```
function optimize_alpha_beta_advanced_separated
    % Number of gimbal axes (e.g., 4 CMGs)
    num_axes = 4;

    % Random initialization for alpha and beta within valid ranges
    rng('shuffle'); % Ensure different results on each run
    alpha0 = rand(1, num_axes) * 360 - 180; % Random values in [-180, 180]
    beta0 = rand(1, num_axes) * 180;        % Random values in [0, 180]
    x0 = [alpha0, beta0];                   % Combine into one vector

    % Bounds for alpha and beta
    lb = [-180 * ones(1, num_axes), zeros(1, num_axes)];
    ub = [180 * ones(1, num_axes), 180 * ones(1, num_axes)];

    % Optimization options
    options = optimoptions('fmincon', 'Display', 'iter', 'Algorithm', 'sqp');

    % Solve the optimization problem with added constraints
    [x_opt, ~] = fmincon(@cost_function_separated, x0, [], [], [], [], lb,
ub, @nonlinear_constraints, options);

    % Extract optimized alpha and beta
    alpha_opt = x_opt(1:num_axes);
    beta_opt = x_opt(num_axes+1:end);

    % Display results
    fprintf('Optimized Alpha (degrees):\n');
    disp(alpha_opt);
    fprintf('Optimized Beta (degrees):\n');
    disp(beta_opt);
end

% Cost function to be maximized (negative because fmincon minimizes)
function J = cost_function_separated(x)
    % Extract alpha and beta from input vector x (convert to radians)
    alpha = x(1:4) * pi / 180;
    beta = x(5:8) * pi / 180;

    % Preallocate vectors
    g_g = zeros(3, 4);
    g_s = zeros(3, 4);
    g_t = zeros(3, 4);

    % Calculate gimbal axis (g_g), spin direction (g_s), and torque axis
(g_t)
    for i = 1:4
        % Gimbal axis vector
        g_g(:, i) = [cos(alpha(i)) * sin(beta(i)); sin(alpha(i)) *
sin(beta(i)); cos(beta(i))];
```

```
        % Spin direction
        g_s(:, i) = [-sin(alpha(i)); cos(alpha(i)); 0];

        % Torque vector (cross product of g_g and g_s)
        g_t(:, i) = cross(g_g(:, i), g_s(:, i));
    end

    % Define the gimbal angle delta (assuming delta = 0 for simplicity)
    delta = zeros(4, 1); % Adjust delta as needed

    % Construct J_4, J_3, and J_2 matrices
    J_4 = [g_s, g_t];
    J_3 = [g_s(:, 1:3), g_t(:, 1:3)];
    J_2 = [g_s(:, 1:2), g_t(:, 1:2)];

    % Compute determinants
    det_J4_J4T = det(J_4 * J_4');
    det_J3_J3T = det(J_3 * J_3');
    det_J2_J2T = det(J_2 * J_2');

    % Display each determinant value
    fprintf('Determinant of J_4 J_4^T: %.4f\n', det_J4_J4T);
    fprintf('Determinant of J_3 J_3^T: %.4f\n', det_J3_J3T);
    fprintf('Determinant of J_2 J_2^T: %.4f\n', det_J2_J2T);

    % Weights
    w_4 = 2;
    w_3 = 2;
    w_2 = 1;

    % Total cost function
    J = -(w_4 * det_J4_J4T + w_3 * det_J3_J3T + w_2 * det_J2_J2T);

    % Display the total cost function value
    fprintf('Total Cost Function Value: %.4f\n', -J);
end

% Non-linear constraints to enforce pairwise conditions
function [c, ceq] = nonlinear_constraints(x)
    % Extract alpha and beta from input vector x (convert to radians)
    alpha = x(1:4) * pi / 180;
    beta = x(5:8) * pi / 180;

    % Calculate gimbal axis (g_g)
    g_g = zeros(3, 4);
    for i = 1:4
        g_g(:, i) = [cos(alpha(i)) * sin(beta(i)); sin(alpha(i)) *
sin(beta(i)); cos(beta(i))];
    end
    % Compute max(g_i^T * g_j) for all i < j
    max_inner_products = [];
    for i = 1:3
        for j = i+1:4
            max_inner_products = [max_inner_products; g_g(:, i)' * g_g(:,
j)];
        end
    end

    % Constraint to ensure max_inner_products is minimized
    c = max(max_inner_products) - 0.9; % Example threshold (adjust as needed)
    ceq = [];
end
```