```
function [Ui] = inputcontrol_Algorithm5(num_nodes, nodes, Nei_agent, n, epsilon
    , r, r_prime, d, k_scale, Nei_beta_agent, p_ik, q_ik, obstacles, qt1, pt1,
    p_nodes)
     %{
    This function is to find alpha and beta neighbors
    Created by Anthony Bugatto

    Inputs: positions of nodes (nodes),
            indices of alpha neighbors (Nei_agent)
            (n)
            (epsilon)
            active range for alpha agents (r)
            (d)
            (k_scale)
            (p_nodes)

    Outputs: controlled acceleration (Ui)

    %}
    %+++++++++++++++++Constants++++++++++++++++++++

    c_a1 = 30;
    c_a2 = 2*sqrt(c_a1);
    c_b1 = 1500;
    c_b2 = 2*sqrt(c_b1);
    c_mt1 = 1.1;
    c_mt2 = 2*sqrt(c_mt1);
    a = 5;
    b = 5;
    c = abs(a - b) / sqrt(4*a*b);
    r_sig = sigma_norm(r);
    d_sig = sigma_norm(d);

    r_prime_sig = sigma_norm(r_prime);
    k_prime = r_prime / r;
    d_prime_sig = sigma_norm(k_prime / d);
    %+++++++++++++++++BUILDING THE ADJACENCY MATRCEES++++++++++++++++++++

    n_ij = zeros(num_nodes,num_nodes,n); %gradient matrix 1x2
    n_ik = zeros(size(obstacles,1),size(obstacles,1),n); %gradient matrix 1x2
    for i = 1:num_nodes
        for j = 1:num_nodes
            q = norm(nodes(j,:) - nodes(i,:));
            sig_grad = (nodes(j,:) - nodes(i,:)) / (1 + epsilon * sigma_norm
                (nodes(j,:) - nodes(i,:)));

            if q < r && q ~= 0 %is zero otherwise
                n_ij(i,j,:) = sig_grad;
            end
        end

        for k = 1:size(obstacles,1)
            q = norm(q_ik{i}(k,:) - nodes(i,:));
             sig_grad_k = (q_ik{i}(k,:) - nodes(i,:)) / (1 + epsilon *
                 sigma_norm(q_ik{i}(k,:) - nodes(i,:)));
```

```
        if q < r && q ~= 0 %is zero otherwise
            n_ik(i,k,:) = sig_grad_k;
        end
    end
end

%+++++++++++++++BUILDING CONTROL ACCELERATION Ui++++++++++++++++++++++

U = zeros(num_nodes, n); %100x3 matrix for accelerations
Ug = zeros(num_nodes,n); % gamma agent control

gradient = 0;
conscensus = 0;
a_ij = zeros(num_nodes,num_nodes); %spatial adjacency matrix

gradient_k = 0;
conscensus_k = 0;
b_ij = zeros(num_nodes,length(Nei_agent)); %spatial adjacency matrix
for i = 1:num_nodes %loop through all i in Ui matrix
    for j = 1:size(Nei_agent{i}) % loop through all neighbors in neighbor
        matrix for each i
        Nei_val = Nei_agent{i}(j);
        if i ~= Nei_val
            %phi is the time differential of the smooth pairwise
            %   attractive/repulsive  potential
            z = sigma_norm(nodes(Nei_val,:) - nodes(i,:)); %parameter for
                phi_alpha
            z_phi = z - d_sig; %parameter for phi
            phi_bump = bump(z / r_sig);
            sigmoid = (z_phi + c) / sqrt(1 + (z_phi + c)^2);
            phi = .5 * ((a + b) * sigmoid + (a - b));

            phi_alpha = phi_bump * phi;
            %implement the algorithm for the fragmenting control law:
            %
            % Ui = c_a1*SUM[phi_alpha * nij) + c_a2*SUM[aij * (pj -
            %       pi)] + Ug
            %
            gradient = phi_alpha * [n_ij(i,Nei_val,1) n_ij(i,Nei_val,2)];
            conscensus = a_ij(i,Nei_val) * (p_nodes(Nei_val,:) - p_nodes(i
                ,:));
        end
    end

    for k = 1:size(Nei_beta_agent{i}) % loop through all neighbors in
        neighbor matrix for each i
        Nei_val_k = Nei_beta_agent{i};
        if i ~= Nei_val_k
            %phi is the time differential of the smooth pairwise
            %   attractive/repulsive  potential
            z_k = sigma_norm(q_ik{i}(Nei_Val_k,:) - nodes(i,:)); %parameter
                for phi_alpha
            z_phi_k = z_k - d_prime_sig; %parameter for phi
            phi_bump_k = bump(z_k / r_prime_sig);
            sigmoid_k = z_phi_k / sqrt(1 + z_phi_k^2);
            phi_k = sigmoid_k - 1;
```

```
                phi_alpha_k = phi_bump_k * phi_k;

                b_ik = phi_bump_k;
                %implement the algorithm for the fragmenting control law:
                %
                % Ui = c_a1*SUM[phi_alpha * nij) + c_a2*SUM[aij * (pj -
                %       pi)] + Ug
                %
                gradient_k = phi_alpha_k * [n_ik(i,Nei_val_k,1) n_ij(i,
                    Nei_val_k,2)];
                conscensus_k = b_ik(i,Nei_val_k) * (p_ik(Nei_val_k,:) - p_ik(i
                    ,:));
            end
        end

        fg = -c_mt1 * (nodes(i,:) - qt1) + -c_mt2 * (p_nodes(i,:) - pt1);
        fa = (c_a1 * gradient) + (c_a2 * conscensus);
        fb = (c_b1 * gradient_k) + (c_b2 * conscensus_k);

        U(i,:) = fa + fb + fg;
    end

    Ui = U;
end
```