ENME808T
HW5
Controller Description and Observations

Vishnuu Appaya Dhanabalan
116873314

December-12th-2019

Task	Simulation	Robotarium
Init	Success	Success
Navigate	Success	-
Search	Success	Success
Full	Success	-

Table 1: Summary of results

Robotarium Video

1 Init Task

Task: The agents are initially deployed in the outskirts of the search area. The agents are equipped with short-range communication equipment, and their interactions can be modeled using a -disk graph. Assuming that the agents are initially connected, their first objective is to rendezvous and ensure that all agents can communicate with another while avoiding collisions.

For this, an edge tension function was chosen which keeps the agents between Delta and delta.

$$0.05\frac{(\|x_i-x_j\|-d_{ij})^2}{(\Delta-\|x_i-x_j\|)(\|x_i-x_j\|-\delta)}$$

For agents within the Delta- e region, the agents are commanded with weights corresponding to above edge tension function, whereas agents within Delta but outside the Delta - e were given weight wij = 1.
The agents were seen to achieve consensus while maintaining the safety distance.

1.1 Observation

On using the edge tension function :

$$0.05\frac{(\|x_i-x_j\|-d_{ij})^4}{(\Delta-\|x_i-x_j\|)(\|x_i-x_j\|-\delta)}$$

it was seen that the agents happened to collide.

1.2 Code snippet

```
case 'init'
        wij = 0;
        e = 0.1*Delta;
        delta_aug = delta;
        d = (Delta+delta)/2;
        for jj = 1:size(nIDs,2)
            % If within Delta - e start edge function
            z = norm(xj(jj) - xi);
            if (z < (Delta - e))
                wij = 0.05*(Delta*z + delta*z -2*delta*
                    Delta)/((z-delta)^2*(Delta-z)^2);
            else
                wij = 1;
            end
            ui = ui + wij*(xj(jj) - xi);
        end
```

2 Navigate

My code works just fine in the simulation on matlab, however one of the agents gets stuck at the last tree bark before the entrance.

Objectives of leader:

- **Traverse to target:**
  - The leader was made to traverse to target with the weight function being constant that is changed in a hybrid manner.

$$w_{ij} = \left\{ \begin{array}{cc} 0.1 & norm(xt - xi) < 1.4 \\ 0.01 & otherwise \end{array} \right\} \tag{1}$$

The control input is thererby given by

$$u_i = u_i + w_{ij} * R_1 * (x_t - x_i) \tag{2}$$

where R_1 is a constant rotation matrix that drives the agent at an angle relative to the $x_t - x_i$ vector. Here R_1 is given by

$$R_1 = \begin{bmatrix} cos(8^0) & sin(8^0) \\ -sin(8^0) & cost(8^0) \end{bmatrix} \tag{3}$$

```
1 case 'navigate'
2 leaderAgent = ~isempty(missionData.MissionInfo);
3 R1 = [0 0;0 0];
4 off_th = 8;
5 Rl = [cosd(off_th) sind(off_th); -sind(off_th)
      cosd(off_th)];
6 if leaderAgent
7       ui_t = 0;
8       ui_orth_t = 0;
9       ui_obs_t = 0;
10      xt = missionData.MissionInfo.target
11      if norm(xt-xi) < 1.4
12          wij = 1e-1;
13          xt = xt + [0.25;0.25];
14      else
15          wij = 1e-2;
16      end
17      ui_t = wij*Rl*(xt-xi);
```

- **Avoid Obstacles:**
  - Create a set of object vectors for all available finite sensorDistances.This set would contain all obstacles 'O_i' as well as other agents.

```
obs = (sensorData > delta); % for all distances
    greater than 2 delta;
obs = obs & (sensorData<Inf);

v=find(obs); % returns indices of such above
    objects
if size(v) > 0 % if any obstacle present
    O = zeros(2,size(v,2));
    obs_ang = 90;
    for i=1:size(v) % for all such obstacles
        obstacle_found = 0;
        O(:,i) = sensorData(v(i))*sensorDir(:,v(i
            ));
```

  - For each object in Obstacle vector 'O_i', check if distance between leader and obstacle is less than a user-defined obstacle distance = 1.3*delta, when satisfied, we start the energy functions.

  - If the object is within 1.3*delta, an orthogonal repelling vector is applied to the leader. The direction of the repelling force is found based on the sensorTheta.

```
function [R] = Rot(i)
    if (i == 1 || i == 8 || i == 7 )
        R = [0 -1; 1 0];
    elseif (i == 2 || i== 3)
        R = [0 1; -1 0];
    elseif (i == 4 || i == 5)
        R = [0 1;1 0];
    else
        R = [0 -1; -1 0];
    end
end

if (sensorData(v(i)) < 1.3*delta)
    wij_obs_orth = (1e-2)/((2*z^(4)));
    R = Rot(v(i));
    ui_orth = wij_obs_orth*R*O(:,i);
```

  - If the object is within 1.2*delta, a considerable repelling weight function is further added defined with a heavy weight function. A variable that keeps number of obstacles found is incremented.

```
if (sensorData(v(i)) < 1.2*delta)
    obstacle_found = obstacle_found+ 1;
    wij_obs = -(800)/((2*z^(4)))
    ui_obs = wij_obs*(O(:,i));
```

– Check if the obstacle happens to be one the agents itself by comparing the Obstacle vector to the neighbourhood vector xj. If there is a coincidence, set the uis to be = 0. In this case, we decrement the number of obstacles found as well.

```
k = 0.01;
if find( (abs(vecnorm(xj-xi)-(norm(O(:,i))+0.11))
    ) < k)
    ui_orth = 0;
    ui_obs = 0;
    obstacle_found = obstacle_found - 1;
end
```

– After the two above computation, we add the above uis to the total ui orth and ui obs for a particular agent.

```
ui_obs_t = ui_obs_t + ui_obs;
ui_orth_t = ui_orth_t + ui_orth;
```

At the end of iteration through all obstacles, in case we find there is atleast one real obstacle, the leader would only try to avoid the obstacle temporarily and would not try to traverse the target.

```
if obstacle_found > 0
    ui =  ui_obs_t + ui_orth_t;
else
    ui = ui_t + ui_orth_t;
end
```

Objectives of each agent:
Each agent independently has three tasks. Maintain a Delta disk formation with a specified agent/agents while avoiding the real obstacles and avoiding other agents which might come to close (in case there is no edge defined between them).

- **Delta-Disk Formation**
  Each agent tries to maintain a specified formation distance d_{ij}. If agent i is to maintain a formation with an agent j, it doesn't imply the agent j is also trying to maintain a formation with agent i. Each defined edge has a different gain for the weight function. The formation distance was also modified slightly to keep connectivity and avoid obstacles together as a system of agents.
  The connectivity is summarised below.

$$L = \begin{bmatrix} -1 & 0 & 0 & 0 & 1 \\ -1 & 2 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

  The formation control is guided by the function:

$$\frac{\left(\|x_i - x_j\| - d_{ij}\right)^2}{\left(\Delta - \|x_i - x_j\|\right)\left(\|x_i - x_j\| - \delta\right)}$$

  Below is the implementaiton of connectivity graph.

```
switch ID
    case 1
        a = find(~(nIDs-5));
        if a>0
            z = norm(xj(:,a)-xi);
            wij = 5*(z - d)*(z*((Delta-d)-(d-delta))
                + d*(Delta+delta)-2*delta*Delta)/(z*(
                Delta-z)^2*(z-delta)^2);
            ui_ag = ui_ag+ wij*R3*(xj(:,a)-xi); %
                control input from agent
        end

    case 2
        a = find(~(nIDs-1));
        if a>0
            z= norm(xj(:,a)-xi);
            wij = 1*(z - d)*(z*((Delta-d)-(d-delta))
                + d*(Delta+delta)-2*delta*Delta)/(z*(
                Delta-z)^2*(z-delta)^2);
```

```
15              ui_ag = ui_ag + wij*R2*(xj(:,a)-xi);
16          end
17
18          a = find(~(nIDs-5));
19          if a>0
20              z = norm(xj(:,a)-xi);
21              wij = 5e-3*(z - d)*(z*((Delta-d)-(d-delta
                    )) + d*(Delta+delta)-2*delta*Delta)/(
                    z*(Delta-z)^2*(z-delta)^2);
22              ui_ag = ui_ag+ wij*R2*(xj(:,a)-xi); %
                    control input from agent
23          end
24
25      case 3
26          a = find(~(nIDs-2));
27          if a>0
28              z = norm(xj(:,a)-xi);
29              wij = 1.5*(z - (1*d))*(z*((Delta-(1*d))
                    -((1*d)-delta)) + (1*d)*(Delta+delta)
                    -2*delta*Delta)/(z*(Delta-z)^2*(z-
                    delta)^2);
30              ui_ag = ui_ag + wij*R4*(xj(:,a)-xi);
31          end
32
33      case 4
34          a = find(~(nIDs-3));
35          if a>0
36              z = norm(xj(:,a)-xi);
37              wij = 0.5*(z - d)*(z*((Delta-d)-(d-delta)
                    ) + d*(Delta+delta)-2*delta*Delta)/(z
                    *(Delta-z)^2*(z-delta)^2);
38              ui_ag = ui_ag +  wij*R3*(xj(:,a)-xi);
39          end
40
41
42          a = find(~(nIDs-1));
43          if a>0
44              z = norm(xj(:,a)-xi);
45              wij = 5e-1*(z - d)*(z*((Delta-d)-(d-delta
                    )) + d*(Delta+delta)-2*delta*Delta)/(
                    z*(Delta-z)^2*(z-delta)^2);
46              ui_ag = ui_ag+ wij*R1*(xj(:,a)-xi); %
                    control input from agent
47          end
48
49              end
```

- **Collision avoidance with agents**
  In case an other agent comes very close to our agent, we introduce the edge tension function defined by

$$\varepsilon_{ij} = 5 \times 10^{-1} \frac{(\|x_i - x_j\| - d_{ij})^2}{(\Delta - \|x_i - x_j\|)(\|x_i - x_j\| - \delta)}$$

  that would repel our current agent from the other agents.This case would occur when there is no edge defined between two such agents.

```
for i=1:size(nIDs,2)
        z = norm(xj(:,i)-xi);
        if z < 1.3*delta
            w = 1e-1*sqrt(z)*(z-5*delta)/(z-delta)^3;
            ui_ca = ui_ca + w*(xj(:,i) - xi);
        end
end
```

- **Avoid obstacle:**
  Each agent uses the same algorithm used by the leader to avoid obstacles. The total obstacle avoidance control input and total orthogonal obstacle avoidance control input are updated.

$$u_{iobst} = u_{iobst} + u_{iobs};$$
$$u_{iortht} = u_{iortht} + u_{iorth};$$

Since we cannot afford to loose connectivity, each agent has to follow other agents/leader while avoiding obstacles and other agents in close proximity.

$$u_i = u_{iortht} + u_{iobst} + u_{ica} + u_{iag}$$

2.1 Observations

- It was seen that the simulations were quite different from the practical expirement. The agents took more time to take a turn which was the bottle-neck in my case.
- In the practical simulation, it was seen the system's connectivity and collision avoidance task was quite sensitive to leader's speed.
- A complete rigid formation would only the agents to pass between obstacle.
- A simple chain wouldn't make sure all the agents enter the warehouse.
- Hence, a primary chain with secondary weak edges were added to solve this problem of obstacle avoidance while entering the warehouse.

3 Search

Using simple Llyods dynamics, we search around in each cell by introducing artificial potential functions. The mean of this 'C' potential function is varied in such a fashion that the agents spiral outwards from their cells. They traverse the entire cell and hence eventually locate the markers.

```
1 mux = 0.005*currentTime*cosd(15*currentTime);
2 muy = 0.005*currentTime*sind(15*currentTime);
```

3.1 Observations

It was seen that this implementation is the simplest and effective. The success of this depends on the initial position of agents in the warehouse. On rare occasions, the agents seemed to hit the boundary of the house while searching.

************END************