```
%=========================================================
%Anthony Bugatto
%CS 455: Mobile Sensor Netorks
%Project 1: Flocking
%=========================================================

                    clc,clear
                    close all

%================ALGORITHM NUMBER=======================

                    ALG_NUM = 5;

%=============PARAMETER OF SIMULATION===================

d = 15;%Set desired distance among sensor nodes
k_scale = 1.2;%Set the scale of MSN
r = k_scale*d; %Set the active range
r_prime = .22*k_scale*r; %Set the active range of beta agent
epsilon = 0.1; %Set a constant for sigma norm
num_nodes = 100; %Set number of sensor nodes
n=2; %Set number of dimensions

grid_size = 0;
if ALG_NUM ~= 1 && ALG_NUM ~= 2
    grid_size = 150;
else
    grid_size = 50;
end

nodes = grid_size.*rand(num_nodes, n) + grid_size.*repmat([0 1], num_nodes, 1);
    %Randomly generate initial positions of MSN
p_nodes = zeros(num_nodes,n); %Set initial velocties of MSN
delta_t_update = 0.08; %Set time step
t = 0:delta_t_update:7;% Set simulation time

if ALG_NUM == 5 %=================SET OBSTACLES==========================
    obstacles = [50, 100; 150 80; 200, 230; 280 150]; %set positions of
        obstacles
    Rk = [20; 10; 15; 8]; %Radii of obstacles
    num_obstacles = size(obstacles,1); %Find number of obstacles
end

if ALG_NUM ~= 1 %=========SET A STATIC TARGET===============
    qt1 = [150 150]; %Set position of the static target (gamma agent)
    pt1= [0 0]; %Set initial velocity of the target
end

nodes_old = nodes; %KEEP previous positions of MSN
q_mean = zeros(size(t,2), n); %Save positions of COM (Center of Mass)
p_mean = zeros(size(t,2), n); %Save velocities of COM (Center of Mass)
Connectivity = []; %save connectivity of MSN
q_nodes_all = cell(size(t,2), num_nodes); %creates cell array to store history
    of system pos
p_nodes_all = cell(size(t,2), num_nodes); % -   -   -   -   -   -   -   -   -
    -   -   -vel
```

```
nFrames = 20; %set number of frames for the movie
mov(1:nFrames) = struct('cdata', [],'colormap', []); %Preallocate movie
    structure

for iteration = 1:length(t)
    if ALG_NUM ~= 1
        if ALG_NUM == 3
           %Sinewave Trajectory of a moving target
           qt_x1 = 50 + 50*t(iteration);
           qt_y1 = 295 - 50*sin(t(iteration));
        elseif ALG_NUM == 4
            %Circle Trajectory of a moving target
            qt_x1 = 310 - 160*cos(t(iteration));
            qt_y1 = 255 + 160*sin(t(iteration));
        elseif ALG_NUM == 5
            %Line Trajectory of a moving target
            qt_x1 = 200 + 130*t(iteration);
            qt_y1 = 200 + 1*t(iteration);
        end

        if ALG_NUM == 3 || ALG_NUM == 4 || ALG_NUM == 5
            %compute position of target
            qt1(iteration,:) = [qt_x1, qt_y1];

            %compute velocities of target
            if iteration > 1
                pt1(iteration,:) = (qt1(iteration,:) - qt1(iteration-1,:)) /
                    delta_t_update;
            else
                continue
            end
        end

        plot(qt1(:,1),qt1(:,2),'ro','LineWidth',2,'MarkerEdgeColor','r',
            'MarkerFaceColor','r', 'MarkerSize',4.2)
        hold on
    end

    %=====================(LOOP) UPDATE PROCESS for MSN=====================

    if ALG_NUM == 1 %fragmentation
        [Nei_agent, A] = findneighbors1(nodes, r);
        [Ui] = inputcontrol_Algorithm1(num_nodes, nodes, Nei_agent, n, epsilon,
            r, d, p_nodes);
    elseif ALG_NUM == 2 % static target
        [Nei_agent, A] = findneighbors1(nodes, r);
        [Ui] = inputcontrol_Algorithm2(ALG_NUM, num_nodes, nodes, Nei_agent, n,
            epsilon, r, d, qt1, pt1, p_nodes);
    elseif ALG_NUM == 3 || ALG_NUM == 4 %goal following
        [Nei_agent, A] = findneighbors1(nodes, r);
        [Ui] = inputcontrol_Algorithm2(ALG_NUM, num_nodes, nodes, Nei_agent, n,
            epsilon, r, d, qt1(iteration,:), pt1(iteration,:), p_nodes);
    elseif ALG_NUM == 5 %goal following and obstacle avoiding
        [Nei_agent, Nei_beta_agent, p_ik, q_ik, A] = findneighbors5(nodes_old,
```

```
            nodes, r, r_prime, obstacles, Rk, n, delta_t_update);
        [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);
    end

    p_nodes = (nodes - nodes_old)/delta_t_update; %COMPUTE velocities of sensor
        nodes
    p_nodes_all{iteration} = p_nodes; %SAVE VELOCITY OF ALL NODES
    nodes_old = nodes;
    nodes = nodes_old + p_nodes*delta_t_update + .5*Ui*delta_t_update*
        delta_t_update;
    q_mean(iteration,:) = mean(nodes); %Compute position of COM of MSN

    if ALG_NUM ~= 1
        plot(q_mean(:,1),q_mean(:,2),'ro','LineWidth',2,'MarkerEdgeColor','k',
            'MarkerFaceColor','k','MarkerSize',4.2)
        hold on
    end

    p_mean(iteration,:) = mean(p_nodes); %Compute velocity of COM of MSN
    q_nodes_all{iteration} = nodes;
    Connectivity(iteration)= (1 / num_nodes) * rank(A);

    if ALG_NUM == 5 %Draw obstacles
         phi = 0:.1:2*pi;
         for k = 1:num_obstacles
            X = Rk(k)*cos(phi);
            Y = Rk(k)*sin(phi);
            plot(X+obstacles(k,1),Y+obstacles(k,2),'r',nodes(:,1),nodes(:,2),
                'g>')
            fill(X+obstacles(k,1),Y+obstacles(k,2),'r')
            axis([0 250 0 80]);
            hold on
         end
    end

    %================= PLOT and LINK SENSOR TOGETHER ===============
    plot(nodes(:,1),nodes(:,2), '.')
    hold on
    plot(nodes(:,1),nodes(:,2), 'k>','LineWidth',.2,'MarkerEdgeColor','k',
        'MarkerFaceColor','k','MarkerSize',5)
    hold off

    for node_i = 1:num_nodes
        tmp=nodes(Nei_agent{node_i},:);
        for j = 1:size(nodes(Nei_agent{node_i},1))
            line([nodes(node_i,1),tmp(j,1)],[nodes(node_i,2),tmp(j,2)])
        end
    end

    mov(iteration) = getframe;
    hold off
end

%=========================VIDEO SIMULATION=============================
```

```
%{
v = VideoWriter('flocking.avi', 'MPEG-4'); %Make movie
open(v)
writeVideo(v,mov);
%}
%========================PLOT VELOCITY OF MSN===========================

p_each_nodes = [];
for i = 2:size(t,2) %iterates through the timesteps for the history cell matrix
    tmp7 = p_nodes_all{i};
    for j = 1:num_nodes
        if j == 1 %Plot velociy of sensor node 1; you can change this number to
            plot for other nodes
            p_each_nodes(i) = norm(tmp7(j,:));
            figure(3), plot(p_each_nodes, 'b')
            hold on
        end
    end
end

figure(4), plot(Connectivity)
grid on

%========================PLOT TRAJECTORY OF SENSOR NODES===============

for i = 2:length(q_nodes_all)
    tmp8 = q_nodes_all{i};
    figure(5), plot(tmp8(:,1), tmp8(:,2), 'k.')
    hold on
end

hold on
plot(nodes(:,1), nodes(:,2), 'm>', 'LineWidth', .2, 'MarkerEdgeColor', 'm',
    'MarkerFaceColor', 'm', 'MarkerSize', 5)

%========================PLOT TRAJECTORY OF COM AND TARGET===============

figure(6), plot(q_mean(:,1), q_mean(:,2),'k.')
hold on

if ALG_NUM ~= 1 || ALG_NUM ~= 2
    plot(qt1(:,1), qt1(:,2), 'r.')
end
```

```
function [Nei_agent, A] = findneighbors1(nodes, r)

%This function is to find alpha and beta neighbors
%Created by Anthony Bugatto

% Inputs:  positions of nodes (nodes),
          %active range for alpha agents(r)

% Outputs: indices of alpha neighbors (Nei_agent)
           %Adjacency Matrix (A)

%*******Find neighbors of alpha agent*************

num_nodes = size(nodes,1);
dif = cell(num_nodes,1); % save the difference between each alpha agent and all
    other nodes
                         % each element of cell is a matrix(size:num_nodes x n)
distance_alpha = zeros(num_nodes,num_nodes); % save the distance (norm) between
    each agent and all other nodes
                                % each column for one node
Nei_agent = cell(num_nodes,1); %Save the indices of neighbors of each agent
                               %each element of cell is a matrix (maximum size
                                   num_nodes x 1)

for i = 1:num_nodes
    dif{i} = repmat(nodes(i,:),num_nodes,1) - nodes;
    tmp = dif{i}; %recall cell i th of dif

    for j = 1:num_nodes
        d_tmp(j,:) = norm(tmp(j,:)); %compute distance between each alpha agent
            and all other nodes
    end

    distance_alpha(i,:)= d_tmp;
end

for k = 1:num_nodes
    Nei_agent{k} = find(distance_alpha(:,k) < r & distance_alpha(:,k) ~= 0); %
        find the neighbors of agent i
end

%+++++++++++++++++BUILDING THE ADJACENCY MATRIX++++++++++++++++++++++

A = zeros(num_nodes,num_nodes);

for i = 1:num_nodes
    for j = 1:num_nodes
        if i ~= j
            dist_2nodes = norm(nodes(j,:) - nodes(i,:));
            if dist_2nodes < r && dist_2nodes ~= 0
                A(i,j) = 1;
            end
        end
    end
end
```

```
function [Nei_agent, Nei_beta_agent, p_ik, q_ik, A] = findneighbors5(nodes_old,
    nodes, r, r_prime, obstacles, Rk, n, delta_t_update)

%This function is to find alpha and beta neighbors
%Created by Hung Manh La (Jan 2008)
%Oklahoma State University
%Copyright @2008 (any reproduction or modification of this code needs
    permission from the author)
%This code is not published in any paper yet. Hence if you have any
%question of this code please contact the author:lamanhhungosu@gmail.com

% Inputs:  positions of nodes (nodes),
          %active range for alpha agents(r)
          %active range for beta agents(r_prime)
          %positions of obstacles (obstacle)
          %radius of obstacles (Rk)
          %number of dimensions (n)
          %velocities of nodes (p_nodes)
          %time update (delta_t_update)

% Outputs: indices of alpha neighbors (Nei_agent)
           %indices of beta neighbors (Nei_beta_agent)
           %positions of virtual beta agent(q_ik)
           %Velocities of virtual beta agents (p_ik)


          %*******Find neighbors of alpha agent*************

num_nodes = size(nodes,1);
dif = cell(num_nodes,1); % save the difference between each alpha agent and all
    other nodes
                         % each element of cell is a matrix(size:num_nodes x n)
distance_alpha = zeros(num_nodes,num_nodes); % save the distance (norm) between
    each agent and all other nodes
                                % each column for one node
Nei_agent = cell(num_nodes,1); %Save the indices of neighbors of each agent
                               %each element of cell is a matrix (maximum size
                                   num_nodes x 1)

for i = 1:num_nodes
    dif{i} = repmat(nodes(i,:),num_nodes,1) - nodes;
    tmp = dif{i}; %recall cell i th of dif
    for j = 1:num_nodes
        d_tmp(j,:) = norm(tmp(j,:)); %compute distance between each alpha agent
            and all other nodes
    end
    distance_alpha(i,:)= d_tmp;
end

for k = 1:num_nodes
    Nei_agent{k} = find(distance_alpha(:,k)<r & distance_alpha(:,k)~=0); %find
        the neighbors of agent i
end
```

```
  %===========================================================
%*******Find neighbors of beta agent (q_ik)- Virtual beta agent********

num_obstacles = size(obstacles,1); %find number of obstacles

dif_qi_yk = cell(num_nodes,1); % save the difference between all centers of
    obstacles and each node
                                    % each element of cell is a matrix (l x n), l =
                                        number of obstacles
 % Compute miu and projection matrix (ak)
miu = zeros(num_obstacles,num_nodes);% Each column for each obstacle
ak = cell(num_nodes,1); %each element of cell is matrix (num_obstacles x n)
P = cell(num_nodes,1); %each element of cell is matrix (n x n), n = 2
%Compute positions of q_ik and velocities of p_ik
q_ik = cell(num_nodes,1);%each element of cell is matrix (l x n), n= 2
p_ik = cell(num_nodes,1);%each element of cell is matrix (l x n), n= 2
dif_beta =cell(num_nodes,1); % save the difference between all beta agents and
    each node
                              % each element is a matrix(size:l x n)
distance_beta = zeros(num_obstacles,num_nodes); % save the distance (norm)
    between each shadow on obstacle
                              %and all other nodes each
                              %column for one node
Nei_beta_agent = cell(num_nodes,1);

for i = 1:num_nodes
    %Find the difference between each center of obstacle and nodes
    dif_qi_yk{i} = repmat(nodes(i,:),num_obstacles,1)- obstacles;
    tmp_dif = dif_qi_yk {i}; %recall cell i th of dif_qi_ki

    ak_tmp = zeros(num_obstacles,n); %temporary save projection matrix in each
        obstacle k
        for k = 1:num_obstacles
            miu_tmp(:,k)= Rk(k,:)/norm(tmp_dif(k,:)); %compute distance between
                qi and yk
            ak_tmp(k,:) =  tmp_dif(k,:)/norm(tmp_dif(k,:));
        end
    miu(:,i)= miu_tmp;
    ak{i}= ak_tmp;
    P{i} = eye(n,n)- (ak{i,:})'*(ak{i,:}); %Compute projection matrix (nho
        check lai)

    %++++++++++++++++++++++++++++++++++++++++
    q_ik_tmp = zeros(num_obstacles, n);%temporary save q_ik in each obstacle k
    p_ik_tmp = zeros(num_obstacles, n);%temporary save q_ik in each obstacle k
    p_i = (nodes(i,:)- nodes_old(i,:))/delta_t_update;

        for k = 1:num_obstacles
            %%Compute positions of q_ik
            q_ik_tmp(k,:) = miu(k,i)*nodes(i,:)+ (1-miu(k,i))*obstacles(k,:);
            %Compute velocities of p_ik
            p_ik_tmp(k,:) = (miu(k,i)*P{i}*p_i')';
        end

    q_ik{i} = q_ik_tmp ;
    p_ik{i} = p_ik_tmp ;
```

```
    %++++++++++++++++++++++++++++++++++++++
     %Find beta neighbors of alpha agent after obtaining q_ik
    dif_beta{i}= q_ik{i}- repmat(nodes(i,:),num_obstacles,1);

    %Compute norm (distance) of dif_beta
    tmp_norm = dif_beta{i};
       for k = 1:num_obstacles
        distance_beta_tmp(k,:)= norm(tmp_norm(k,:));

       end
    distance_beta(:,i) = distance_beta_tmp;
    % Find neighbors
    Nei_beta_agent{i} = find(distance_beta(:,i)<r_prime & distance_beta(:,i)~=0
        ); %find the beta neighbors of agent i

end

%++++++++++++++++BUILDING THE ADJACENCY MATRIX++++++++++++++++++++

A = zeros(num_nodes,num_nodes);

for i = 1:num_nodes
    for j = 1:num_nodes
        if i ~= j
            dist_2nodes = norm(nodes(j,:) - nodes(i,:));
            if dist_2nodes < r && dist_2nodes ~= 0
                A(i,j) = 1;
            end
        end
    end
end
```

```
function [Ui] = inputcontrol_Algorithm1(num_nodes, nodes, Nei_agent, n, epsilon
    , r, d, 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);
    a = 5;
    b = 5;
    c = abs(a - b) / sqrt(4*a*b);
    r_sig = sigma_norm(r);
    d_sig = sigma_norm(d);

    %+++++++++++++++++BUILDING THE ADJACENCY MATRCEES+++++++++++++++++++++

    n_ij = zeros(num_nodes,num_nodes,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
    end

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

    U = zeros(num_nodes, n); %100x3 matrix for accelerations

    gradient = 0;
    conscensus = 0;
    a_ij = zeros(num_nodes,num_nodes);
    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
                rho_h = bump(z / r_sig);
                sigmoid = (z_phi + c) / sqrt(1 + (z_phi + c)^2);
                phi = .5 * ((a + b) * sigmoid + (a - b));

                phi_alpha = rho_h * phi;

                a_ij(i,Nei_val) = rho_h;
                %implement the algorithm for the fragmenting control law:
                %
                % Ui = c_a1*SUM[phi_alpha * nij) + c_a2*SUM[aij * (pj - pi)]
                %

                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

        U(i,:) = (c_a1 * gradient) + (c_a2 * conscensus);
    end

    Ui = U;
end
```

```
function [Ui] = inputcontrol_Algorithm2(ALG_NUM, num_nodes, nodes, Nei_agent, n
    , epsilon, r, d, 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_mt1 = 5.1;
    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);

    %++++++++++++++++++BUILDING THE ADJACENCY MATRCEES+++++++++++++++++++++

    n_ij = zeros(num_nodes,num_nodes,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
    end

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

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

    conscensus = 0;
    a_ij = zeros(num_nodes,num_nodes); %spatial adjacency matrix
    for i = 1:num_nodes %loop through all i in Ui matrix
        gradient = 0;
        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;

                a_ij(i,Nei_val) = phi_bump;
                %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

        p = 0;
        if ALG_NUM ~= 2
            p = -c_mt2 * (p_nodes(i,:) - pt1);
        end

        fg = -c_mt1 * (nodes(i,:) - qt1) + p;
        fa = (c_a1 * gradient) + (c_a2 * conscensus);

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


    Ui = U;
end
```

```
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
```

```
function signorm = sigma_norm(in)
    epsilon = .1; %needs to be less than 1
    signorm = (1/epsilon)*(sqrt(1 + epsilon*(norm(in,2)^2)) -1);
end

%test and comparison
%s1 = sigma_norm([3,5],[7,5]) = 6.1245
%norm = 4

%s2 = sigma_norm([2,8],[4,5]) = 5.1658
%norm = 3.6056
```

```
function out = bump(z)
    if (z >= 0) && (z < .2)
        out = 1;
    elseif (z >= .2) && (z <= 1)
        out = .5*(1 + cos(pi*((z - .2) / .8)));
    else
        out = 0;
    end
end
```