Assignment 4: Sensor Network SNL

To run the SN Matlab functions, there needs to be a file named: CS6380_SN_Agent1.m in the home directory. The simulation function will make more copies of this as needed to make more sensor network agents (motes). The motes are placed in a 10x10 region.

Example Run

```
> clear all    % the agents use persistent variables, this clears them

> [res_50,motes_50] = CS6380_SN_sim(20,50,3,1); % 20 time steps; 50 motes;
                      3 for broadcast range; 1 means random placement
> trace_50 = CS6380_SN_mess2trace(res_50);  % extract content from messages
> clus_50 = CS6380_SN_clusters(trace_50);  % extract clusters from content
> CS6380_SN_show_motes(motes_50,clus_50);
```

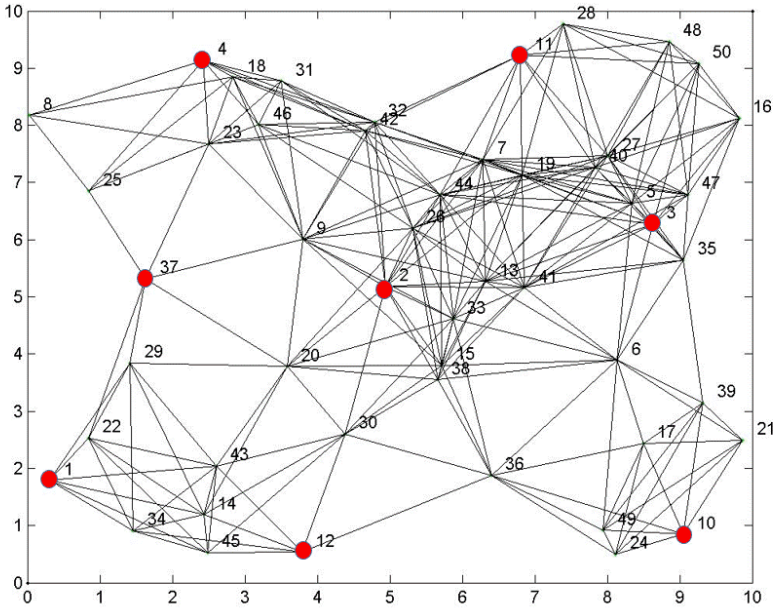

Figure 1. Example of SNL Protocol with 50 Motes

Agent Behavior

The agent is defined as a FSM. It takes a percept as input and returns actions. The percept consists of two parts: a message, and a temperature:

- percept.message:
  - .type (int): currently only uses ACL_INFORM (assigned value 8)
  - .sender (int): sender UID
  - .receiver (int): receiver UID (if 0 ➔ BROADCAST)
  - .content (int vec): SNAL (Sensor Network Agent Language) message

(1): SNAL code; one of:

- `MY_ID = 1;`
- `CLUSTER = 2;`
- `TEMP_REQ = 3;`
- `TEMP = 4;`
- `DISTS_REQ = 5;`
- `DISTS = 6;`

- `percept.temp (float): temperature at mote location`

The action returned has two components which each may have several entries:

- actions.do (will be [] for this assignment)
- actions.message (message list):
  - (i).message (message data type)

Each agent should also have a set of persistent variables to keep track of its state (e.g., my agent):

```
persistent UID state resolved leader time remaining neighbors distances
persistent my_leader locs time_start3 time_end3 known_leaders

if isempty(state)  % initial values
    UID = 1;
    state = 1;
    resolved = 0;
    leader = 0;
    time = 0;
    remaining = [UID];
    neighbors = [];
    distances = [];
    my_leader = 0;
    locs = [];
    time_start3 = -1;
    time_end3 = -1;
    known_leaders = [];
end
```

Function Descriptions

```
function [results,motes] = CS6380_SN_sim(max_time,num_motes,b_range,...
    type_init)
% CS6380_SN_sim - simulate sensor network agents
% On input:
%     max_time (int): max number of time steps
%     num_motes (int): number of sensor network agents
%     b_range (float): broadcast range for motes
%     type_init (int): picks type of sensor network layout
%       1: random (uniform 2D) in 10x10 region
%       2: grid: sqrt(num_motes) per side
%       3: special test layout: Nei(1,2), Nei(1,3), Nei(4,2), Nei(4,3)
%          should result in 1 and 4 as leaders
% Call:
%     clear all
%     [res_50,motes_50] = CS6380_SN_sim(20,50,3,1);
%     trace_50 = CS6380_SN_mess2trace(res_50);
%     clus_50 = CS6380_SN_clusters(trace_50);
%     CS6380_SN_show_motes(motes_50,clus_50);
% Author:
%     T. Henderson
%     UU
%     Summer 2014
%


function motes = CS6380_SN_init_motes(num_motes,b_range,type_init)
% CS6380_SN_init_motes - initializes the mote locations and neighbors
% On input:
%     num_motes (int): number of motes
%     b_range (float): maximum broadcast range
%     type_init (int): picks type of sensor network layout
%       1: random (uniform 2D) in 10x10 region
%       2: grid: sqrt(num_motes) per side
%       3: special test layout: Nei(1,2), Nei(1,3), Nei(4,2), Nei(4,3)
%          should result in 1 and 4 as leaders
% On output:
%     motes (mote data structure):
%       (i).x (float): x location
%          .y (float): y location
%          .nei (int vec): list of neighbor indexes (==UID's)
% Call:
%     motes = CS6380_init_motes(10,3,1);
% Author:
%     T. Henderson
%     UU
%     Summer 2014
%
```

```
function temp = CS6380_SN_temp(x,y)
% CS6380_SN_temp - return temperature at location [x;y]
% On input:
%     x (float): x location
%     y (float): y location
% On output:
%     temp (float): temperature at [x;y]
% Call:
%     tp = CS6380_SN_temp(2.3,5.7);
% Author:
%     T. Henderson
%     UU
%     Summer 2014
%
```

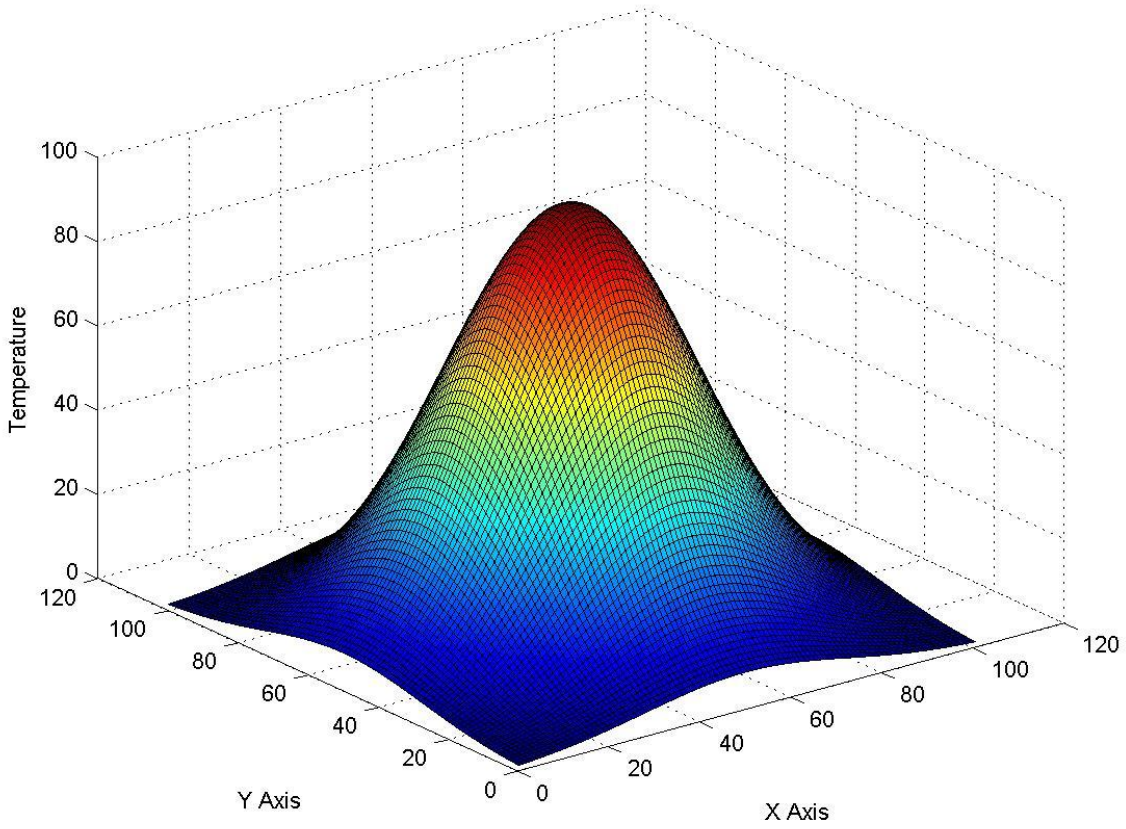

Figure 2. Temperature Function for SN Assignment

```
function messages = CS6380_SN_in_range(UID,a_mess,motes)
% CS6380_SN_in_range - return messages that are within range of mote
% On input:
%     UID (int): unique ID
%     a_mess (message list): list of messages in actions
%     motes (mote data structure): mote info
% On output:
%     messages (message list): messages in range of UID
% Call:
%     mess_list = CS6380_SN_in_range(3,prev_mess,motes);
% Author:
%     T. Henderson
%     UU
%     Summer 2014
%

function trace = CS6380_SN_mess2trace(messages)
% CS6380_SN_mess2trace - convert simulation trace to messages
% On input:
%     messages (message data structure): from SN simulation
%       (i).message
%            .type  (only type==8 used: ACL_INFORM)
%            .sender (int): UID of sender
%            .receiver (int): UID of receiver (or 0 if BROADCAST)
%            .content (int vec): SNAL(Sensor Network Agent Language) format
%                (1): MESSAGE_TYPE
%                (2:end): MESSAGE INFO
%          .RSS (float): received signal strength
% On output:
%     trace (trace data type):
%       (i).info: content from messages
% Call:
%     trace_50 = CS6380_SN_mess2trace(res_50);
% Author:
%     T. Henderson
%     UU
%     Summer 2014
%

function clusters = CS6380_SN_clusters(trace)
% CS6380_SN_clusters - extract clusters from message trace
%     any content message with SNAL command == 2 gives cluster
% On input:
%     trace (trace data structure): list of message contents
%       (i).info (int vec): [SNAL_cmd info]
% On output:
%     clusters (cluster data structure): clusters in sorted order
%      (i).cluster (int vec): [LEADER f1 f2 ... fn]
% Call:
%     clus_50 = CS6380_SN_clusters(trace_50);
% Author:
%     T. Henderson
%     UU
%     Sumer 2014
%
```

```
function CS6380_SN_show_motes(motes,clusters)
% CS6380_SN_show_motes - display motes, comm network and leaders
% On input:
%     motes (mote data structure): mote info
%       (i).x (float): x location
%       (i).y (float): y location
%       (i).nei (int vec): neighbors (UID) indexes
%     clusters (cluster data structure): clusters in sorted order
%      (i).cluster (int vec): [LEADER f1 f2 ... fn]
% On output:
%     Figure showing layout of SNL
% Call:
%     CS6380_SN_show_motes(motes_50,clus_50);
% Author:
%     T. Henderson
%     UU
%     Sumer 2014
%
```