00
Objective function returned Inf; trying a new point...
   30     196    9.778827e-01    0.000e+00    7.459e-04    3.573e+00
Objective function returned Inf; trying a new point...

                                            First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
   31     204    9.777947e-01    0.000e+00    2.526e-03    4.546e+00
Objective function returned Inf; trying a new point...
   32     214    9.772767e-01    0.000e+00    1.219e-03    8.864e+00
Objective function returned Inf; trying a new point...
   33     223    9.772133e-01    0.000e+00    4.435e-03    4.546e+00
Objective function returned Inf; trying a new point...
   34     233    9.760736e-01    0.000e+00    1.145e-03    6.959e+00
Objective function returned Inf; trying a new point...
   35     242    9.759897e-01    0.000e+00    2.205e-03    3.541e+00
Objective function returned Inf; trying a new point...
   36     252    9.747726e-01    0.000e+00    7.229e-04    5.864e+00
Objective function returned Inf; trying a new point...
   37     264    9.747203e-01    0.000e+00    1.356e-03    7.984e+00
Objective function returned Inf; trying a new point...
   38     272    9.747092e-01    0.000e+00    6.972e-03    4.031e+00
Objective function returned Inf; trying a new point...
   39     285    9.733084e-01    0.000e+00    3.110e-04    5.229e-02
Objective function returned Inf; trying a new point...
   40     298    9.731603e-01    0.000e+00    1.975e-04    1.297e-01
Objective function returned Inf; trying a new point...
   41     312    9.731202e-01    0.000e+00    1.731e-04    7.582e-02
Objective function returned Inf; trying a new point...
   42     326    9.731043e-01    0.000e+00    1.642e-04    3.728e-02
Objective function returned Inf; trying a new point...
   43     341    9.731009e-01    0.000e+00    1.624e-04    8.535e-03
Objective function returned Inf; trying a new point...
   44     356    9.731002e-01    0.000e+00    1.620e-04    1.885e-03
Objective function returned Inf; trying a new point...
   45     372    9.731001e-01    0.000e+00    1.620e-04    2.066e-04
Objective function returned Inf; trying a new point...
   46     387    9.731001e-01    0.000e+00    1.620e-04    4.520e-05
Objective function returned Inf; trying a new point...
   47     402    9.731001e-01    0.000e+00    1.165e-03    9.887e-06

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.


Optimal Design Variables:
Tilt Angle (deg): 5.67
Prop Diameter (ft): 1.00
Number of Rotors: 3
```

```
Omega (Hz): 157.55
Battery Capacity (Wh): 622.40

Lagrange Multipliers:
    0.0728
    0.0001
    0.7271
    0.0001
    0.0001
    0.0001
```
