Clustering in WSN using Self-Organizing Map Neural Network

Software: NetSim Standard v14.4, Visual Studio 2022, MATLAB (**Note**: Deep Learning toolbox mandatory)

Project Download Link:

https://github.com/NetSim-TETCOS/SOM_Clustering_in_WSN_v14_4/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

1 Objective

The goal of this project is to maximize the lifetime of a Wireless Sensor network using Self Organizing Map (SOM) based Neural Network algorithms for cluster head selection.

2 Introduction

We define the lifetime of a WSN as the time at which the power of half the sensors reaches zero (also called half-life of Network). Initially all sensors start with a fixed amount of energy. Subsequently energy is consumed during transmission, reception, and idle states. Packets are transmitted from sensors to their cluster head sensor and then it is forwarded to sink node through other cluster heads. The selection of the cluster heads is done using SOM. All MAC / PHY layer simulations are carried using NetSim while the cluster head selection using SOM algorithm is done using MATLAB.

3 Self-Organizing Map based Neural Network

We would be using a 2-Dimensional SOM to get a k sized cluster from n sensors located in 2D space using distance as a metric for clustering.
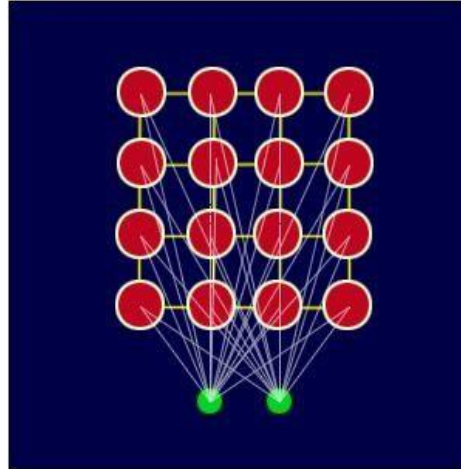
Figure 1: Neural network schematic with k 2D lattice points, where red nodes represent lattice points, green neurons denote the input layer, and connections between them illustrate the links.

As shown in the above figure, a neural network is created from k 2D lattice points (also known as nodes) each of which is connected with the input layer. Each link has an associated weight. As the input vectors are 2D points here, there are 2 neurons in input layer of neural network. Each node has a topological position (x coordinate and y coordinate) and also a weight vector of 2 dimensions (one weight for each dimension).

So, with input vectors and weight vectors, the SOM algorithm explained below, orders the weight vectors in a way that represents similarities with input vectors.

The algorithm consists of the following steps:

1. Each node's weight are randomly initialized.

2. Choose an input vector and find that node whose weight vector is closest to the chosen point. The most common method to calculate distance is finding the Euclidean distance. This node is called BMU (Best matching unit).

3. The neighborhood of BMU is defined as all the nodes lying within its radius of influence. The no of neighbors decreases over time because Radius of influence is decreased over time.

4. The weight vector associated with neighbor node (and BMU too) is updated using following equation –

$$iw(q) = iw(q-1) + \alpha(p(q) - iw(q-1))$$

Where p(q) is the input vector chosen and iw(q-1) is the weight vector associated with node i and iw(q) is the updated value of weight vector.

5. Repeats from step 2 till the iteration limit has been reached.

The above procedure is repeated for large no of iterations (chosen as 200 in our example)

There would be k output nodes in the neural network where each output node is associated with some pattern or cluster in the input point.

Each point would be passed through network and suppose i^{th} output node has highest value, then this point belongs to the cluster i.

The topology function of the k nodes and the distance function used to evaluate distance between sensor and node can be chosen from a given set of values as below.

4 Topology

The neurons in the layer of an SOM are arranged originally in physical positions according to a topology function. The function gridtop, hextop, or randtop can arrange the neurons in a grid, hexagonal, or random topology.

1. **The gridtop topology starts with neurons in a rectangular grid of dimensions which you may specify**

Suppose you want to classify n points into k clusters. Then, you can start with k neurons arranged in rectangular grid of dimensions [k1 k2] such that k1*k2=k. e.g.- pos = gridtop([2, 3]) pos =

$$
\begin{array}{cccccc}
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 2 & 2
\end{array}
$$

Suppose you had chosen dimensions to be [3, 2], then you would get following configuration of neurons pos = gridtop([3, 2]) pos =

$$
\begin{array}{cccccc}
0 & 1 & 2 & 0 & 1 & 2 \\
0 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

2. **In hextop topology, neurons are initially arranged in a hexagonal pattern.**

In hextop topology, neurons are initially arranged in a hexagonal pattern.

e.g. A 2-by-3 pattern of hextop neurons is generated as follows: pos = hextop([2, 3]) pos =

$$
\begin{array}{cccccc}
0 & 1.0000 & 0.5000 & 1.5000 & 0 & 1.0000 \\
0 & 0 & 0.8660 & 0.8660 & 1.7321 & 1.7321
\end{array}
$$

Hextop is the default pattern for SOM networks generated by selforgmap

3. **The randtop function creates neurons in a random pattern in the specified dimensions.**

   Pos=randtop ([2, 3]) Pos =

$$
\begin{array}{cccccc}
0 & 0.42 & 0.29 & 0.87 & 0.07 & 0.43 \\
0 & 0.01 & 0.26 & 0.48 & 1.32 & 1.33
\end{array}
$$

5 Distance functions

Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist, and mandist

The link distance from one neuron is just the number of links, or steps that must be taken to get to the neuron under consideration.

The dist is Euclidean distance from neuron to a point.

The mandist calculates the Manhattan distance between points.

Creating a Self-Organizing Map Neural Network (selforgmap)

SOM is created using selforgmap function whose syntax is as given below.

Selforgmap (dimensions, coversteps, initNeighbour, topologyFunction, distanceFunction)

Where the parameters can take following value-

1. Dimensions is a row vector of dimension sizes of the initial neurons. Default value= [8 8].
2. coversteps is number of training steps to cover the whole input dataset initially (Default=100)
3. initNeighbour is the size of initial neighbourhood. (default =3)
4. topologyFunction is the initial topology of neurons (default ='hextop')
5. DistanceFunction is neuron distance function (default='linkdist')

Suppose you want to cluster n points located in 2D space into k clusters based on Euclidean distance-

Let x be a matrix with dimension 2*n which contains the coordinate of points.

net = selforgmap([2 k/2], 100, 3, , 'gridtop', dist');

You can set the no of iterations the neural network will train using net.trainParam.epochs=1000;

Network is trained using train (network, dataset) as net = train (net, x);

To get the cluster id of the points by passing them as input to the learnt neural network- y=net(x);

y would be a 4*n matrix. The ith column of y would be the output for the ith point and all the entries in the column would be zero except one which is the cluster to which that points belong or more precisely the node which is the cluster head of the ith point.

To get cluster-id in range (1, k)- IDX=vec2ind(y);

Where IDX is a n length vector.

Now we have to get the geometrical centroid of each cluster which can be obtained by iterating through all the points that belong to that cluster and finding mean of their position vectors.

On running the above code, a GUI Deep Learning Toolbox appears in which there are several visualizations of the network that is learnt like SOM topology, SOM neighbour connection, SOM neighbour distances, SOM input planes, SOM sample hits, SOM Weight positions.

6 Interfacing WSN Simulation in NetSim with SOM algorithm running in MATLAB

SOM based clustering is implemented in NetSim by Interfacing with MATLAB for the purpose of running the SOM algorithm. The sensor coordinates are fed as input to MATLAB and Self Organizing map neural network algorithm that is implemented in MATLAB is used to dynamically perform clustering of the sensors into n number of clusters.

In addition to clustering, we also determine the cluster head of each cluster mathematically in MATLAB. The distance of each sensor from the centroid of the cluster to which it belongs is calculated. Then the sensor which has the least distance is elected as the cluster head.

From MATLAB we get the cluster id of each sensor, cluster heads of each cluster and the size of each cluster.

All the above steps are performed periodically which can be defined as per the implementation. Each time the cluster members and the cluster heads are determined based on the current position and they are not fixed.

The codes required for the mathematical calculations done in MATLAB are written to a **som_optimization.m** file and this file available in MATLAB folder under bin_x64 of SOM_Optimization_Workspace

A **SOM_Clustering.c** file is added to the DSR project which contains the following functions:

- **fn_NetSim_som_clustering_CheckDestination();** //This function is used to determine whether the current device is the destination.
- **fn_NetSim_som_clustering_GetNextHop();** //This function statically defines the routes within the cluster and from cluster to sinknode. It returns the next hop based on the static routing that is defined.
- **fn_NetSim_som_clustering_IdentifyCluster();** //This function returns the cluster id of the cluster to which a sensor belongs.
- **fn_NetSim_som_clustering_run();** //This function makes a call to MATLAB interfacing function and passes the inputs from NetSim (i.e) the sensor coordinates, number of clusters and the sensor count.
- **fn_netsim_som_form_clusters();** //This function assigns each sensor to its respective clusters based on the cluster id's obtained from MATLAB.
- **fn_netsim_assign_cluster_heads();** //This function assigns the cluster heads for each cluster based on the cluster head id's obtained from MATLAB.

- **fn_NetSim_som_Clustering_Init();** //This function initializes all parameter values.

Static Routing

Static Routing is defined in such a way that the sensors in the cluster send the packets to the cluster head. The cluster head then directly sends the packets to the destination (sinknode).

If the current sensor is the source device and if it is not a cluster head, then its next hop is its cluster head.

If the current sensor is the source device and if it is a cluster head, then its next hop is the destination (i.e.) the sinknode.

If the current sensor is not the source, then the packet is sent to the destination (i.e.) the sinknode.

7 Steps to run SOM Clustering Code in NetSim

1. Import the Workspace **SOM_Optimization_Workspace**
2. Add the following MATLAB install directory path in the Environment PATH variable
   <MATLAB_INSTALL_DIRECTORY>\bin\win64
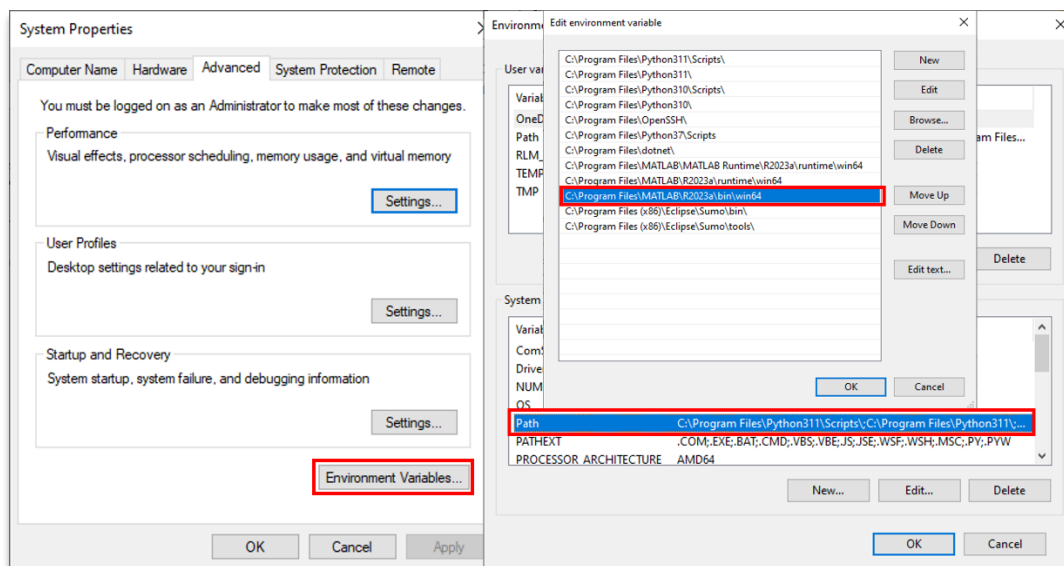   For eg: C:\Program Files\MATLAB\R2021b\bin\win64

m

Figure 2: Environment variable PATH

Note: If the machine has more than one MATLAB installed, the directory for the target platform must be ahead of any other MATLAB directory (for instance, when compiling a 64-bit application, the directory in the MATLAB 64-bit installation must be the first one on the PATH).

3. Open Command prompt as admin and execute the command "matlab -regserver". This will register MATLAB as a COM automation server and is required for NetSim to start MATLAB automation server during runtime.

4. Open the Source codes in Visual Studio by going to Your work-> Source code and Clicking on Open code button in NetSim Home Screen window.

5. Under the DSR project in the solution explorer you will be able to see **Som_clustering.c** files which contain source codes related to interactions with MATLAB and handling clustering in NetSim respectively.

8 Example

1. Run NetSim in Administrative mode.

2. **SOM_Optimization_Workspace** comes with a sample configuration that is already saved. To open this example, go to Your Work and click on the **Som_clustering_Example**.

3. The saved network scenario consists of 64 sensors uniformly placed in the grid environment along with a sink node forming a Wireless Sensor Network. Traffic is configured from each sensor node to the Sink Node.
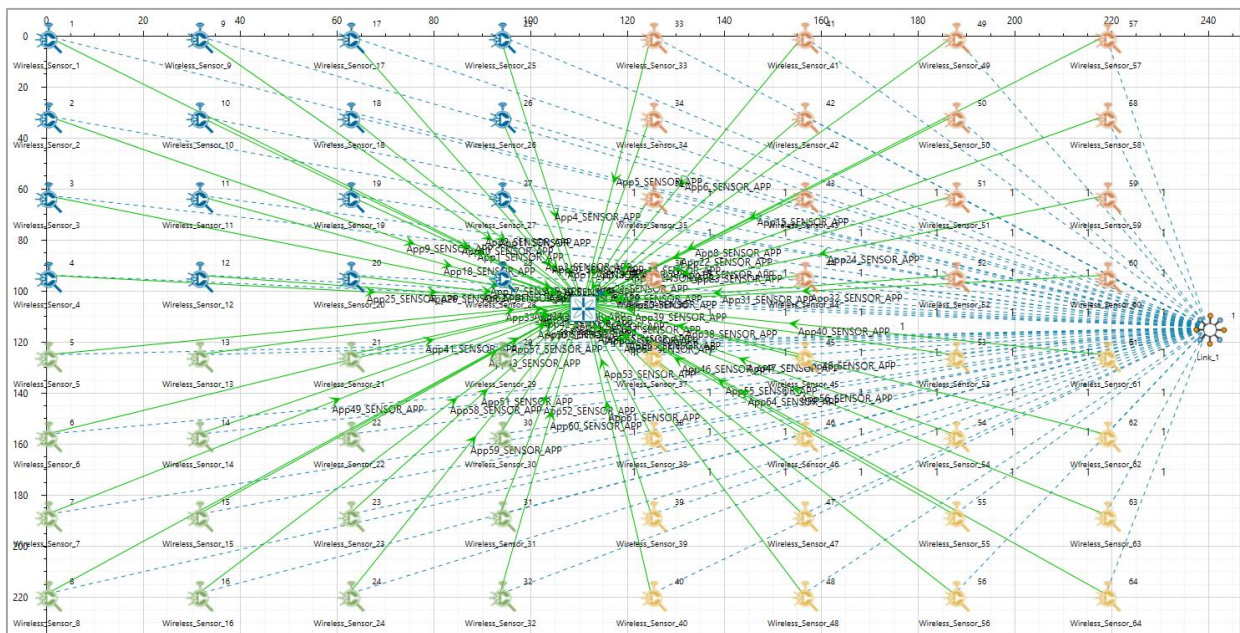

Figure 3: Network Topology

4. Run the simulation and press any key to continue.

5. It will open MatlabInterface.exe console window. You will observe that as the simulation starts in NetSim, MATLAB gets initialized and graph associated with energy consumption in the sensor network is plotted during runtime.

6. There are two algorithms implemented to find the best clusters and cluster heads which uses SOM with distance as metric and other is modified version of the first algorithm where a function of both remaining power and the distance from cluster head is minimized over all the sensors in the cluster

to get the cluster head with least distance from geometrical centroid of cluster and maximum remaining power.

Case 1:

SOM using distance as a metric to identify the cluster head (Clustering_Method = 1)

After importing the workspace, open **som_optimization.m** MATLAB file that is present in the **bin_x64 -> MATLAB** folder of the imported workspace, for changing the clustering methods and to verify different cases
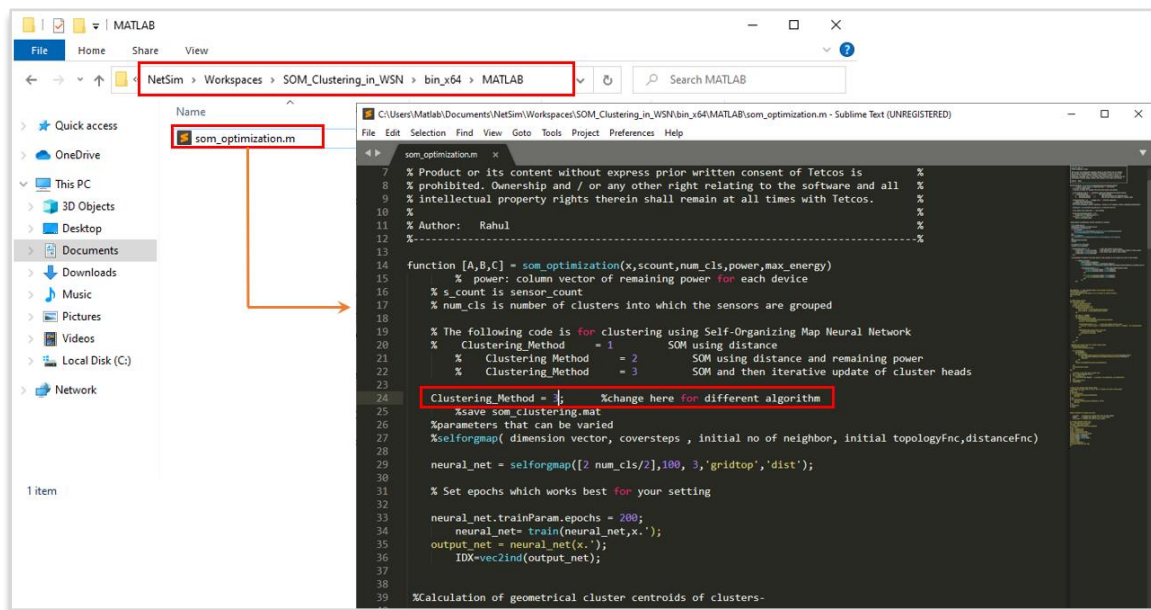

Figure 4: MATLAB file path and Clustering method for different cases

For **Case 1** change the Clustering_method to 1, similarly for **case 2**: Clustering_method =2 and for **Case 3**: Clustering_method = 3.

The clusters would be created to minimize the sum of distance between the sensor and the sensor which is cluster head. The remaining power in each sensor is not considered in this algorithm.
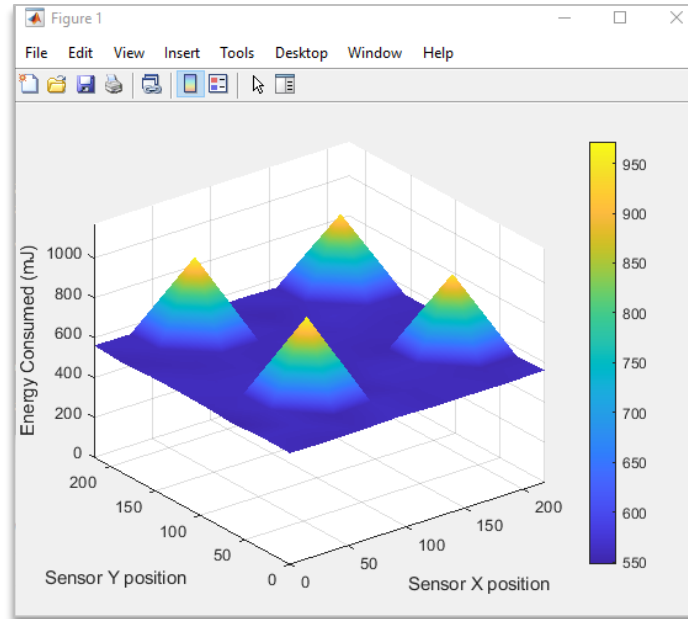
Figure 5: Plot for power consumption

64 sensors are placed evenly on x-y plane and each sensor is given a fixed amount of initial power (100 in this case). The number of clusters has been fixed to 4.

The z axis represents the power consumed while the sensors are placed on the x, y plane.

It can be seen from the plot, there are 4 peaks in the plot corresponding to 4 sensors that will be selected as the cluster heads. Since the sensors are static, there are same cluster heads and cluster during the whole simulation period.

9 Neural Network Training GUI

To view the Neural Network Training window, modify the following line in the som_optimization.m file which can be found in the *<current_workspace_directory>/bin_x64/MATLAB/* path:

neural_net.trainParam.showWindow = false;

to

neural_net.trainParam.showWindow = true;

After performing the above changes, rerun the simulation, during which the Neural Network Training window will appear, with several option buttons such as:

SOM Topology: Defines the structure or layout of the Self-Organizing Map (SOM) in the Neural Network Training GUI.

SOM Neighbour connections: Represents the connections between neighbouring nodes in the SOM within the GUI, influencing network learning.

SOM Neighbour distances: It shows the distance of sensors from cluster centres as computed using distance function and the neighbourhood of each cluster centres are shaded in different colours.

SOM Input Planes: Refers to the input data layers or dimensions considered in the SOM within the GUI for training the neural network.

SOM Sample Hits: Indicates the frequency with which each node in the Self-Organizing Map is selected during training

SOM Weight Positions: The cluster centres are shown at their weight vector (using them as position vector) along with all the sensors in the WSN.
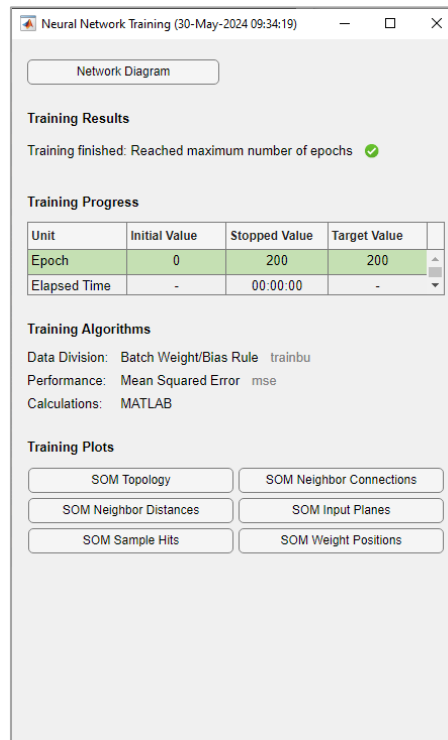

Figure 6: Select SOM Topology in Neural Network Training

Clicking on **SOM Weight Position** you would get the following plot.
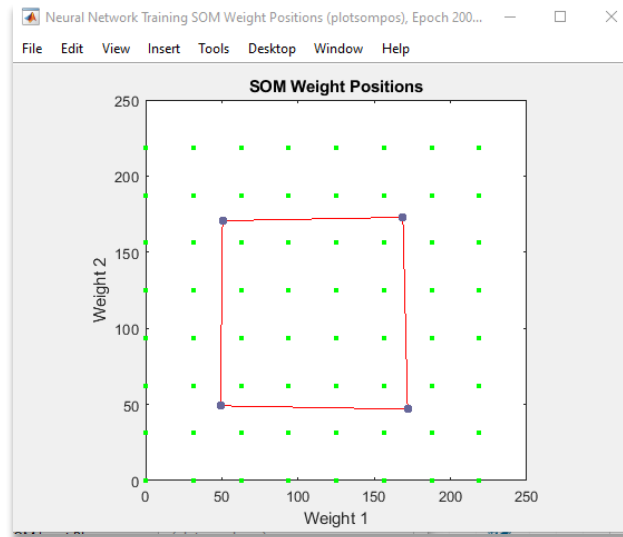
Figure 7: SOM Weight Positions

Here the four points in blue show the final weight positions of the trained neural network. The green points are the sensors whose position vectors were used as input to the neural network while training. Weight1 and Weight2 are corresponding to x coordinate and y coordinate of the position vectors of input.

File log.txt is created in the MATLAB root directory where the SOM_optimization.m file was placed. It contains the location of cluster heads and the sensor no which is cluster head from the start of simulation.
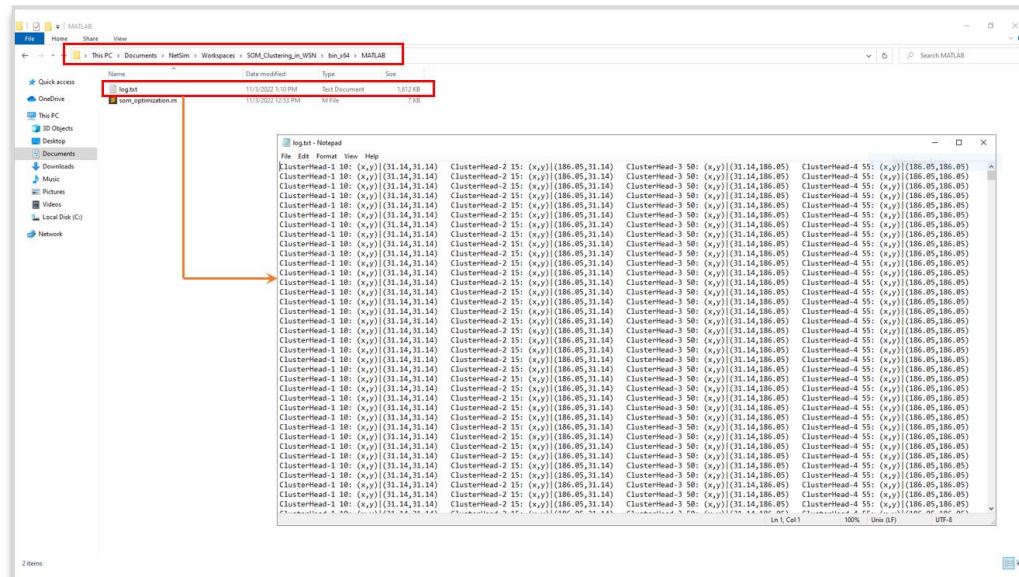

Figure 8: Log.txt file location along with the sensor co-ordinates

Steps are similar for both **Case 2** and **Case 3**

Case 2:

Modified SOM using power and distance as metric for electing cluster head

(Clustering_Method = 2 in the MATLAB file SOM_optimisation.m)

Algorithm:

SOM library of MATLAB is used to find the cluster id of each sensor and the sensor for which the objective function (composed of power and distance from cluster center) is minimum is chosen as cluster head.

The power consumption obtained using this is close to that of kmeans in the uniform placement of sensors, but it might differ in case of complex distribution of placement of sensors.
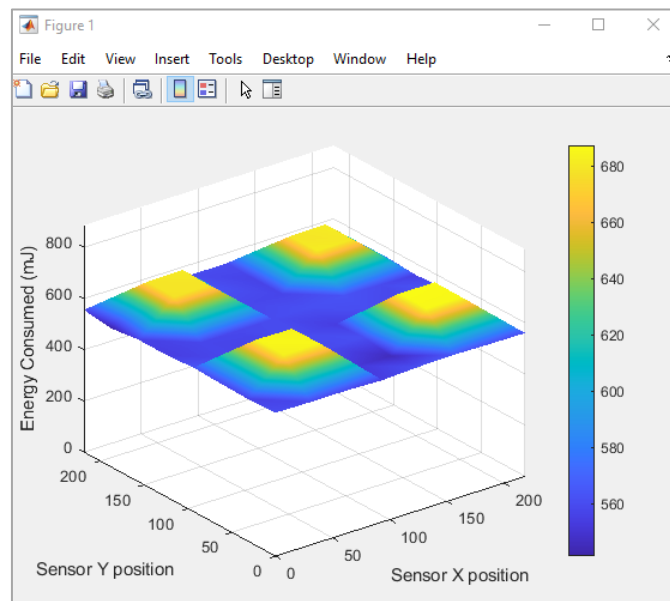

Figure 9: plot for power consumption of sensors

In the initial phase the plot resembles the previous one. But after some time, since the power associated with cluster heads would decrease fast and so, there would be new cluster head whose distance from geometrical centroid of cluster is considerably low and power is also high. Hence as the time passes, it can be observed that the power is consumed by all the sensors at approximately the same rate.

There are no peaks in this plot unlike the previous one because modified SOM considers the power level of each sensor and thus each sensor will be appointed as the cluster head in its respective cluster.

Case 3

Recalculating clusters iteratively after getting cluster using SOM initially.

Algorithm:

Initially, cluster is evaluated using SOM which uses distance as metric. The cluster to which each sensor belongs to is known. Now, cluster head is chosen as the sensor for which the objective function which constitutes remaining power and the distance from geometrical centroid of cluster to the sensor, is minimized.
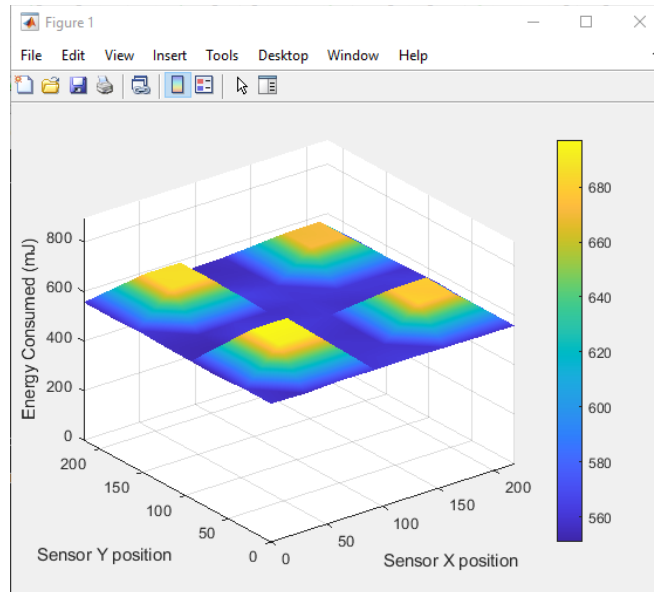

Figure 10: Plot for Power Consumption of Sensors

After this cluster is recalculated and each sensor is assigned to the cluster whose cluster head is closest to it. Cluster heads and then the cluster is computed iteratively.