```
function gimbal_angles = optimize_cmg_zero_momentum_balanced()
    % Parameters
    N = 5; % Number of CMGs
    charges = zeros(N, 3); % Store charge positions on the unit sphere
    angles = zeros(N, 2); % Store azimuth (alpha) and elevation (beta) angles
    damping_coeff = 15.0; % Virtual damping coefficient
    max_iters = 1000; % Maximum iterations
    tolerance = 1e-6; % Convergence threshold

    % Initial conditions for the first three gimbal axes
    charges(1, :) = [1, 0, 0]; % Gimbal axis 1 along x-axis
    charges(2, :) = [-0.5, sqrt(3)/2, 0]; % Gimbal axis 2 at 120 degrees
    charges(3, :) = [-0.5, -sqrt(3)/2, 0]; % Gimbal axis 3 at 240 degrees

    % Check if the first three gimbal axes satisfy zero-momentum condition
    assert(norm(sum(charges(1:3, :), 1)) < tolerance, 'First three axes do
not sum to zero.');

    % Randomly initialize gimbal axes 4 and 5
    for i = 4:N
        charges(i, :) = normalize(rand(1, 3) - 0.5);
    end

    % Main optimization loop for 4th and 5th axes
    for iter = 1:max_iters
        forces = zeros(N, 3); % Initialize forces

        % Calculate Coulomb forces only for axes 4 and 5
        for i = 4:N
            for j = 4:N
                if i ~= j
                    r = charges(i, :) - charges(j, :);
                    force = r / (norm(r)^3); % Coulomb force
                    forces(i, :) = forces(i, :) + force;
                end
            end
        end

        % Project forces onto the sphere and apply damping
        for i = 4:N
            forces(i, :) = forces(i, :) - dot(forces(i, :), charges(i, :)) *
charges(i, :);
            charges(i, :) = charges(i, :) + forces(i, :) / damping_coeff;
            charges(i, :) = normalize(charges(i, :)); % Constrain to unit
sphere
        end

        % Check if the momentum vectors 4 and 5 sum to zero
        h45 = charges(4, :) + charges(5, :);
        if norm(h45) < tolerance
```

```
            break;
        end

        % Adjust positions iteratively to enforce the zero-momentum condition
        charges(4, :) = charges(4, :) - 0.5 * h45; % Reduce imbalance
        charges(5, :) = charges(5, :) - 0.5 * h45;
        charges(4, :) = normalize(charges(4, :)); % Normalize after
adjustment
        charges(5, :) = normalize(charges(5, :));
    end

    % Convert cartesian coordinates to spherical angles
    for i = 1:N
        [alpha, beta] = cartesian_to_spherical(charges(i, :));
        angles(i, :) = [alpha, beta];
    end

    % Output results
    gimbal_angles = angles;
    disp('Optimal Gimbal Angles (Azimuth, Elevation) in Degrees:');
    disp(array2table(angles, 'VariableNames', {'Azimuth', 'Elevation'}));
end

function cart = spherical_to_cartesian(alpha, beta)
    % Convert spherical (degrees) to cartesian coordinates
    alpha = deg2rad(alpha);
    beta = deg2rad(beta);
    cart = [cos(beta) * cos(alpha), cos(beta) * sin(alpha), sin(beta)];
end

function [alpha, beta] = cartesian_to_spherical(cart)
    % Convert cartesian to spherical (degrees)
    beta = asind(cart(3)); % Elevation
    alpha = atan2d(cart(2), cart(1)); % Azimuth
end

function vec = normalize(vec)
    % Normalize a vector
    vec = vec / norm(vec);
end
```