

# Object-Centric Intelligence: Sensor Network and Thermal Mapping

**Naresh Yamani**

Design and Creative Technology, School of Engineering  
Auckland University of Technology

A thesis is submitted to Auckland University of Technology in  
fulfilment of the requirements for the degree of  
Doctor of Philosophy

August, 2013

# Acknowledgements

I am grateful to my primary supervisor Professor Adnan Al-Anbuky for the continuous support and cooperation in the hours of need and for his expert technological and innovative advices in the area of Wireless Sensor Networks. He was a beacon light, whose constant efforts and encouragement proved to be a parallel stimulus in completing this project successfully. I would like to thank AUT University for providing me this opportunity and financial support.

I offer my profuse thanks with humble reverence to Dr. Clyde Daly and Dr. Nicola Simmons, Carne Technologies Ltd, Cambridge for their support. They are well known in the meat industry both in New Zealand and internationally.

The funding for this work was provided by the Foundation for Research, Science and Technology (FRST) as sponsored by Carne Technologies Ltd under the programme contract number CARN0801. I would like to thank and acknowledge their continued support.

This thesis is dedicated to my loving wife, Jyothsna Yamani and loving son Kanishka Gandharva Yamani for providing me with a strong support which helped me undertake this doctoral journey and successfully prepare this thesis.

# Declaration

I hereby declare that this submission is my own work and that to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Name: Naresh Yamani

Sign:

Date:

# Contents

---

1	Introduction.....	1
1.1	Introduction to Wireless Sensor Networks.....	1
1.2	Wireless Sensor Networks: Target Applications.....	2
1.2.1	WSN Background .....	3
1.2.2	WSN research challenges.....	5
1.3	Artificial Neural Networks Background .....	6
1.3.1	On-line Learning Algorithms.....	7
1.3.2	Gradient Descent Learning Algorithm.....	8
1.3.3	Back-Propagation algorithm .....	9
1.3.4	Multiple local optima and epochs .....	9
1.4	Thermal Mapping Background.....	10
1.5	Motivation .....	11
1.6	Problem Statement and Approach .....	13
1.6.1	Research Objectives .....	13
1.7	Contributions .....	13
1.8	Publications .....	14
1.9	Structure of the Thesis.....	15
2	Literature Analysis.....	17
2.1	Cold Room Monitoring and Tracing Systems.....	17
2.2	Mapping and Thermal Mapping.....	18
2.3	Artificial Neural Networks: A Review.....	19
2.3.1	Soft computing in physical fields.....	20
2.3.2	ANNs applied to thermal mapping .....	20
2.4	Wireless Sensor Networks: A Review .....	21
2.4.1	ANNs applied to wireless sensor works .....	22
2.4.2	Nodes placement and minimization .....	23
2.5	Data Collection and Query Processing in WSN.....	23
2.6	Research focus.....	25
3	Environment for nWSN Development .....	27
3.1	Introduction .....	27
3.2	Simulation Environments .....	28



3.2.1	IBM's Mote Runner .....	28
3.2.2	Mote Runner architecture.....	29
3.2.2.1	Flexsim Discrete Event Simulation.....	30
3.2.2.2	Visualization and 3D engine .....	31
3.2.2.3	Model views .....	31
3.2.2.4	Flexsim objects.....	31
3.2.3	Artificial Neural Net Software .....	32
3.2.4	Voxler visualization program.....	33
3.3	Hardware Selection for WSN .....	34
4	Spatial Analysis: Thermal Mapping .....	35
4.1	Introduction .....	35
4.2	Ideology of the nWSN.....	35
4.3	Object-Centric Intelligent Environment .....	36
4.4	Thermal Mapping Methodologies .....	38
4.4.1	Thermal mapping using Shepard's algorithm.....	39
4.4.2	Neural net based solution to thermal mapping.....	41
4.4.3	Data collection and node implementation.....	42
4.5	Computation and Thermal Coverage Focus .....	43
4.5.1	Space division based on layers.....	44
4.5.2	Coverage aspects.....	45
4.6	Nodes Minimization Approach .....	46
4.6.1	k-neighbour search for nodes.....	46
4.6.2	k-neighbour search for a query point .....	47
4.7	Extending the Solution through Space Partitioning .....	48
4.7.1	nWSN structure for subspace partitioning .....	49
4.8	Time Synchronization for nWSN Data Processing.....	51
4.8.1	An algorithmic approach – QnDP algorithm .....	52
5	nWSN Simulation and Validation .....	54
5.1	Introduction .....	54
5.1.1	Temperature profile.....	55
5.1.2	Transient model behaviour.....	58
5.2	Thermal Mapping Based on a Fixed Sensing Points with Single Cluster Head/Infrastructural Node .....	59
5.2.1	Impact of NN internal architectural parameters.....	61

5.3	Thermal Mapping Based on Random Sensing Points .....	63
5.3.1	Thermal mapping using Shepard's Algorithm.....	63
5.3.2	Thermal mapping using neural net approach.....	66
5.4	Thermal Mapping Experimental Scenarios .....	68
5.5	nWSN Testbed Experimentation.....	73
6	nWSN Implementation Factors .....	77
6.1	Components of the nWSN Implementation .....	77
6.2	Approach for Query Based nWSN Spatial Thermal Mapping.....	78
6.2.1	Query based nWSN data processing framework .....	79
6.3	Time Synchronization and its Implementation.....	82
6.3.1	Configuration and setup for time synchronization.....	84
6.4	Neural Net Cluster Dynamic Grouping .....	87
6.4.1	Sequence diagram and process flow .....	88
6.5	Minimizing Nodes Approach .....	89
6.5.1	Sequential search and nodes minimization .....	90
6.5.2	Bayesian approach to identify more mutually influenced nodes .....	92
6.5.3	Nodes minimization simulation results and discussion .....	93
7	Cool Storage in a Meat Plant: A Case Study .....	95
7.1	Introduction .....	95
7.2	Modeling Scenario in a Cool Storage.....	96
7.3	Simulation environment setup and experimentation .....	97
7.3.1	(A) Predicting the temperature for better coverage by placing nodes in three layers. ....	99
7.3.2	(B) Thermal analysis at the subspace surface region.....	102
7.3.3	(C) Thermal analysis at increased carcass inter-arrival time. i.e. Carcass count reduced to 86. ....	104
7.3.4	(D) Thermal mapping of the space where the temperature fluctuation is taken between $\pm 4^{\circ}\text{C}$ .....	105
8	Conclusions and Future Directions.....	108
8.1	Conclusions .....	108
8.2	Future Work and Directions .....	110
9	References.....	113
A.	Appendices .....	126
A.1	Hardware Environment:.....	126

A.1.1 Infrastructural node .....	127
A.1.2 Portable node .....	128
A.1.3 Java script file for web-based data monitoring.....	130
A.1.4 Time synchronization – QBnWSN framework: initialization.....	132
A.1.5 Time synchronization – QBnWSN framework: infrastructural node.....	132
A.1.6 Time synchronization – QBnWSN framework: portable node .....	135
A.2 Simulation Environment: .....	143
A.2.1 Nodes deployment in simulation environment.....	144
A.2.2 Nodes reset to initialize code.....	150
A.2.3 Grid generation code in simulation .....	153
A.2.4 Internal functions code .....	154
A.2.5 Nodes minimization approach.....	156
A.2.6 Portable nodes object's update .....	157
A.3 GUI for Nodes Minimization.....	159
A.4 GUI for Setup Scenarios .....	159

# List of Figures

---

Figure 1.1 Wireless sensing node .....	4
Figure 1.2 Activation of neurons and its layers .....	6
Figure 1.3 Thermal mapping - (a) Thermal map (b) Contour map.....	10
Figure 1.4 Concept Diagram of Object-Centric Environment.....	11
Figure 2.1 Research focus Intersection areas.....	26
Figure 3.1 Mote Runner Architecture [110] .....	29
Figure 3.2 SeNSe Testbed Architecture.....	30
Figure 3.3 Schematic diagram of the cold storage.....	32
Figure 3.4 GUI built in Flexsim to setup a scenario .....	32
Figure 3.5 Volumetric rendering data produced by voxler.....	33
Figure. 3.6 Wireless sensor node from Atmel .....	34
Figure 4.1 Ideology of the problem .....	36
Figure 4.2 Conceptual diagram of the object-centric environment.....	37
Figure 4.3 Objects activity – an overview of the object centric model.....	38
Figure 4.4 Predicting temperature at a point using Shepard’s algorithm.....	39
Figure 4.5 Pseudo code for Shepard’s algorithm .....	40
Figure 4.6 FFNN for thermal mapping .....	41
Figure 4.7 Pseudo code for nWSN cluster head .....	43
Figure 4.8 Space divisions into layers .....	45
Figure 4.9 k-neighbour search algorithm .....	47
Figure 4.10 Subspace partitioning and their regions.....	48
Figure 4.11 nWSN Structure using 4 clusters.....	49
Figure 4.12 NMi Inode and Pnode interaction among the nodes in nWSN.....	51
Figure 4.13 QnDP Memory buffer model.....	52
Figure 4.14 QnDP Algorithm.....	53
Figure 5.1 Contour map of the thermal profile based on the sensing points .....	56
Figure 5.2 Sensing location points (a) pattern with 5 nodes (b) pattern with 9 nodes (c) pattern with 13 nodes .....	59
Figure 5.3 MAE and RMSE comparison between Shepards and ANN.....	60
Figure 5.4 ANN Parameters comparison for training .....	60
Figure 5.5 Time comparison at different MLP layers and epochs.....	62
Figure 5.6 Sensing node distribution over the space.....	63
Figure 5.7 Actual vs Predicted temperatures .....	64
Figure 5.8 (a) Actual thermal profile (b) Predicted thermal profile.....	65
Figure 5.9 (a) Actual thermal profile (b) Predicted thermal profile.....	65
Figure 5.10 Actual and predicted temperatures using ANN approach .....	66
Figure 5.11 Actual and predicted temperatures .....	66
Figure 5.12 (a) Actual thermal profile (b) Predicted thermal profile.....	67
Figure 5.13 (a) Actual thermal profile (b) Predicted thermal profile.....	67
Figure 5.14 Schematic diagram of the experiment .....	68
Figure 5.15 RMS error Vs No. Infrastructural Nodes.....	70
Figure 5.16 RMS error Vs Room volume.....	71
Figure 5.17 RMS Error Vs Training Data.....	72
Figure 5.18 Testbed layout at SeNSe.....	73
Figure 5.19 Temperature profile of each sensor node.....	74
Figure 5.20 Actual room temperature contour map .....	74
Figure 5.21 Contour map of the testing data set .....	76
Figure 5.22 Contour map of the predicted data set .....	76

Figure 6.1 QBnWSN Framework .....	79
Figure 6.2 WSN Nodes arrangement in the application scenario .....	81
Figure 6.3 Node location change with time .....	82
Figure 6.4 Flow diagram for query processing .....	83
Figure 6.5 nWSN configuration setup .....	84
Figure 6.6 QB-nWSN Framework results interface.....	85
Figure 6.7 Volumetric temperature precision .....	86
Figure 6.8 Query and response times for 4 nodes.....	86
Figure 6.9 NMi Inode process flow .....	87
Figure 6.10 Sequence diagram of the nWSN process flow .....	88
Figure 6.11 Minimizing nodes approach flowchart .....	91
Figure 6.12 Nodes hierarchy .....	91
Figure 6.13 Precision Vs Nodes (at 50) .....	94
Figure 6.14 Precision Vs Nodes (at 100) .....	94
Figure 7.1 Thermal profile of a beef within the first 24hrs.....	95
Figure 7.2 pH variation of a beef carcass.....	96
Figure 7.3 Tenderness variation of a beef carcass .....	96
Figure 7.4 Schematic diagram of the cool storage .....	97
Figure 7.5 GUI to setup experimentation.....	98
Figure 7.6 Schematic arrangement of infrastructural sensor nodes .....	99
Figure 7.7 Portable node layers within the cool store.....	100
Figure 7.8 Volumetric temperature precision at dip .....	100
Figure 7.9 Volumetric temperature precision at peak.....	101
Figure 7.10 MAE at peak and dip for Inf. sensor nodes .....	101
Figure 7.11 MAE at nodes 3, 8, 13, 18 at peak.....	102
Figure 7.12 MAE at nodes 3, 8, 13, 18 at dip .....	102
Figure 7.13 <i>MAE at peak and dip for Inf. sensor nodes</i> .....	103
Figure 7.14 MAE at nodes 3, 8, 13, 18 at peak.....	103
Figure 7.15 MAE at nodes 3, 8, 13, 18 at dip .....	103
Figure 7.16 Volumetric temperature precision at peak.....	104
Figure 7.17 Volumetric temperature precision at dip .....	104
Figure 7.18 Volumetric temperature precision at peak.....	105
Figure 7.19 Volumetric temperature precision at dip .....	105
Figure 7.20 Volumetric temperature precision at peak.....	106
Figure 7.21 Volumetric temperature precision at dip .....	106
Figure 7.22 MAE at nodes 3, 8, 13, 18 at peak.....	107
Figure 7.23 MAE at nodes 3, 8, 13, 18 at dip .....	107

# List of Tables

---

Table 3.1 Key features of the Mote Runner .....	28
Table 5.1 Temperature profile used for the sensing data .....	56
Table 5.2 Thermal profile used for the test set .....	57
Table 5.3 Initial neural net parameters for mapping .....	61
Table 5.4 Actual and Predicted temperatures using Shepard's algorithm .....	64
Table 5.5 Additional deployed nodes at hot spots .....	65
Table 5.6 Model run scenarios .....	69
Table 5.7 RMS Errors – Inf. Nodes changing 8 to 40 at 2000m <sup>3</sup> Volume and 20 Training data .....	70
Table 5.8 RMS errors – Room volume changing 2000m <sup>3</sup> to 12500m <sup>3</sup> at 24 Inf. Nodes and 20 Training data .....	71
Table 5.9 RMS Errors – Training data set changing 10 to 120 at 8 Inf. Nodes and 4500m <sup>3</sup> volume .....	72
Table 5.10 Training data confidence plot .....	75
Table 5.11 Validation data within the training set confidence plot .....	75

# List of Symbols and Abbreviations

---

A/D	Analog-to-Digital
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASM	Assembly
CFD	Computational Fluid Dynamics
DBM	Data Based Mechanistic
DBS	Dual Buffer Search
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEM	Finite Element Methods
FFNN	Feed Forward Neural Network
GUI	Graphical User Interface
I/O	Input and Output
ID	Sensor node short address
ISM	Industrial Scientific and Medical
k-NNA	k Nearest Neighbour Algorithm
LAN	Local Area Network
LED	Light Emitting Diode
MAC	Media Access Control
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-layer Perceptron
$N_1$	Infrastructural/Stationary sensor node numbered 1
NMi	Nodes Message interaction
NN	Neural Network
nWSN	neuro Wireless Sensor Network
OCT-Map	Object-Centric Thermal Mapping
$P_1$	Portable sensor node (carried by an object) numbered 1
QB-nWSN	Query Based nWSN
QnDP	Query based nWSN Data Processing
$R^2$	Correlation coefficient
RAM	Random Access Memory

RF	Radio Frequency
RFID	Radio Frequency Identification
RMSE	Root Mean Square Error
SVM	Support Vector Machines
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
XML	Extended Mark-up Language
$\alpha$	Thermal diffusivity
$b$	Bias
$\delta$	Residual or adjustment parameter
$\eta$	Predefined learning rate
$\rho$	Density
$\chi$	Set of sensor nodes
$\lambda, \emptyset$	Sensor dependent parameters
$B[k]$	Memory buffer at sensor node level
$c_p$	Specific heat capacity
$C_{x_i y_i z_i}$	Probabilistic coverage of a point
$d_k$	Euclidean distance
$e$	Epoch
$E$	Error between output and target
$h$	Heat transfer coefficient
$k$	Thermal conductivity
$L_I$	Shortest path length
$o_{ij}$	Actual output
$o_k$	Output of node $k$
$p_D$	Packet Data
$P_t$	Training sample
$q$	Query point
$\mathbb{R}$	Confined space or region
$R_{ci}$	Communication radii
$\mathbb{R}^m$	$m$ -dimensional Euclidean space
$S$	Dataset
$Se$	Sensitivity of the sensor node
$S_t$	Memory vector
$t_i$	Time point for data synchronization



$T$	Temperature
$T_{min}$	Minimum temperature
$T_{max}$	Maximum temperature
$\bar{T}_{Act,i}$	Average actual temperature for $i^{th}$ observation
$\bar{T}_{Est,i}$	Average predicted temperature for $i^{th}$ observation
$v_{ij}$	Desired output
$\Delta w$	Small increment to the model parameter
$w_{ij}$	Weight of the connection from node $i$ to node $j$
$w_k$	Weight function for a given node
$w_{jk}^{new}$	New value of the weight from node $j$ to node $k$
$w_{jk}^{old}$	Old value of the weight from node $j$ to node $k$
$(x,y,z)$	Location coordinates at a given point
$y$	Network output

# Abstract

Quality of product is an important aspect in many commercial organizations where storage and shipment practices are required. Temperature is one of the main parameters that influence quality and temperature treatments of agricultural products therefore require special attention.

The temperature variation in a meat chiller has a significant effect on tenderness, colour and microbial status of the meat, therefore thermal mapping during the chilling process and during chilled shipment to overseas markets is vital. The literature indicates that deviations of only a few degrees can lead to significant product deterioration. There are several existing methods for thermal mapping: these includes Computational Fluid Dynamics (*CFD*), Finite Element Methods (*FEM*) for examination of the environmental variables in the chiller. These methodologies can work effectively in non real-time. However these methods are quite complex and need high computational overhead when it comes to hard real-time analysis within the context of the process dynamics.

The focus of this research work is to develop a method and system towards building an object-centric environment monitoring using collaborative efforts of both wireless sensor networks and artificial neural networks for spatial thermal mapping. Thermal tracking of an object placed anywhere within a predefined space is one of the main objectives here. Sensing data is gathered from restricted sensing points and used for training the Neural Network on the spatial distribution of the temperature at a given time. The solution is based on the development of a generic module that could be used as a basic building block for larger spaces. The Artificial Neural Networks (*ANNs*) perform dynamic learning using the data it collects from the various sensing points within the specific subspace module. The ANN could then be used to facilitate mapping of any other point in the related sub-space. The distribution of the sensors (nodes placement strategy for better coverage) is used as a parameter for evaluating the ability to predict the temperature at any point within the space.

This research work exploits the neuro Wireless Sensor Network (*nWSN*) architecture in steady-state and transient environments. A conceptual model has been designed and built in a simulation environment and also experiments conducted using a test-bed. A Shepard's algorithm with modified Euclidian distance is used for comparison with an adaptive neural network solution. An algorithm is developed to divide the overall space into subspaces covered by clusters of neighbouring sensing nodes to identify the thermal profiles. Using this approach, a buffering and Query based *nWSN* Data Processing (*QnDP*) algorithm is proposed to fulfil the data synchronization. A case study on the meat plants cool storage has been undertaken to demonstrate the best layout and location identification of the sensing nodes that can be attached to the carcasses to record thermal behaviour.

This research work assessed the viability of using *nWSN* architecture. It found that the Mean Absolute Error (*MAE*) at the infrastructural nodes has a variation of less than 0.5°C. The resulting MAE is effective when *nWSN* can be capable of generating similar applications of predictions.



# Chapter 1

## 1 Introduction

This chapter discusses the introduction to wireless sensor networks, artificial neural networks and thermal mapping. The aim of this research work is to map the thermal profile of a given specified space using both sensor networks and neural networks. Hence focus is given to each elemental background and further the motivation of the work is addressed. The research objectives and the detailed approach are described in this section. Later on the contribution of the overall work is discussed along with the research publications. The structure of this thesis has been developed in such a manner to enable the original hypothesis to be developed and exploited in a clear and concise format.

### 1.1 Introduction to Wireless Sensor Networks

In a WSN, numerous tiny, battery-powered computing devices are scattered throughout a physical environment. Each device is capable of monitoring, sensing, and displaying actuating information. Sensing may include the collection of values for temperature, humidity, vibration or other data. An actuating device may cause a LED to blink, turn on lights, change colours on a display, display textual information, and trigger any other action that prompts a response or informs an operator. WSNs are used in commercial, industrial, environmental, and healthcare applications to monitor data that would be difficult or expensive to capture using wired sensors.

Sensor networks can provide detailed coverage due to the small size of the nodes and therefore the large number that can be used within the required environment. However the small size can also mean that the node has a limited power supply. Due to this limitation, the sensor nodes will therefore have limited computational power, sensing capability and memory size. Sensors that can't communicate directly with the base station can pass their data to nearby sensors, in a multi-hop fashion, until the data reaches its base station. A Combination of local sensor

networks, together with regional and globally distributed sensor systems, permits a seamless study of different variables over very different spatial scales.

A variety of applications have been presented in the literature for WSN. These include ecological habit monitoring, smart spaces and geological monitoring. The deployment of embedded devices in recent decades has been widespread with increasing computational capabilities. Wireless sensor nodes [1-5] are very powerful and becoming popular to provide measurement of specific physical attributes. WSN nodes can communicate with neighbouring sensor nodes to exchange information with each other.

Multiple sensor nodes together constitute a wireless sensor network; these can be applicable to large scale environmental monitoring [6, 7], building management, industrial and transportation systems. The wireless sensor nodes can be deployed across the area of interest and where they can measure the relevant data. The sensor nodes transmit data to a central station where it can be further processed to enable the system to make certain decisions about environmental or system management. The current trend is moving towards incorporating higher levels of more complex, computational capabilities into the sensor node itself and to communicate with other nodes only when required.

In many applications hundreds of sensors are used in order to facilitate data environment for data collection. Estimating the temperature at any arbitrary position, where there are possibly no sensor nodes is a challenging task. Most of the WSN applications require monitoring of live streaming sensor data; hence the data processing must be performed in real time. Also data processing algorithms are time consuming and may not be able to cope with real time constraints. WSNs provide a better way of interacting with physical environments, hence these have been the focus of many research programmes over the last few years.

## **1.2 Wireless Sensor Networks: Target Applications**

Sensor networks can be useful in a variety of domains including cool stores, greenhouses, warehouses, cargo containers, building monitoring, climate monitoring, logistics and several industrial applications. Temperature prediction constitutes a crucial issue for different applications. There are many applications where the temperature distribution in a given space has to be assessed. Monitoring of temperature is of extreme importance in the food and agricultural industry. Many

of the food and agricultural products are sensitive to the temperatures at which they are stored and transported. For example, the storage temperature of kiwi fruits is critical to ensure the quality of the fruit is not compromised. Therefore it is important that a producer is aware when temperature falls outside the required range to enable corrective action to be immediately taken.

In another aspect, frost damage is a significant concern for fruit growers, where bud formation and flowering at the start of the growing season occurs. Unseasonably cold temperatures results in these flowers being killed and therefore, a reduction in fruit harvests. Hence temperature monitoring of the production system is vital. Of course, manual temperature monitoring of food products is possible but very time consuming. Furthermore, it is often not possible to measure multiple points in the production environment.

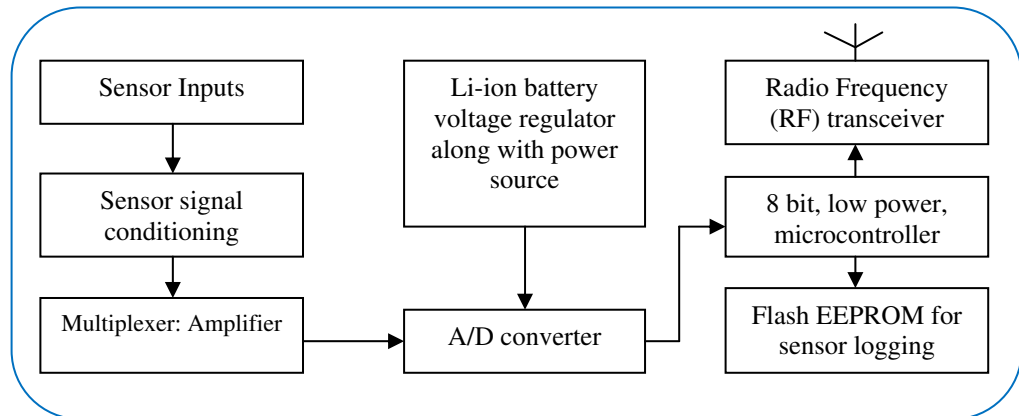
As a further example, in multiple storage containers, an incorrect temperature level may exist for an extended period before it is discovered. Thus an automatic thermal mapping solution is needed. The temperature points from the given system must be reliably gathered over a period of time to identify the zones where the thermal condition is beyond the designated values. The resulting data can be mapped to identify the problem areas within the system. There is a need for research and several key issues must be resolved to allow application of these in practice.

WSNs can facilitate the acquisition of physical attributes like temperature from a multitude of commercial applications. Within the cold chain of any perishable food commodity it is necessary to have information on and control of storage temperature, atmospheric temperatures, rate of cooling and length of storage to ensure cold chain integrity from harvest to consumption. It has been estimated that the fruit loss in Europe alone due to temperature abuse can cost \$NZ6 million, excluding costs associated with reworking product [16]. The meat industry is another primary industry where temperature of the carcass or meat primal must be monitored and controlled to maintain the quality of the meat.

### 1.2.1 *WSN Background*

A wireless sensor network consists of transceivers, sensors, microcontrollers and power sources. The current technologies have led to advancements in developing low priced, low powered and multi-functional sensor nodes, as shown in Figure 1.1. Sensor nodes are capable of environmental sensing along with data processing. The

interconnection between the external sources and the wireless sensor networks is used to communicate and exchange data.



*Figure 1.1 Wireless sensing node*

WSNs are used in several applications that include environmental monitoring, acquisition of data, buildings monitoring, security and safety supervision [8]. The sensor node communicates with a gateway unit which further transfers data through LAN, WLAN or WSN. These devices facilitate exchanging and monitoring data.

WSNs use the radio transmission medium provided by the Industrial Scientific and Medical (*ISM*) bands. Several studies have been conducted using these devices and the findings have been utilized by the research community to enable the uptake of sensor readings in WSNs using different protocols such as Bluetooth, Wi-Fi and Zigbee [11, 12, 13, 14]. The ISM bands have huge spectrum allocation and they are available on license free, such as the 2.4 GHz band to operate globally [8, 9, 10].

Multi-hop communications have greater advantages than traditional single hop communications in WSN since they consume less power [8, 9]. Zigbee and Bluetooth are the latest multi-hop communication protocols available and have become very popular.

The IEEE 802.15.4 standard is a physical radio specification that provides low data rate connectivity among relatively simple devices that consume minimal power and which typically connect over short distances. This communication standard is ideal for monitoring, tracking and controlling industrial and home applications [15].

Radio signals have the capability to penetrate through wall and glass, due to their lower frequency and longer transmission range.



### 1.2.2 WSN research challenges

Due to the sensors limited capabilities, there are a lot of design issues that must be addressed to achieve an effective and efficient operation of WSN.

- 1) **Autonomy:** Sensor nodes are commonly deployed in places where cable access is either not possible or expensive. Sensor nodes are normally operated on a battery power and recharging may not be possible at all locations. Therefore, energy usage is extremely valuable in sensor nodes, building algorithms for minimizing the load on nodes is crucial when designing sensor networks. Another consequence of autonomy is the need for sensor nodes to organize themselves by learning and adapting to a changing environment.
- 2) **Location Identification:** When an application requires the location of the sensor node, it is important to embed an algorithm that uses a location discovery protocols. Most of the tracking applications need to have a specified location. There are solutions available that use the GPS based technology, but the cost and energy consumption are high with this system. Recent research reveals methods to compute the location of the nodes by utilizing very minimal information. But further work is required to develop this to a point where it can be used in any system.
- 3) **Limited Computational Power:** There is a lot of advancement in integrated circuit technology and its improved processing capacity. The capability available for data processing, data communication and memory of sensor nodes has increased markedly over recent times. Due to the limited energy of sensor nodes, it is advisable to minimize the computational times. Therefore, any information processing within the sensor nodes has to take into account the corresponding limitations. It is important to take proper measures while allocating the memory buffers and assigning the data types of the variable assignment.
- 4) **Complex Dynamics:** The dynamics of the measured environment data can be complicated, for example if the current value of the dynamics depends on the long term historical value, or if the dynamic model at one location is related to the one at another location. Therefore, in some cases, the dynamics of the measured data is a complex system with both temporal and spatial characteristics.

### 1.3 Artificial Neural Networks Background

The theoretical background related to ANN is discussed here but it will be limited to the topics directly related to the solutions given in this thesis. In theory, there are many popular Neural Network (NN) architectures available, particularly those used in offline tasks and these are complicated combinations of diverse neural structures [18, 19, 20] along with many statistical models [21, 22, 23, 24]. Due to the limited processing capacity of the sensor nodes, it may not be important to discuss the hybrid structures and evolutionary neural models [25, 26, 27]. A gradient calculation of NN architecture is considered since it is the main factor of the learning model.

The computation within the NN was inspired by the functionality of a biological network, namely the human brain. Similar to a biological neuron, the artificial neuron has the structure as shown in Figure 1.2, where the neuron collects and accumulates the data  $\vec{X} = [x_1, \dots, x_i]^T$  from outside, then fires an output after applying a summation over a nonlinear function  $g(x)$ , called the activation function. This can be formulated as shown in the equation 1.1, where  $b$  is the bias of the neuron, similar to the neuron's activation threshold.

$$y = g(\sum_i w_i x_i + b) \quad 1.1$$

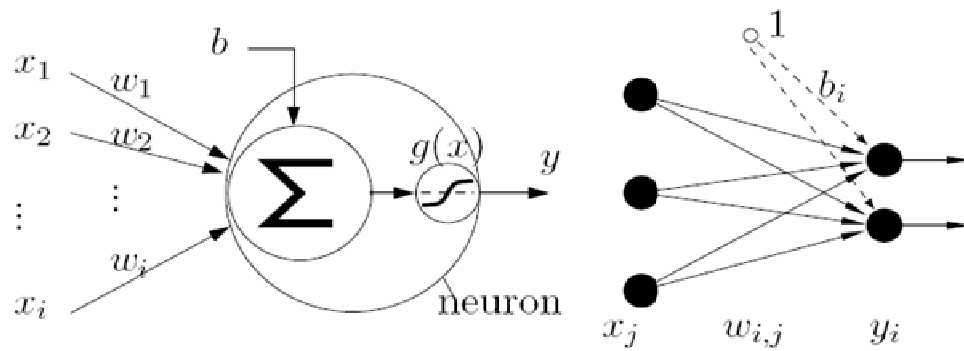


Figure 1.2 Activation of neurons and its layers

An ANN is a collection of many neurons, where in most cases the neurons activate synchronously and propagate their activation to another group of neurons. One of the important classical example of this is the feed forward operation between two

layers of neurons as shown in Figure 1.2, where black circles represent neurons, a hollow circle denotes the bias. The group contains neurons usually independent of each other. The information is propagated from one group to another in an ANN.

### 1.3.1 *On-line Learning Algorithms*

In this research work, we proposed an online learning algorithm that is suitable for the application domain. Hence it is important to review the available algorithms. Neural network models can be categorized into two main classes based on the presentation of the training data and training methods. Those are the batch training methods and the online training methods. Batch training is defined as each iteration being trained with a batch of data at one time, such that the  $\Delta w$  of the model parameter  $w$  is derived from the data set  $\{x_1, \dots, x_n\}$ :

$$w_t = w_{t-1} + \Delta w \mid \{w_{t-1}, x_1, \dots, x_n\} \quad 1.2$$

Online learning also adjusts the model parameters in each iteration with the increment  $\Delta w$ , which depends on the input data. In recurrent neural network, there is a memory vector  $S_t$ , which holds the history of input data; the parameters' update is given by equation 1.3.

$$w_t = w_{t-1} + \Delta w \mid \{w_{t-1}, S_t, x_n\} \quad 1.3$$

Batch training methods have the ability to provide sufficient results using finite training set in each iteration. However this method requires a large amount of memory and high computational power. Hence, the batch training method is not suitable for sensor network applications. On the other hand, online training is a simplified method where it takes only a single data set into account in each iteration. The evidence also clarified that the online training can restore the trajectory of the batch learning model [96].

While looking at the difference in learning algorithms, error functions of neural network applications can normally take the form given by equation 1.4.

$$e[w] = \frac{1}{2} \sum_{i=1}^d (\hat{y}_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^d (\hat{y}_i - [f(w)]_i)^2 \quad 1.4$$

where the evaluation error  $e$  measuring the Euclidean-like distance between output and target depends on parameters  $w$ , symbol  $\hat{y}$  stands for the corresponding target. The network output  $y$  can be expressed as a function of  $w$  such that  $y = f(w)$ , and the coefficient  $\frac{1}{2}$  is used for convenience of calculation.

With the definition of residual  $\delta$ :

$$\delta_i = \hat{y}_i - y_i = \hat{y}_i - [f(w)]_i, \forall_i \quad 1.5$$

the associated gradient calculation is given by equation 1.6.

$$\frac{\partial E[w]}{\partial w} = - \sum_{i=1}^d \delta_i \frac{\partial y_i}{\partial w} = - \sum_{i=1}^d \delta_i \frac{\partial [f(w)]_i}{\partial w} \quad 1.6$$

### 1.3.2 Gradient Descent Learning Algorithm

The gradient descent learning algorithm adjusts its parameters by following the error gradient. As defined in the equation 1.4, at a given task with an evaluation error  $E$  and also a gradient based on the equation 1.6, the adjustment step parameter is given by equation 1.7 and the parameters are updated using the equation 1.8.

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_{i=1}^d \delta_i \frac{\partial y_i}{\partial w_j} \quad 1.7$$

$$w_j = w_j + \Delta w_j \quad 1.8$$

where  $\eta \in \mathbb{R}$  is the predefined learning rate, which is usually very small,  $j$  is the iteration index and  $\delta_i$  is the corresponding residual for index  $i$ .

### 1.3.3 *Back-Propagation algorithm*

Within the multi-layer perceptron (*MLP*), the back-propagation algorithm is a further advancement to the gradient descent learning algorithm. This algorithm has been implemented in several applications and has proven accurate [18, 19].

The back-propagation algorithm cycles through two distinct passes, a forward pass followed by a backward pass through the layers of the network. The algorithm alternates between these passes several times as it scans the training data. Typically, the training data has to be scanned several times before the networks ‘learns’, thereby generating an accurate prediction with minimal error.

### 1.3.4 *Multiple local optima and epochs*

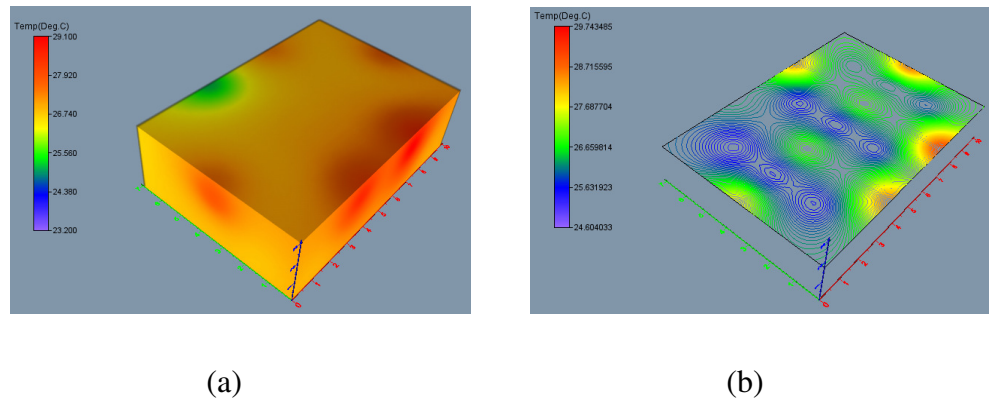
The back-propagation algorithm is a version of the steepest descent optimization method applied to the problem of finding the weights that minimize the error function of the network output. Due to the complexity of the function and the large numbers of weights that are being ‘trained’ as the network ‘learns’, there is no assurance that the back propagation algorithm (and indeed any practical algorithm) will find the optimum weights that minimize the error. The procedure can get stuck at a local minimum. In these circumstances, it is useful to randomize the order of presentation of the cases in a training set between different scans. It is possible to speed up the algorithm by batching, that is, updating the weights for several exemplars in a pass. However, at least the extreme case of using the entire training data set on each update has been found to get stuck frequently at poor local minima. A single scan of all cases in the training data is called an epoch. Most applications of feed-forward networks and back propagation require several epochs to minimize the error. A number of modifications have been proposed to reduce the epochs needed to train a neural net. One commonly employed approach is to incorporate a momentum term that injects some inertia in the weight adjustment on the backward pass. This is done by adding a term to the expression for weight adjustment for a connection that is a fraction of the previous weight adjustment for that connection. This fraction is called the momentum control parameter. High values of the momentum parameter will force successive weight adjustments to similar directions.

Another idea is to vary the adjustment parameter  $\delta$  so that it decreases as the number of epochs increases. Intuitively this is useful because it avoids over fitting which is more likely to occur at later epochs than earlier ones.

## 1.4 Thermal Mapping Background

Thermal mapping is described as predicting an unknown temperature at a given point and it can further generate a profile based on multiple points within the given specified space. Figure 1.3 shows a mapping of a given space. The thermal mapping provides an easy identification of the hotspots in the space.

Temperature prediction constitutes a crucial issue for different applications. There are many applications where the temperature distribution in a given space has to be assessed and several techniques or algorithms are available to predict unknown temperature data by using known data. There are several methodologies applied in various studies for thermal mapping. The majority of the techniques used are interpolation methods and discretization methods.



*Figure 1.3 Thermal mapping - (a) Thermal map (b) Contour map*

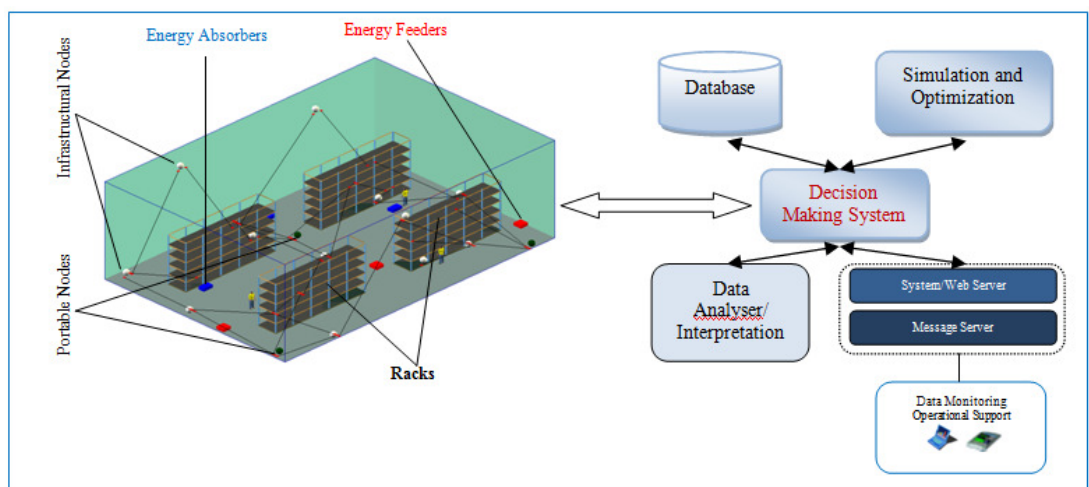
A few conventional methods including CFD and FEM are also available to examine the environmental variables in a given space. In these studies multiple parameters can be varied in three dimensional calculations and their influence on distributions of temperature analysed. However, these studies are quite complex and need high computational overhead when it comes to real time analysis. Also data processing algorithms are time consuming and may not be able to cope with real time

constraints. Most of the WSN applications require monitoring of live streaming sensor data, hence the data processing must be performed in real time.

## 1.5 Motivation

Quality of product is an important aspect in many commercial organizations where storage and shipment practices are an integral part of the operation. Temperature and humidity are the main parameters that influence product quality in agricultural and pharmaceutical industries. The applications for the cold chain integrity of meat products require special attention to work successfully. The focus is to identify the methodologies required to monitor and map the temperatures for a given infrastructure by using the WSN. In the meat industry the temperature variation in cool store has a significant effect on tenderness, colour and microbial status of the meat; therefore thermal mapping during the chilling process and shipment of the product is vital.

Most of the consumable products such as fruits, vegetables or meat require controlled temperatures while transporting from one place to the other. In order to avoid deterioration and market loss, thermal monitoring is required and the products should be maintained at rated temperatures. Short interruptions in the control of the cold chain may result in immediate deterioration of product quality. Quality control, monitoring of goods transportation and delivery services is an increasing concern for producers, suppliers and consumers.



*Figure 1.4 Concept Diagram of Object-Centric Environment*

Temperature is the most important factor for extending shelf life. It is essential to ensure that temperature before and during transportation is stable. Reports indicate that a gradient of 5°C or more and a deviation of only a few degrees can lead to spoiled goods and thousands of dollars in damages [17]. Another reason for deterioration is the water loss that reduces the marketability of fresh meat, fruits and vegetables.

Figure 1.4 gives a conceptual diagram of the object-centric environment. Practical motivation for this problem derives from current technological changes and reducing the size of sensor nodes, which will facilitate the acquisition of data on physical attributes (temperature, humidity and gas, etc.) under circumstances. There are several different sensor nodes available. These nodes differ from each other in their modalities, monitoring range, detection capabilities and cost. The sensor nodes can be classified into either fully functional or reduced functional versions based on their capacity in terms of processor, memory and battery costs. The philosophy of this work is to utilize fully functional ones to build the infrastructural nodes and reduced functional ones to use as portable nodes.

The WSN is considered as the future technology of Radio Frequency Identification (*RFID*) tag evolution that advances how devices communicate with each other. These features contribute towards the development of an object-centric environment for thermal mapping, where the objects may vary based on the applications. Hence algorithmic techniques are needed to use the sensor nodes effectively. The infrastructural nodes may be expensive nodes and are deployed at some specific locations in such a way that they can contribute their services within the network of portable nodes, which may or may not move in the given space. This work considered the problem of placing sensors in a space for several applications. Examples are in buildings, warehouses, greenhouses, cool stores and containers, etc to map the temperature of the space. With the current growing infrastructural needs, we require decision support tools which can assist in planning locations for the nodes. The contribution of this research is related to the mapping of the thermal space and optimal placement configuration of the infrastructural nodes as well as the ubiquitous management of the system that focuses at sensor, system and mobile levels.

This thesis discusses the development of a cool store thermal mapping system based on the nWSN.



## **1.6 Problem Statement and Approach**

### **1.6.1 *Research Objectives***

The focus of this research is to develop a method and system towards building an object-centric environment using WSN for spatial environmental mapping. Thermal tracking of an object within a predefined space is one of the main objectives here. This research will look at using soft computing for spatial thermal analysis based on gathered data from restricted sensing points. The space is divided into two or more clusters of neighbouring nodes to communicate between each other in multiple regions. Each cluster having embedded therein an artificial neural network, which takes as an input data from other sensor nodes that monitor the environment surrounding them and adaptively maps the dynamic environment for the associated region. The distribution of the sensors (nodes placement strategy for better coverage) will be used as a parameter for evaluating the ability to predict the temperature at any point within the space.

In this study the temperature mapping is analysed by using ANN. In order to test the system, a conceptual model is constructed based on the nWSN architecture. The objects involved in the modeling, such as the infrastructural nodes have been designed and a few assumptions made while building the scenarios.

## **1.7 Contributions**

This thesis has initially discussed the state of the art of WSNs with a focus on applications in the field of thermal monitoring of indoor spaces. The literature review also revealed that the ANN in the field of WSN area is now growing. The research focus of this work, which is on the thermal mapping using ANNs and WSNs has not been widely covered in the literature. Therefore our contribution is to explore the possibilities using these techniques.

Specifically, this thesis covers the following areas:

- 1) The requirement of real-time monitoring in many applications including meat industry.

- 2) The importance of thermal mapping and the generation of a temperature gradient profile in real time.
- 3) ANNs and their applications are explored and further addressed the areas mentioned above.
- 4) A concept towards building an Object Centric Thermal Mapping environment based on the use of WSNs has been developed; here the sensor network is represented through infrastructural and portable sensor nodes of any functional space. The infrastructural sensor nodes facilitate initial values for the calculation of the temperature at a given location within the space. These results showed that the neural network for temperature mapping is feasible. Furthermore, a nWSN architecture has been developed to train the neural network continuously. An algorithm is developed to divide the overall space into subspaces covered by clusters of neighbouring sensing nodes to identify the thermal profiles. As part of this, a buffering and QnDP algorithm is proposed to fulfil the data synchronization requirement.
- 5) A test bed has been constructed at SeNSe lab using Atmel's RZUSBSTICK as a gateway and AVRRAVEN as *moten* to conduct the experiments. This work enhances the nWSN architecture that has been developed for spatial thermal mapping with query system for time synchronization and relevant aggregation functions at the sensor level.
- 6) A case study of a meat plants cool storage has been undertaken to demonstrate the best layout and location identification of the sensing nodes that can be attached onto the carcasses to provide accurate thermal data.

## **1.8 Publications**

During this study, the following International peer reviewed publications and conference proceedings have been completed.

- 1) Naresh Yamani and Adnan Al-Anbuky, “Query Based nWSN Data Processing for Spatial Thermal Mapping”, *The Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2011)*, Adelaide, Australia, December 6-9, 2011.
- 2) Naresh Yamani and Adnan Al-Anbuky, “neuro Wireless Sensor Network Architecture: Cool Stores Dynamic Thermal Mapping”, *IEEE Sensors Applications Symposium*, San Antonio, TX, USA, February 22-24, 2011.
- 3) Naresh Yamani and Adnan Al-Anbuky, “Object-Centric Thermal Mapping (OCT MAP): A Wireless Sensor Network Perspective”, *In the 8th Annual IEEE Conference on Sensors*, Christchurch, New Zealand, October 25-28, 2009.
- 4) Naresh Yamani, Adnan Al-Anbuky and Amoakoh Gyasi-Agyei, “Portable Object Thermal Awareness: Modelling Intelligent Sensor Network for Cool Store Applications,” *In the Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dunedin, New Zealand, December 1-4, pp. 218-224, IEEE Computer Society, 2008.

## **1.9 Structure of the Thesis**

This thesis is divided into three parts. Part 1 (Chapter 1, Chapter 2 and Chapter 3) covers the Introduction, Literature review and Environment for model and test-bed development. Part 2 (Chapter 4 and Chapter 5) focuses on the conceptual design of the spatial thermal mapping, nWSN simulation and validation. Part 3 (Chapter 6, Chapter 7) discusses the nWSN implementation factors and a cool storage in a meat plant as a case study. Future research areas are discussed in chapter 8.

*Chapter 1:* The current chapter explains the WSN background and target applications. A little background on ANN and thermal mapping is also presented. The research motivation and objectives of this thesis is described as well as the contributions and solutions in general.

*Chapter 2:* Presents the literature review and the current research in related topics, mainly in three categories including artificial neural networks, wireless sensor networks and thermal mapping. The focus of this research is highlighted where the ANN, WSN and thermal mapping are intersected.

*Chapter 3:* This chapter discusses the modelling and development tools used for this research. The model building to mimic the real time system is important and these tools help to setup the environment to conduct the experiments.

*Chapter 4:* This chapter covers the philosophy of the neuro WSN and its structural design to suit the problem domain. The thermal mapping methodologies are compared and the requirements are discussed, including the computation and coverage aspects, subspace and its overlap, nodes message interaction model and query based nWSN data processing, nodes minimization algorithms.

*Chapter 5:* The nWSN simulation and validation is covered in this chapter. This includes generation of thermal profile for the modeling along with the discussion of the assumptions. Various scenarios are constructed to validate the concept and compare the ANN approach. The results from the experimental test-bed are also given in this section.

*Chapter 6:* The implementation of the nWSN architecture requires various components and these are derived and explained here. This includes the query based nWSN, time synchronization and its implementation, neural net cluster dynamic grouping and nodes minimization approach.

*Chapter 7:* A case study on the meat plants cool storage is discussed to discover the best layout and location identification of the portable nodes that can be attached on to the carcasses by keeping thermal accuracy in mind.

*Chapter 8:* Finally, conclusions and future work directions are discussed in this chapter.

# Chapter 2

## 2 Literature Analysis

The aim of this chapter is to review the existing literature that covers the areas related to thermal mapping, ANN and WSN for monitoring and traceability capability. This review also introduces the concept of cool room monitoring and tracing systems. The interdisciplinary work among these areas is also addressed and further section discusses the research focus of the current thesis.

### 2.1 Cold Room Monitoring and Tracing Systems

Tracking is defined as gathering the information related to the current location of products whereas monitoring refers to the ongoing assessment of the progress of transport by means of continuous or repeated measurement and evaluation [30]. In recent times, focus has been on the development of intelligent tracking systems. These systems have been developed with or without human intervention, and with wireless based systems. There are methodologies implemented using wired and wireless communications for many applications [28]. A number of supply chain and tracking systems have been developed and among those systems most of them are meant to be for non-intermodal transportation systems [29].

The refrigerated containers are most widely used for perishable products like fruits and meat transportation. Improper functioning of the cooling system can result in significant production loss and therefore monetary loss to suppliers. Data loggers are the most common devices that are used by several companies to trace and track the products temperature profiles. But the disadvantage of data loggers is that it will not provide information about the whole thermal space mapping of the given volume and these data loggers are operated offline only.

The tracing systems have to operate independently of the refrigeration systems to ensure that if anything goes wrong with the refrigeration system, the tracing system has to remain operational. To this end, there are systems where wired connections and sensors have been used to improve the overall monitoring [31].

## 2.2 Mapping and Thermal Mapping

The concepts of general three dimensional mapping have been studied over the past few years. The proposed models considered numerical, analytical, interpolation methods, discretization methods and Artificial Intelligence (*AI*) as well as their hybrids. Each model has its own limitations, depending on the application.

The process of general mapping involves solving the image transformation relative to the vector data by taking into consideration its attribute values. The survey of traditional texture mapping is described in [32]. Geostatistical applications are investigated in [33], considering the problem of spatial sampling and interpolation methods. Geographical Information Systems (*GIS*) are also considered to be one of the mapping tools of the digital age.

The main purpose of thermal mapping is to ensure that all areas of the process achieve the required temperature levels. Many researchers have attempted to predict indoor/outdoor temperatures and several models have been reported. An accurate numerical model of coupled heat transfer in buildings has been developed [34]. Data Based Mechanistic models (*DBM*) are also used for real time monitoring and online adaptive control of three dimensional distributions in both an individual biological product and in a given movement [35]. Teodosiu et al. [36] employed a computational fluid dynamics technique and a modified  $k-\epsilon$  turbulence model to predict indoor air moisture and its transport in a mechanically ventilated test room to estimate the level of thermal comfort. Among these models, more detailed and complex ones are Navier-Stokes equations which describe the flow of fluids for air flow, temperature and contamination distributions. Most of these problems are solved using discretization methods.

These numerical methods, called physical models, can be used to simulate the air temperature distributions. The main drawbacks of these models are extensive computations, which lead to time consuming simulations. Hence for a medium size building, it may take days to complete indoor temperature simulation in a modern personal computer [37]. Most of the conventional modeling techniques run offline, as it is quite difficult to run in real time. This is due to its computational times and thus requires high end processors to simulate the model. Further difficulties develop if the model includes environmental dynamics including energy absorbers and energy feeders. Several computational fluid dynamics tools are available to examine the environmental variables in a given space with different boundary

conditions [45] where multiple parameters can be given that can be varied in 3D calculations and their influence on distributions of temperature analysed. However these studies are quite complex and require high computational times when it comes to real time analysis.

A variety of applications require temperature data distributions of a specified area. Predicting 3D spatial temperature uniformity from inlet temperature distribution in food storage systems has been studied [46]. Control of food storage environment is usually done using a limited number of temperature sensors in the facility. The design, deployment, and output of a large scale WSN in agriculture are described in paper [47]. Sensor data were analysed in a vineyard to monitor temperatures at various locations as temperature is considered to be one of the primary variables affecting the growth of grapes. The most significant findings were that the areas with highest temperature varied from day to day. In addition the heat summation data that is generated from these studies can provide grape growers with a better awareness of potential variability in fruit maturity. So far several researchers have analysed various methodologies and implemented algorithms for temperature mapping specific and relevant to the application area.

### **2.3 Artificial Neural Networks: A Review**

Soft computing is a multidisciplinary field that was proposed by Dr. Lotfi Zadeh, whose goal was to construct new generation Artificial Intelligence (AI), known as computational intelligence. The idea of Soft Computing was initiated in 1981 when Dr. Zadeh published his first paper on soft data analysis (see[52]). The main goal of soft computing is to develop intelligent machines and to solve nonlinear and mathematically unmodelled system problems.

An artificial system can emulate a simplified version of a neural computational system. The ANN is an example of such an artificial neural system [53]. ANNs have often been used as an alternative to the techniques of standard nonlinear regression and cluster analysis to carry out statistical analysis and data modeling [54]. The main characteristic of ANNs is their ability to learn. The learning process is achieved by adjusting the weights of the interconnections according to some applied learning algorithms. Therefore, the basic attributes of ANNs can be classified into architectural attributes and neurodynamic attributes [55]. The architectural attributes define the network structure, i.e., number and topology of

neurons and their interconnectivity. The neurodynamic attributes define the functionality of the ANN.

### 2.3.1 *Soft computing in physical fields*

The knowledge of temperature variation is used for the prediction of energy consumption in solar buildings. To estimate the daily temperature variation a number of different methods have been used and artificial neural networks have also provided a method for prediction in many applications. Neural networks were successfully used to model nonlinear systems and have been applied to greenhouse environment modeling as they have a strong ability for nonlinear function mapping [57, 58]. In energy applications, the information generated by the wide variety of experiments will be processed with the aid of models based on artificial neural nets in order to assess its importance. The example of this paradigm is an experiment being carried out in the particle accelerator Large Hadron Collider (*LHC*) in the European Nuclear Research Centre. The advances in artificial neural networks, methodological development and applications were studied, among others, in [59], which described various ANN architectures and training algorithms. Support Vector Machines (*SVM*) have also been used in parallel with ANN as a set of supervised generalized linear classifiers for atmospheric temperature prediction [60]. These methods have performed centrally, which means the data processing has not been distributed among the nodes or processors for computations.

Distributed computational techniques have also developed in the field of ANN. A distributed computing architecture and environment based on grid technologies has been developed for rapidly and accurately dealing with the fitting of neural network for flood peak forecasting [61]. In another study, a parallel implementation of feed forward neural network has been developed using C# and message passing interface on .NET platform [62]. A toolkit [63] offered a XML based framework for implementing distributed ANNs; this framework is implemented to study the flexibility and scalability issues when multiple systems are connected to obtain a power beyond the power of human biological neural networks.

### 2.3.2 *ANNs applied to thermal mapping*

Artificial intelligence methods for thermal mapping are considered to be a better approach for real time mapping. Unlike the above-mentioned physical models,



ANN entirely depend on experimental data, which can be made adaptive and offer a much faster computation. A neural network is a powerful data modeling tool that is able to capture and represent complex input/output relationships. Various complex problems have been solved using ANNs, like weather prediction [38] and heat transfer prediction [39]. Neural networks have a good ability for pattern recognition and classification of data with multiple attributes. They have been widely used in estimating permeability from well logging information [40], and pixel by pixel classification of satellite images for making surficial geological maps [41, 42]. Several applications of neural networks for spatial estimation and interpolation of geological data have also been reported [43, 44], these are called interpolation methods using a neural network.

## **2.4 Wireless Sensor Networks: A Review**

A WSN permits the measurement of variables distributed over a network. WSNs provides an unprecedented way of interacting with physical environments [48] something which has become a hot topic for research over the last few years. In many sensor networks applications, sensor nodes collect correlated measurements of physical fields [49]. The NNARX system is proposed [50] for modelling the internal greenhouse temperature as a function of outside air temperature and humidity, global solar radiation and sky cloudiness. The model showed a good performance without the need to frequently retune the parameters with a good fitness. The temperature gradient analysis is vital when transporting in containers or trucks. There was a proposal published [51], where a WSN was employed in refrigerated vehicles. An alarm is triggered when the temperature gradient falls beyond the limits, to avoid the deterioration of the products.

As one example, city buses are equipped with sensors for atmospheric temperature and pollution measurements [64]. Most of the Ecological Research stations are increasingly using wireless sensor systems. WSNs are also of great interest for studies on the number, movement and behaviour of wild animals. In self-managed WSNs [65], nodes are deployed, they wake up, perform a self-test, find out their localization and monitor their energy levels. The proposed management solution results showed that it can improve the performance of the various continuous WSN configurations and give the observer relevant information. How a sensor network detects small changes in a smart environment has been studied [66]. Small changes

in an office, such as temperature or human movements can be detected. Another study explored the applications and challenges for underwater sensor networks [67]. They have focused mainly on the potential applications of using sensor network nodes in offshore oil fields for seismic monitoring, equipment monitoring and underwater robotics.

Sensor networks are already used for climate monitoring to detect rainfall, water level and weather conditions. Several works have been published in this field where the sensors supply information to a centralized database system [68]. Real time surveillance systems are also proposed [69], where the WSN measures temperature and humidity, and smoke for fire detection. In many agricultural production systems the real-time measurements can provide important information which can be used as a basis for system management. WSN are playing a major role in precision agriculture and irrigation [70]. Wireless sensor technologies eliminate the difficulties when a network is deployed with wired sensors across the field. Greenhouse monitoring and control is another field where WSN can be rapidly implemented. In 2003, the first application of WSN in greenhouse environment was reported [71]. Later, another proposal [72] was published for greenhouse control and monitoring system using Zigbee.

#### 2.4.1 *ANNs applied to wireless sensor works*

WSNs supported by ANNs offer promising solutions for numerous applications. With the growing requirement to closely monitor and manage systems across many different areas, comes the requirement to provide accurate and robust decision support methods. However, one area that has received little attention is the application of ANNs within WSNs.

In a study given by Flouri et al. [91], an SVM classifier is applied in a distributed fashion in WSN. A consensus mechanism and a gossip algorithm are used to train the network. The model was described and the information communicated to one-hop neighbours in order to update the estimation at each iteration.

In another study a Machine Learning (ML) approach appears to be a powerful tool for fast predictive modelling of temperatures [92]. It is identified that the mapping of the temperature inversion phenomenon is essential to classify areas where frost can occur. The combination of ANN with K-NN has applied in other applications outside the thermal mapping domain. ML methods can handle the data-driven air

temperature prediction maps to precise and reliable modelling of mean air temperatures. There are two different aggregation architectures presented, in which wavelets for initial data processing and ANN for categorization of the sensory inputs [93]. A greenhouse sensor network model is designed [94], to address conditional monitoring and facilitation for diagnostics and prognostics of various asset types inside the greenhouse that includes plants, machinery and others. These studies haven't fully focused on WSNs that use ANN architectures for real time prediction and mappings. There are not many studies published in the intersection of WSN, ANN and thermal mapping shown in Figure 2.1 and further gaps needs to be explored.

#### *2.4.2 Nodes placement and minimization*

The nodes minimization approach is required to identify the number of nodes required to map the thermal space. In the literature review only a small number of studies in this area could be found. An attempt to solve the sensor placement for diagnosing problems in plants has been made [84]. However, they have built a scenario to minimize the sensor nodes within a discrete number of possible positions rather than the continuous space. There have also been non-numerical approaches to sensor placements that show how to place visual sensors in a building for 3D mapping using a combinatorial optimization approach [85]. There are several integer programming formulations to sensor placements for contamination detection in water networks [86-88]. In another method [89], an initial estimate is made, via a sensitivity analysis of the set of potential sensor locations. Then the author seeks the minimum number of these sensors required to ensure accurate observations of the present state of the network. Other techniques exist in the literature for designing sensor networks around environmental obstacles such as walls or cliffs [90].

## **2.5 Data Collection and Query Processing in WSN**

The existing literature indicates that the data collection within the WSN community is identified among three major trends: Systems, Testbeds, and algorithmic research [73]. Systems research is based on the sensor network query processors like TinyDB and Cougar etc. This simplifies the user access at communication, routing

and node programming levels. The research using these systems is based on the number of algorithms which are embedded within the sensor nodes. The testbeds are other area where the distributed data is shared among different users. It can give a better focus on testing algorithms and their scalability issues. Current trend and evolving requirements have motivated the algorithmic research in the WSN. Our contribution to this research falls within this area. In WSN data processing, the nodes can keep on sending data to the base station. This functionality has been studied in other WSN applications [74], but the data are received asynchronously from all other sensor nodes by the base station. The asynchronous data need to be filtered in order to produce any valuable information at the front end among the nodes. If the application requires a real time synchronous data, then a query processing needs to be implemented. Based on the existing methods, algorithmic approach would provide an efficient way to work in a homogenous or heterogeneous network. In any typical WSN it might be a common scenario to have a sink, a few cluster heads and more end nodes. By using a clustering approach, it ensures only the cluster heads are actively involved in the transmission of queries. Most of the works till today are based on the systems approach that involves query processors like TinyDB. In an experiment conducted by Gehrke et al. [75] for a smart sensor query processing architecture using database technology, they focus on networks composed of homogenous collections of nodes. In other studies [76, 77], location-centric storage is envisioned for on demand data storage in networking environments. Based on their algorithms it has been identified that the stored data of any event only takes a small number of communication hops to query the sensor nodes. Another study on the interplay between mobile devices and static sensor nodes has been discussed [78]. The methodology supports and enables the heterogeneous design space. This work contains staged operations including query generation, query routing, query injection and query result routing to fulfil a two-layer network approach. Data collection and monitoring will be a tedious task for large networks of sensors. Selective query processing from a user interface would be widely accepted if it is on a web based management system. An implementation related to the wireless sensor gateway for efficient query management through World Wide Web is described [79]. This approach is more suitable for remote accessing and managing the network. The whole sensor network is simulated and is connected through a socket for communication. A dynamic data aggregation scheme [80] is introduced to elaborate more aggregation schemes. These schemes

allow nodes to process data collected from sensors and aggregate the data based on the interest messages. An adaptive holistic scheduler [81] is introduced to adapt the schedule to the runtime dynamics. It shows that the performance of query processing improves in various dynamic settings. Inter query is another area of interest [82], to compute algorithms and reduce data redundancy.

The data aggregation can be classified into two types, single query data aggregation and inter-query data aggregation. In multi-hopping networks, inter-query data aggregation algorithms can be very useful. In other study [83], a mechanism for spatial queries in WSN to detect dangers in disaster situations is discussed. These queries are created at more than one packet to acquire irregular contour data. Most of these query based algorithms have not much focussed heavily on data synchronization.

## **2.6 Research focus**

The earlier topics have described the previous work done in the area of WSN, ANN and thermal mapping. A thorough literature review has revealed that there are several studies in the areas of WSN, ANN and thermal mapping individually. However, few studies cover a combination of any two of these areas. It has been identified that the research on thermal mapping using neural networks and WSN area is not well developed and requires further attention. Figure 2.1 shows the area where the current research work is focused and also provides some of the key references.

Our research is to identify the methodologies to map the given space for thermal profile in WSN platforms using soft computing. Hence, the focus is given to the intersection of these three elements as shown in Figure 2.1. The literature review for each of these elements is already described in the earlier sections. Hence, we have focused on developing a generic concept where it can fit to an object-centric environment to map the thermal profile. These objects could be food items, plants or any industrial products where it is critical to monitor temperature. Towards this, we have examined the possible techniques that can be used in the WSN domain as the sensors nodes are very limited in computational resources. We have also realized the advantages of the online learning paradigm, which is suitable to drive the WSN in a real time environment for thermal mapping using ANN.

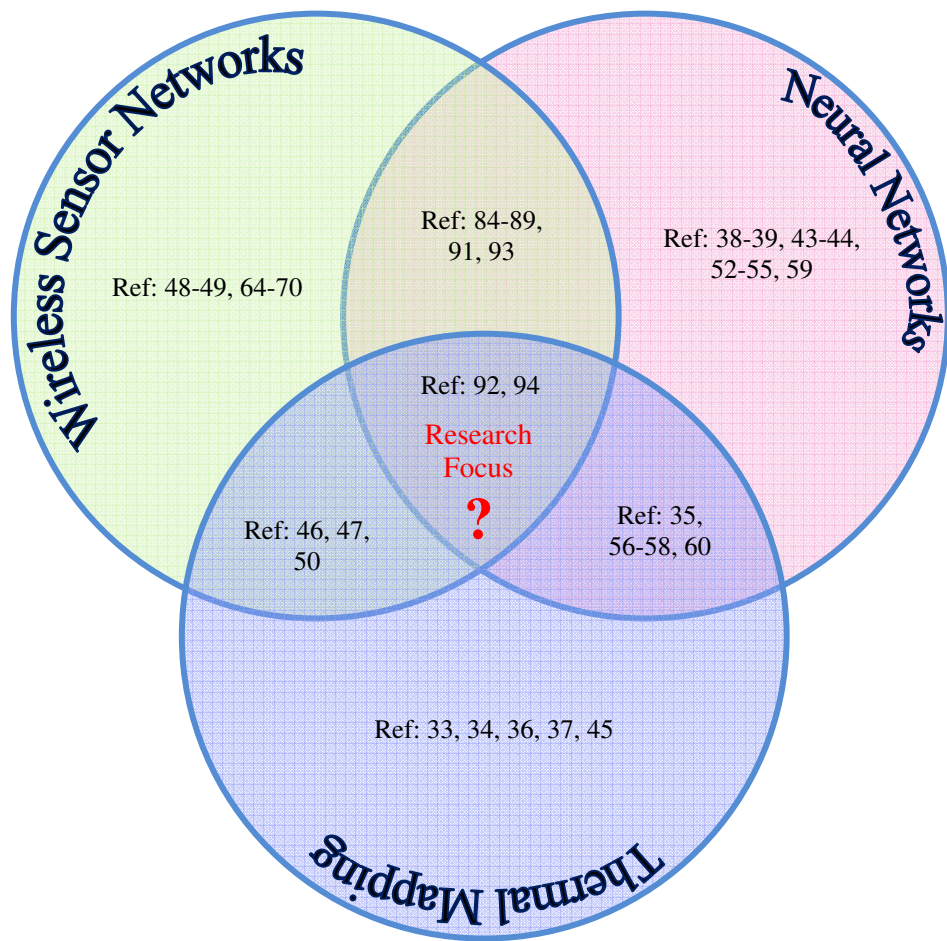


Figure 2.1 Research focus Intersection areas

# Chapter 3

## 3 Environment for nWSN Development

This chapter introduces the tools required for nWSN development. In order to develop a concept we have to work either to build a model in a software environment or develop a prototype system. Nowadays there are several commercial software packages available that can mimic the real systems, while a prototype may have limited flexibility to create various scenarios other than the simulation environment. This chapter focuses on the software environment for simulation, neural network and data visualization for thermal mapping and the hardware environment for sensor networks.

### 3.1 Introduction

Development tools are necessary to undertake the modeling of a concept before implementation. This can be done either in a simulation environment or by directly embedding the logical code in suitable hardware or test beds. Several simulation tools are available to work in WSN simulation modeling to mimic the real system. These include Mote Runner from International Business Machines Corporation (*IBM*), OPNET, Visual Sense, sQualnet etc. There are also tools available from research institutes, but these are not widely used. The commercial versions mentioned above are popular in the WSN research area. When it comes to the specific modeling requirements of this research, there is no such environment or tools available to simulate and visualize the results. This is due to the reason that most of the simulation environments don't support customization. In other terms, they are not open source tools. Hence, we chose to have more than one single environment to simulate and visualize the concept and to analyse the results.

The main software environments used in the research work are as follows

- 1) IBM's Mote Runner - WSN Operating System,
- 2) Flexsim – Simulation Software,

- 3) Peltarion Synapse Neural Net Software, and
- 4) Voxler – A three Dimensional Scientific Visualization Program.

## 3.2 Simulation Environments

### 3.2.1 IBM's Mote Runner

The IBM's Mote Runner runtime environment for WSN is currently under development at the Zurich Research Laboratory. Mote Runner is an operating system for the WSN environment. This also comes with simulation software to work offline to test the programs before loading into the sensor board. The hardware (RZUSBSTICK and AVRRAVEN) that has been used for the test bed discussed in Chapter 5 is from the Atmel Corporation. This hardware does support IBM's Mote Runner operating system. Hence, we have chosen this hardware to enable ease of the development in simulation and real time modes.

The key features of the simulation environment and its supporting hardware are shown in the Table 3.1.

Programming Languages:	<b>Java and C# (either/or/mixed) + optimizer</b>
Hardware Requirements:	8K RAM, 64K Flash (8bit/16bit/32bit CPUs)
Supported Mote Hardware:	IRIS (Memsic), RZUSBSTICK (Atmel), AVRRAVEN (Atmel),
Programmable Middleware:	Control, customization, setup, and testing of mote networks.
Mote Simulation:	Debugging, testing, analysis of sensor networks including power consumption, sensor feeds, inspection, tracing.
IDE:	Source level distributed debugging using Eclipse.
Web Front-End:	Integration of WSN applications with web-based front end.

*Table 3.1 Key features of the Mote Runner*

It allows programmers to use object oriented programming languages and development environments such as C# and Java to develop portable WSN applications that may be dynamically distributed, loaded, updated, and deleted even after the WSN hardware has been deployed. The Mote Runner operating system is targeted at small embedded systems. The programming languages C# and Java have a number of adaptations in order to run effectively on embedded systems. These changes are all aiming to improve the performance; small footprint, high bytecode throughput and reduced RAM usage.

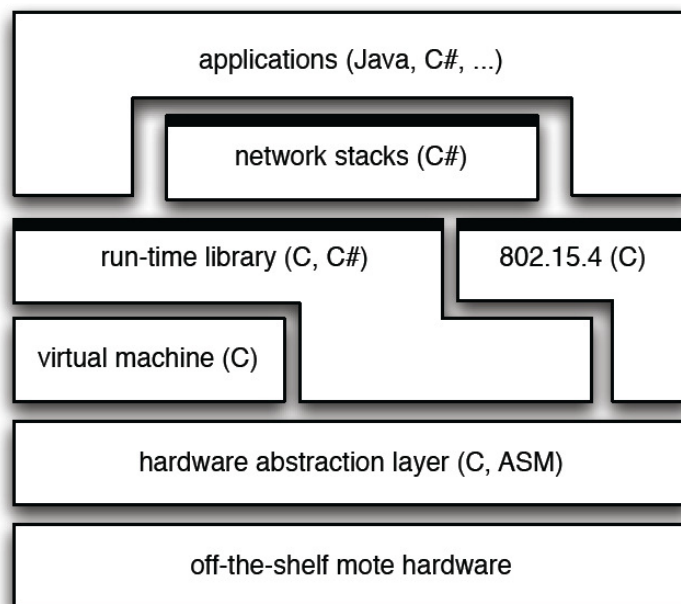


The Mote Runner provides a virtual machine for executing byte codes and an operating system to organize access to different devices and to schedule the various activities. The virtual machine in Mote Runner provides only a single thread of execution. An application registers callbacks with operating services which will be invoked on certain events.

Mote Runner WSN applications provide seamless integration with state of the art back end infrastructure by means of an event driven process engine effectively bridging the gap to large scale business scientific applications without requiring particular technology skills.

### 3.2.2 *Mote Runner architecture*

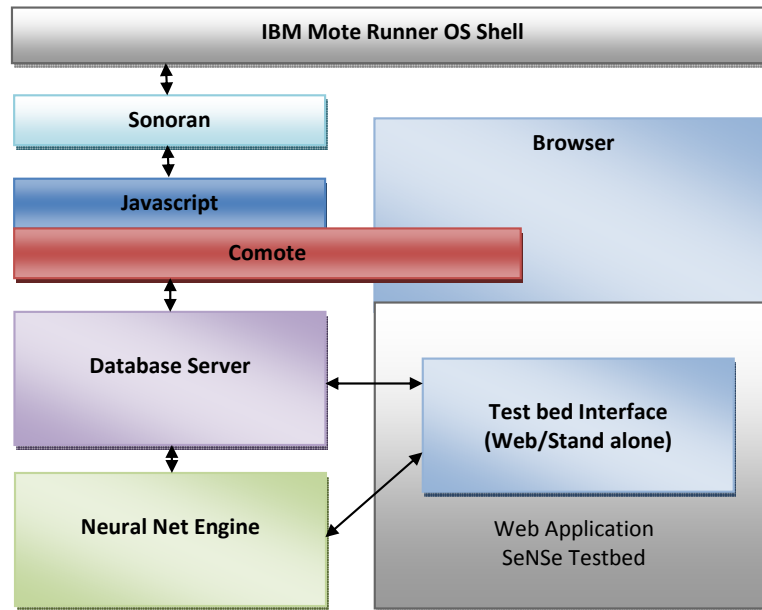
The Mote Runner implementation builds upon off the shelf embedded (Mote) hardware with a thin hardware abstraction layer written in C and Assembler encapsulating any hardware specific functionality. At the next layer, there is a virtual machine (written in C), runtime library (written in C and C#) and 802.15.4 MAC layer (written in C). Runtime library and the MAC layer expose APIs for application development in higher level languages such as C# and Java.



*Figure 3.1 Mote Runner Architecture [110]*

Figure 3.1 shows the Mote Runner Architecture. We have procured RZUSBSTICK and AVRRIVEN WSN modules from Atmel for the experimental setup at our

SeNSE (Sensor Network and Smart environment – <http://www.sense.ac.nz>) Research laboratory. Figure 3.2 shows the SeNSE test-bed architecture.



*Figure 3.2 SeNSE Testbed Architecture*

The Atmel's modules can support Mote Runner operating system, as IBM has released the firmware to support these. The nWSN concept is initially developed in a simulation environment and later Atmel modules are used to implement nWSN structure and QnDP algorithms discussed in Chapter 5 and 6.

#### 3.2.2.1 Flexsim Discrete Event Simulation

The nWSN concept development requires a three-dimensional space where the space can be generated via a grid based thermal profile to mimic the real system. This space is further used to deploy sensor nodes in a virtual reality environment. These nodes can be capable of sensing the temperature data at its location programmatically. Hence, a three-dimensional environment is required to evaluate the concept for its ability to run various scenarios. These customized nodes can be simulated as fixed nodes or mobile nodes. We have used Flexsim virtual reality simulation environment for designing this space, where the sensor nodes are created and customized to deploy in the space.

Flexsim is a powerful simulation and modeling software. This software can help to make intelligent decisions in design and operation of a system. A real time three dimensional environment is provided to build a model of a real life system. Flexsim

is more general purpose simulation software. The objects available within the software can be customized by using the C++ programming language and inbuilt functions.

#### 3.2.2.2 *Visualization and 3D engine*

Flexsim is a highly visible technology that can be used by forward thinking researchers. It is surprising how effective an animated simulation model can be for getting people attention and influencing their way of thinking. The animation displayed during a simulation provides a superb visual aid for demonstrating how the final system will perform. It has a powerful three dimensional engine that supports high performance interactive three dimensional vector graphics. The graphics library uses OpenGL technology. We can write code in C++ and using custom libraries/functions for creating the objects in the given space. The concept modeling and validation uses the data generated by the simulation environment.

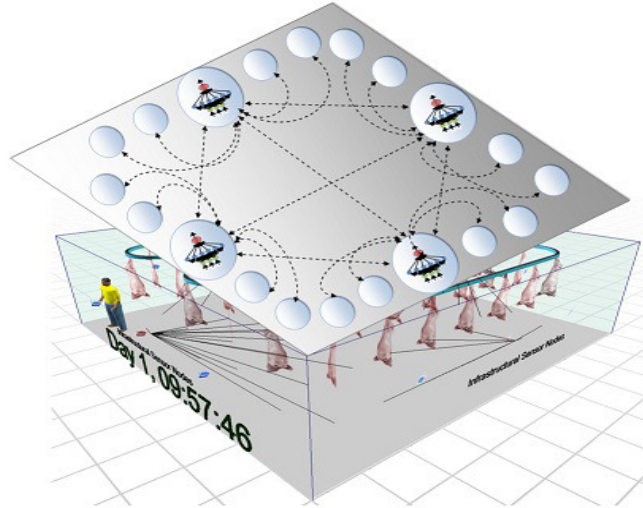
#### 3.2.2.3 *Model views*

Flexsim uses a three dimensional modeling environment. The default model view for building models is called an orthographic view. We can also view the model in a more realistic perspective. It is generally easier to build the model's layout in the orthographic view, whereas the perspective view is more for presentation purposes. However, we may use any view option to build or run the model.

#### 3.2.2.4 *Flexsim objects*

There are mainly two basic objects named Fixed Resources and Task Executors. The available model objects in the library are all derived from these basic objects. For the nWSN concept development, we have customized a fixed resource object and created an infrastructural node and a portable node. Further neural net algorithms are written by using C++ and user defined functions within the modeling environment. The custom Graphical User Interface (*GUI*) can be built for user input while running the simulation model. All the cold storage defined spaces for various scenarios are created in the Flexsim environment for concept modeling and are discussed in Chapter 7. The schematic diagram of the cold storage built in Flexsim is shown in Figure 3.3.

The infrastructural and portable node objects can be created dynamically and placed within the modeling environment to setup the scenarios. An example GUI built for running these scenarios is shown in Figure 3.4.



*Figure 3.3 Schematic diagram of the cold storage*

*Figure 3.4 GUI built in Flexsim to setup a scenario*

### 3.2.3 Artificial Neural Net Software

Peltarion Synapse software [105] is used to run the neural net algorithm from the data generated by the simulation model to identify the best architecture and program validation. Peltarion Synapse can generate the neural net algorithm into a .NET DLL which can be connected to the simulation environment to calculate the

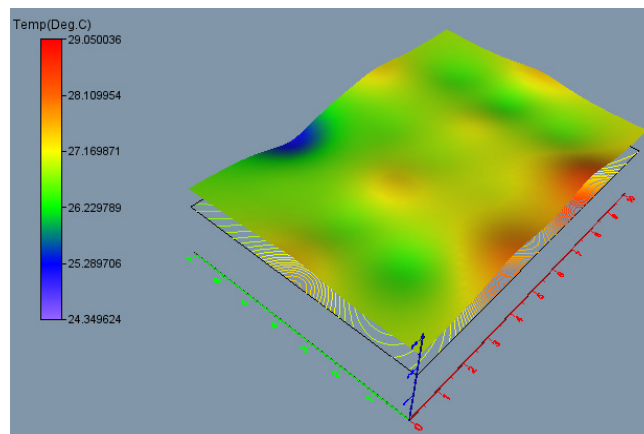
prediction. The program can be written in C# language and several neural net architectures can be tested for their performance.

Peltarion Synapse is the most advanced development environment for adaptive systems. It allows the user to thoroughly analyse and process the data to design, train, post-process and deploy adaptive systems. Synapse has the most power algorithms and training architectures within its integrated development environment, hence it is used to compare various architectures for suitability to the application domain.

#### 3.2.4 *Voxler visualization program*

Data visualization is important for any three dimensional data analysis. The volumetric rendered maps can produce a better comparison than any two dimensional charts. We have used Voxler visualization software for producing these three dimensional maps.

In this research work, we have used Voxler for volumetric rendering the temperature data within the specified space. Voxler is a powerful visualization program oriented primarily towards volumetric rendering and three dimensional data display.



*Figure 3.5 Volumetric rendering data produced by voxler*

In our scenarios, we have all the temperature data available in three dimensional space and Voxler has assisted throughout the conceptual and implementation phases for comparing the actual temperature to the predicted temperature as shown

in Chapter 5, 6 and 7. The volumetric rendering temperature data at the SeNSe laboratory as shown in Figure 3.5.

### 3.3 Hardware Selection for WSN

One of the challenges is to decide what is the right hardware environment for this work. There are several commercial vendors who supply different hardware with various specifications. However, most of these vendors do not support a simulation mode where the algorithms can be built and tested before embedment. In simulation mode where we can write the algorithms and run offline to test the logic before it can be deployed; a reduced developmental life cycle can result.



*Figure. 3.6 Wireless sensor node from Atmel*

IBM has the supported firmware for Atmel's WSN nodes and IBM's Mote Runner has a simulation mode. Hence we have procured these sensor nodes to develop and test our conceptual model. The AVRRAVEN has an AT86RF230 2.4 GHz radio transceiver and each kit contains two AVRRAVENs (LCD module) and one RZUSBSTICK (USB dongle). The RZUSBSTICK uses a communication device class creating a virtual COM port. This will allow simple communication between the host PC and the RZUSBSTICK. Atmel's AVR Wireless radio transceivers are designed to be compliant with IEEE 802.15.4 physical layer requirements that specify a mode of transmission where the RF output is off unless an active message packet is being sent.

# Chapter 4

## 4 Spatial Analysis: Thermal Mapping

### 4.1 Introduction

The temperature fluctuation has a significant effect on the quality of the products stored in a given space. Hence it is essential to identify a methodology for spatial mapping that can monitor the dynamics of the environment. This chapter introduces the ideology of the nWSN. It focuses on the architectural overview of an integrated solution of neural network and WSN as applied to spatial analysis and thermal mapping. The Neural Network approach is compared with the Shepard's algorithm [100] modified Euclidian distance.

### 4.2 Ideology of the nWSN

The expected wide deployment of WSN can be used to predict the environmental, physical and behavioural attributes to manage and control the data. The physical attribute, that is temperature, is the main element considered for the data collection and analysis in this work. The space to be monitored is assumed to have the following components.

- 1) Static Infrastructure – which allows for placement of sensing nodes,
- 2) Portable Objects – which could carry sensing nodes,
- 3) Energy Absorbers,
- 4) Energy Feeders.

While the first two represent the key generic components that facilitate the accommodation of the sensors that sample the space, the second two act as the main sources that influence the environment. However the elements (3) and (4) are not considered within the modeling, since they represent the excitation to the environmental dynamics. The assumption here is that at any moment in time the temperature of any point at the given space is in an equilibrium state. Hence, the temperature at any point in time is given by equation 4.1 below.

$$T(x,y,z) = f(\text{element } 3, \text{element } 4)$$

4.1

The nodes  $N_1$  to  $N_k$  deployed on the infrastructure shown in Figure 4.1 are stationary sensing nodes. Some of these nodes are elected as data collectors for other nodes and may have more computational capabilities than other nodes.

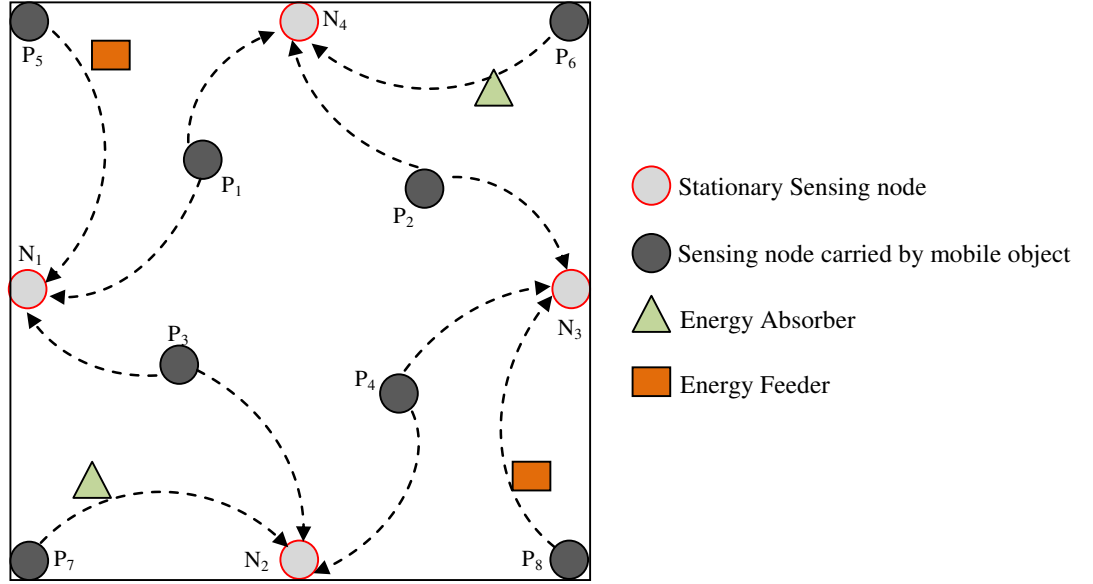


Figure 4.1 Ideology of the problem

The nodes  $P_1$  to  $P_k$  are the sensing nodes that are carried by mobile objects and  $P_5$ ,  $P_6$ ,  $P_7$  and  $P_8$  nodes are stationary. The nodes can provide sample temperature readings at their specific location within the space. The temperature at any arbitrary position can be estimated through either analytical or knowledge based approach by using the sample data from all sensor nodes available within the confined space.

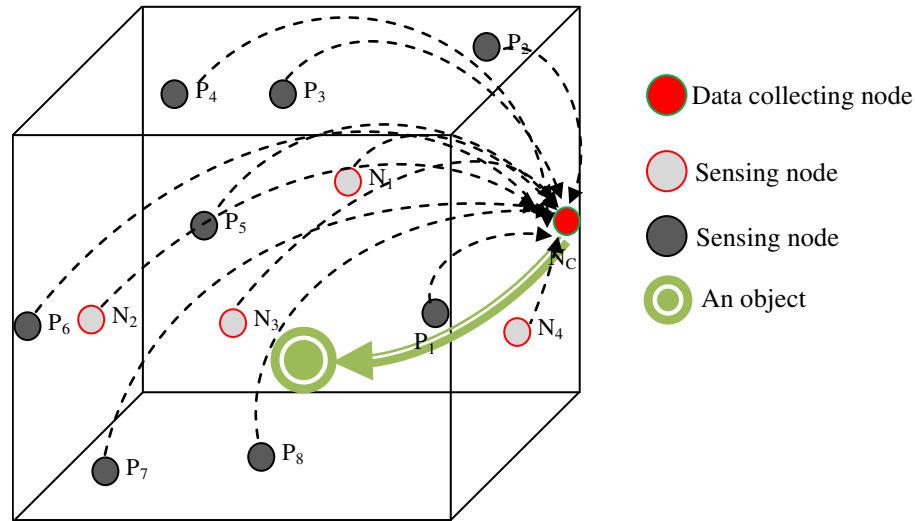
### 4.3 Object-Centric Intelligent Environment

An object centric intelligent environment utilizes the temperature sensing samples provided by the scattered sensors within the confined space and calculates the temperature at the location of a specific object. These objects could be human, food material, plant or any other object that requires information about its temperature exposure. An intelligent environment has the ability to sense its current physical and computational environments. The objects are considered to be families of products that can vary from plants to human beings, each of which is characterized



by a set of commonalities. In theory, the environment designed for thermal mapping can accommodate any object.

For example, if the application area is a greenhouse, then it drives towards a plant-centric environment. Controlling and monitoring temperature in greenhouses affects growth and development processes directly. The main climate state variable is typically the air temperature, which must be controlled to achieve proper plant development.



*Figure 4.2 Conceptual diagram of the object-centric environment*

Another example is that of fruit cool stores, where pallets containing fruits, are required to retain temperature records over time. Figure 4.2 gives a conceptual diagram of the object-centric environment for thermal mapping. The given object-centric environment consists of three components, as follows.

- 1) Data collecting node
- 2) Sensing node
- 3) An object.

The sensing nodes are either portable ( $P_1, P_2 \dots P_x$ ) or fixed ( $N_1, N_2, N_x$ ) where  $x=1,2,3 \dots n$ . The data collecting node is responsible for gathering the information from each sensing node. These data collecting nodes act as a cluster head for the given sensing nodes. The temperature in the vicinity of the object is required to be identified. Figure 4.3 shows the objects activity pseudo code and that gives an overview of the object centric model.

It is possible to use environmental information to guide the cluster head in response to varying environmental conditions. This research drives towards an object-centric

intelligent environment for thermal mapping. The proposed concept will have the possibility to incorporate additional information into the system by combining prior knowledge of the environmental objects that exist and the system reacts according to the condition.

---

```

//Objects activity


---


objectsActivity( )
{
    Initialize node parameters
    Do
        New object identified
        Object's status initialised and recorded
        Loop for all sensor nodes
        {
            Request data from the sensor nodes
            Collect location and temperature data
        }
        Initialize the algorithm
        Feed data into it
        Run the algorithm
        Generate thermal profile
        Identify temperature in the vicinity of the object
    While (new object identification)
}

```

---

*Figure 4.3 Objects activity – an overview of the object centric model*

## 4.4 Thermal Mapping Methodologies

In many agricultural production systems the accumulative data on the surrounding temperature dynamic behavior is vital for identifying the product condition. There are several methodologies applied in various research studies for thermal mapping to measure temperature at a given location within a space. In a WSN arena, this task is challenging, as most of the data processing algorithms are time consuming and are difficult to implement in real time. As discussed in the literature review, most of the conventional methods are not suitable for the WSN arena when frequent update is required as an alternative approach to the analytic approach. The following section discusses two approaches used here. These are Shepard's algorithmic solution and Neural Net based solution.

#### 4.4.1 Thermal mapping using Shepard's algorithm

It is assumed that each sensing node in the given space knows its location information  $(x, y, z)$  and attribute value temperature  $(T)$  at any given time. Hence, any given node defined in the space has its location and attribute information represented by expression 4.2 and equation 4.3, where  $w_k$  is the weight function for a given node  $N_k$ , which is a function of Euclidean distance  $d_k$ .

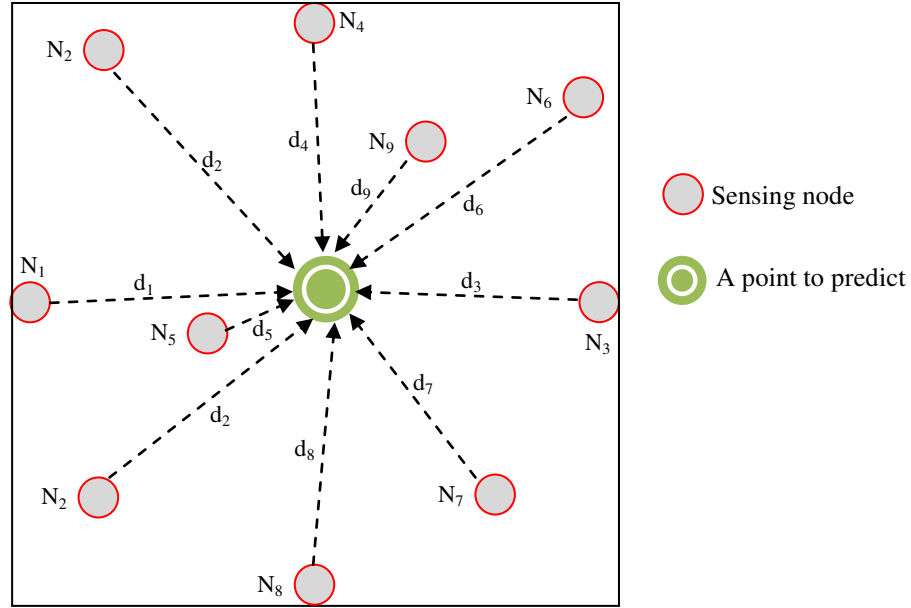


Figure 4.4 Predicting temperature at a point using Shepard's algorithm

$$N(\text{Location}_{(x,y,z)}, \text{Attributes}_{(T)}) \quad 4.2$$

$$w_k = f(d_k) \quad 4.3$$

Since we focus on dynamic temperature mapping, existing approaches such as Shepard's algorithm [100] with modified Euclidean distance are used to test the performance. This model uses a weighted average of surroundings or neighbouring nodes data to compute the temperature by using an interpolated function based on the distance as shown by Figure 4.4.

Let  $\mathbb{R}^m$  denote  $m$ -dimensional Euclidean space in a dataset  $S \subset \mathbb{R}^m$  where  $S = \{(x_0, y_0, z_0), \dots, (x_a, y_b, z_c), \dots, (x_l, y_m, z_n)\}$  which contains a number  $k$  of portable sensor nodes located randomly at points  $(x_a, y_b, z_c)$  in such a way that

$\mathbb{R}^m = \{(x, y, z) : x, y, z \in \mathbb{R}\}$  at points  $x=x_a$  at any  $a=0,1,2,\dots,l$ ,  $y=y_b$  at any  $b=0,1,2,\dots,m$ , and  $z=z_c$  at any  $c=0,1,2,\dots,n$ .

The attribute value, i.e. temperature ( $T$ ) at a given query point  $q$  in the space, can be evaluated by equation 4.4.

$$T_{q(x, y, z)} = \frac{\sum_{j=1}^k w_j T_j}{\sum_{j=1}^k w_j} = \sum_{j=1}^k w'_j T_j \quad 4.4$$

where  $T_{q(x, y, z)}$  is the estimated or predicted temperature at any query point  $q(x_a, y_b, z_c)$ , and  $w'_j$  is the weight of each portable node to the query point and it is given in equation 4.5.

$$w'_j = \frac{d_j^{-1}}{\sum_{j=1}^k d_j^{-1}} \quad 4.5$$

$d_j$  is the Euclidean distance between the sensing node and the predicting point for each  $j=1, 2, \dots, k$ . Figure 4.5 shows the pseudo code written to compute the temperature prediction.

---

~~**#Compute the temperature at a point  $q(x,y,z)$**~~

---

Temperature\_at\_q(x,y,z)

{

    Initialize the number of nearest sensing nodes –  $k$

    Loop for all number of sensing nodes for each  $j=1$  to  $k$

        Calculate *distance* ( $d_j$ ) =

$$\sqrt{(q_x - N_x)^2 + (q_y - N_y)^2 + (q_z - N_z)^2}$$

        Square the *distance*:  $d_j^2$

        Invert the *distance*:  $1/d_j$

        SumInverseDistance +=  $1/d_j$

    Loop for all number of sensing nodes for each  $j=1$  to  $k$

        Calculate  $Weight_{(j)} = \text{Inverse the distance } 1$

---

SumInverseDistance

    Loop for all number of sensing nodes for each  $j=1$  to  $k$

        Predicted Temp at  $q(x,y,z)$  +=  $Weight_{(j)} * N_{temp}$

}

*Figure 4.5 Pseudo code for Shepard's algorithm*

The number of sensing nodes within the confined space does have an impact on the prediction accuracy. Hence, there is a tradeoff between the number of sensing nodes and the accuracy of the temperature prediction. The distribution of the sensing nodes is another factor which has an effect on the prediction. Balanced sampling of the space is therefore important. Shepard's algorithm is always dependent on the distances and offers low computational overhead. However, it has a poor performance on the scenarios that have been conducted in our experiments.

#### 4.4.2 Neural net based solution to thermal mapping

Based on the literature analysis, ANNs are used widely in the field of agriculture, manufacturing, food and pharmaceuticals. ANNs are also applied to model nonlinear systems (Ex. greenhouse environment) and they are particularly useful for handling nonlinearities and dealing with nonlinear function mapping.

In a dynamic environment where temperature changes with time, online training is required to update the parameters of the neural net. Several approaches can be considered based on the application within the ANN.

In Feed Forward Neural Networks (*FFNN*) the network connections are directed from input layer to output layer. All the sensing nodes within the confined space feed their location and temperature data to their cluster head as show in the Figure 4.6. In this network, there is an input layer consisting of 3 neurons, an output layer consisting of one neuron and a hidden layer with six neurons.

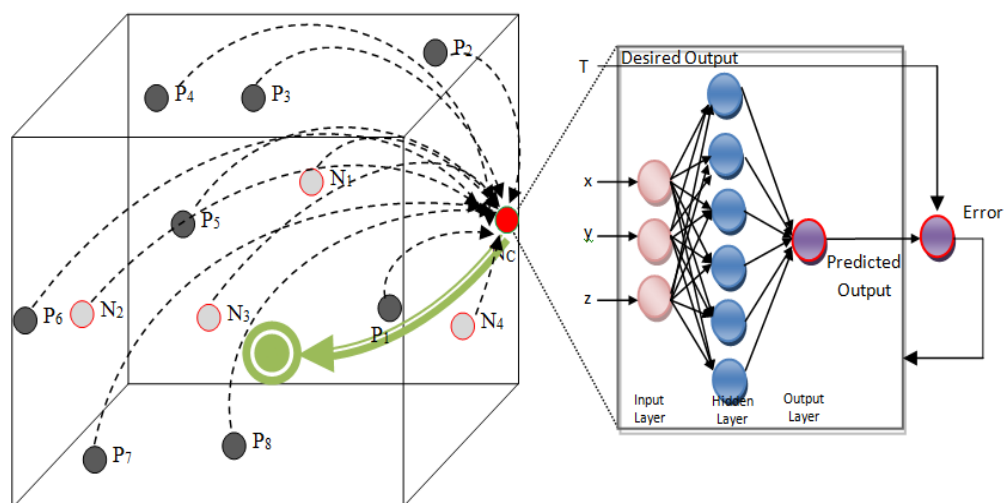


Figure 4.6 FFNN for thermal mapping

The input neurons get their data from all the given sensing nodes  $N_1$  to  $N_k$ . This data contains the sensing node location information  $(x,y,z)$  and the temperature  $(T)$  at that point. The information from each sensing node delivered to the cluster head is considered as a ‘sample’  $(Pt)$ , and is used to train the network. Hence the sensing nodes pattern can be represented by training data  $(Pt_1, Pt_2, Pt_3, \dots, Pt_n)$  at regular time intervals,  $t_1, t_2, t_3, \dots, t_n$  where  $n$  is the time interval. The on-line learning is accomplished, by ensuring each propagation is followed immediately by a weight update. The training is done by using back-propagation in two passes. The forward path is used to evaluate the output of the neural network for the given input in the existing weights. In the reverse path, the difference in the neural network output  $(T_n)$  with the desired output  $(T)$  is compared and fed back to the neural network as an error to change the weights of the neural network.

The key parameter that affects the precision is the number of iterations to run the back propagation algorithm for each sample received from any sensing node. On the other hand, the number of hidden layers can also influence the output precision. The following section discusses the nWSN structure and the impact of internal parameters of the neural net architecture for thermal mapping.

#### 4.4.3 Data collection and node implementation

The neural network learning process means finding an appropriate set of weights that influence the inputs to the neurons. The neurons are interconnected and each neuron operates by multiplying each incoming signal by a weight and then summing the weighted inputs. The iterative process for determining appropriate weights is called neuro-learning. The least mean error square is commonly used for minimizing the objective function. This is given by equation 4.6.

$$E_c = \frac{1}{2} \left[ \sum_{i=1}^n \sum_{j=1}^N (v_{ij} - o_{ij})^2 \right] \quad 4.6$$

where  $v_{ij}$  is the desired output (conditioning value),  $o_{ij}$  is the actual output,  $n$  is the number of samples in the training dataset that are fed from all the sensing nodes, and  $N$  is the number of nodes of the output layer.  $E_c$  is the network cumulative error of the nWSN, which is used as a criterion for learning. The learning continues until  $E_c$  converges to an acceptably small value, which is less than 1. The rate of

convergence is governed by the learning rate and momentum as neural parameters. Equation 4.6 is more generalized and more appropriate if the confined space contains more than one single cluster head. The neural net algorithm is embedded in all the cluster heads. The pseudo code written for the nWSN neural net is given in Figure 4.7

---

```

//Train the neural net at a cluster head


---


Train_net( $x,y,z,T$ )
{
    Initialize number of sensing inputs ( $n$ )
    Initialize neural weights, learning rate, momentum
    Collect sensing location ( $x,y,z$ ) and temperature ( $T$ )
    While not terminating condition (training error<0.001 or 10,000
epochs) do
        Initialize index for hidden and output layer –  $i, j$ 
        Initialize sum
        Loop for all hidden neurons
        {
             $Sum += inputs[j] * h_l\_weights[j][i]$ 
             $h_l\_a[i] = f(sum - h_l\_threshold[i])$ 
        }
        Loop for output neuron
        {
            for( $j=0; j<numHnodes; j++$ ) {
                 $sum += h_l\_a[j] * o_l\_weights[j][i]$ 
                 $o_l\_a[i] = f(sum - o_l\_threshold[i])$ 
            }
        }
        if((Target Temp-output)>training error)
            Adjust neural weights
        else
            beak;
    end While
    Repeat for all sensing inputs
}

```

---

Figure 4.7 Pseudo code for nWSN cluster head

## 4.5 Computation and Thermal Coverage Focus

The computation in the nWSN architecture is represented by distributed patterns of activity where it takes place in a decentralized manner. The main advantage of this concept is to minimize the computational burden on nodes, as it can lead to an automatic generalization. In real time monitoring systems, the data has to be

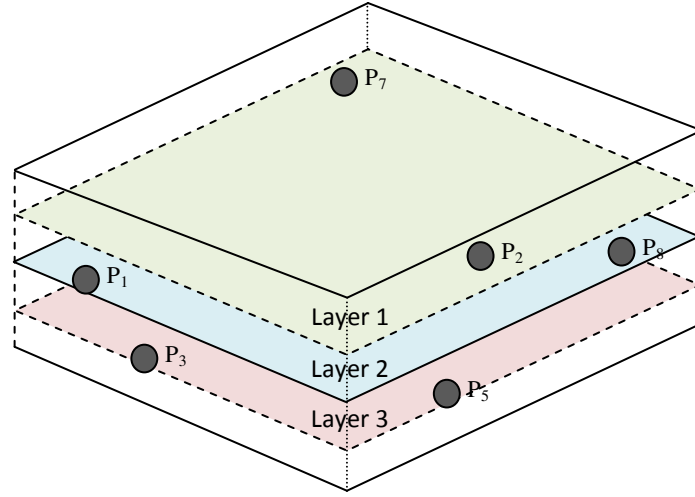
updated within a time interval. These systems are scheduled to update the data based on the application.

The optimal deployment strategy within a confined space is important to maximize the coverage using the resource constrained sensor nodes. The optimal placement of nodes would minimize the number of nodes required, minimise cost, and reduce communication overhead while maximizing the coverage of the space. A typical deployment of WSN consists of a few hundred nodes which are distributed randomly or over a predefined distribution within the space to be recorded. In most of the applications, sensor nodes sense the environment and provide information to the sink or base station. But collaboration of multiple sensor nodes within a confined space would allow much more advanced tasks to be accomplished effectively. This means grouping of sensor nodes and performing tasks would allow wide areas of applications. In this scenario, the cluster head (infrastructural node) has a capability to compute certain tasks that leads to application specific decisions. One of the key issues here is to map the thermal profile with time (spatial-temporal mapping). This could be done by deploying WSN nodes that cover the local region where they can gather the temperature data as input to feed the cluster head. There is a computational burden to the cluster head where it can compute and generate a thermal profile. The accuracy of the thermal map varies with the number of sensing points and their placement strategy.

#### 4.5.1 *Space division based on layers*

The space can be divided into a number of vertical or horizontal layers. In a spatio-temporal mapping, the input data points can be collected continuously or discretely. In this work we have considered that the data points are gathered at discrete points. Accordingly, the space is also divided into layers, as shown in Figure 4.8. These layers can be used to map the profile at a specified time. Layer 1, Layer 2 and Layer 3 are divided for better coverage within the confined space to enable placement of the sensor nodes. These layers are partitioned horizontally for the meat plant application. It is given an importance to the beef cuts that are expensive in the retail market (Ex. Topside, sirloin). It is also considered that the cold air is denser than warm air. Hence, the horizontal layers can reflect the same for better analysis.





*Figure 4.8 Space divisions into layers*

#### 4.5.2 Coverage aspects

In nWSN, the concept of area coverage is considered as a measure of quality of service. But in our case, we are considering the thermal coverage of the confined space rather than the communication coverage. And of course the thermal coverage is a part of communication coverage. There are mainly three coverage types and these can be divided as follows.

- 1) Blanket Coverage,
- 2) Barrier Coverage, and
- 3) Sweep Coverage.

We have chosen type one, as it can be achieved by a static arrangement of nodes while maximizing the detection rates at arbitrary points within the sensing field. The coverage of a confined space requires an optimal number of nodes to be deployed in such a way that it can represent every point in the space.

In this work, we have used computational geometry and probability theories to address the thermal mapping coverage. Initially we have considered the sensing models and the concept of total coverage. A sensing model addresses the quality of sensing, or sensitivity gradually attenuates with increasing distance, as given by equation 4.7.

$$S_e(N_i, P) = \frac{\lambda}{d(N_i, P)^\phi} \quad 4.7$$

where  $S_e$  is the sensitivity of a sensor  $N_i$  at point  $P$ .  $\lambda$  and  $\emptyset$  are the sensor dependant parameters and  $d(N_i, P)$  is the Euclidean distance between the sensor and the point. A sensing range can be defined for each node and the sensitivity decreases with increasing distance. The basic model has been extended to a realistic one named probabilistic sensing model. The simplest communication model includes the probabilistic sensing model. It is assumed that each sensor node  $N_i$  is able to communicate only upto a specific distance (i.e. communication radius,  $R_{ci}$ ). At any arbitrary point which has been covered by more than one sensor node at the same time, each sensor node contributes a definite rate of coverage. This total coverage at any point is also described by equation 4.8.

$$C_{x_i y_i z_i}(\chi) = 1 - \prod_{i=1}^k (1 - c_{x_i y_i z_i}(N_i)) \quad 4.8$$

Let  $\chi$  be the set of nodes  $N_i$  where  $i = 1, 2, \dots, k$  whose sensing range covers the points  $P_{(x_i, y_i, z_i)}$ .  $c_{x_i y_i z_i}(N_i)$  is the probabilistic coverage of a point. The two nodes  $N_i$  and  $N_j$  are able to communicate with each other if the Euclidean distance between them is less than or equal to the minimum of their communication radii, when  $d_{(N_i, N_j)} \leq \min\{R_{ci}, R_{cj}\}$ .

## 4.6 Nodes Minimization Approach

The minimum number of sensor nodes required to map the given space is an important criterion. Each node acts as an intelligent agent that can run its own algorithm by itself. The following section describes the methodology involved in minimizing the number of nodes required in the given space. The important factor here is that each node scrutinizes its neighbours and keeps track of its data including its location and temperature values.

### 4.6.1 *k-neighbour search for nodes*

Each node will have the data of its neighbour nodes. The neighbour nodes  $k$  will be generated dynamically for each node. The algorithm defined in the node will run frequently with a time span. The  $k$ -neighbour nodes will be decided by the proposed algorithm outlined in Figure 4.9. The accuracy of the predicted temperature largely depends upon the effective selection of the  $k$ -nearest neighbours. The general  $k$ -

Nearest Neighbour Algorithm (*k-NNA*) has got a limitation; that is, once we choose the nearest neighbours, the selection remains unchanged throughout the process. Now in this proposed model, nearest neighbours count changes from one node to another node.

---

**//k neighbour search algorithm**

---

N = Number of nodes

k = Number of neighbour nodes

At any selected Node 'X':

Calculate the Euclidean distance to all (N-1) nodes and sort out and make the rankings from 1 to (N-1)

**Do** for all nodes

Calculate estimated temperature at Node 'X' by considering first three ranked neighbours

Estimate, Error = Temperature(actual)-  
Temperature(predicted)

**While** (N-1)

Identify the location (k count) where error is less

k count will be the valid count to predict Temperature at node 'X'

---

*Figure 4.9 k-neighbour search algorithm*

#### 4.6.2 *k-neighbour search for a query point*

Assume  $p$  is the sensor node identified by  $p(x, y, z)$  as the location of  $p$ .  $S$  is a set of nodes that may be selected for the process and should contain  $N$  nodes that could be elected for the process, where  $N \subset S$ . The sensor node calculates the Euclidean distance for all neighbour  $N$  nodes and they are ranked from 1 to  $n$ , so we have the distances  $d_1 \leq d_2 \leq \dots \leq d_n$ .

Among the specified set  $S$ , at least  $k$  nodes must be defined to estimate the temperature of the node  $p$ . It has been identified that the Dual Buffer Search (*DBS*) algorithm is more efficient to find the  $k$  value when compared to the others [103]. The *DBS* requires two buffers to sort out the neighbour nodes. Since  $p$  has its own neighbours ranked from 1 to  $n$ , we have a clue from the node ranked 1, say  $N_1$  and can give an initial guess and *DBS* is much more efficient with a smaller search between these two nodes. The results are discussed in the later chapter.

Let us assume  $S_p$  as the set of nodes for node  $p$  and  $\hat{S}_l$  as the set of nodes for node  $N_l$ . Hence at least the set of nodes contained in  $S_p \cap \hat{S}_l$  are the nearest neighbours to the node  $p$ . The set of nodes  $S_p \cap \hat{S}_l$  will be taken into consideration as  $k$ -nearest neighbours for node  $p$ .

#### 4.7 Extending the Solution through Space Partitioning

In the nWSN implementation based on the neural network approach, the coverage regions could be divided into sub regions or subspaces where they can overlap each other as shown in Figure 4.10. To map the thermal coverage we are looking at the overlap of the coverage regions of each cluster head (infrastructural node). Hence the profile generated by each infrastructural node is taken while mapping the subspaces. The influences of the multiple subspaces are studied for thermal coverage to map the overall space with increased profile accuracy.

The coverage of a confined space requires an optimal number of nodes to be deployed in such a way that they can cover every point in a space. To map the thermal coverage, we looked at the overlap of the coverage regions of each cluster head. These scenarios examine the surface temperature variation around the subspaces.

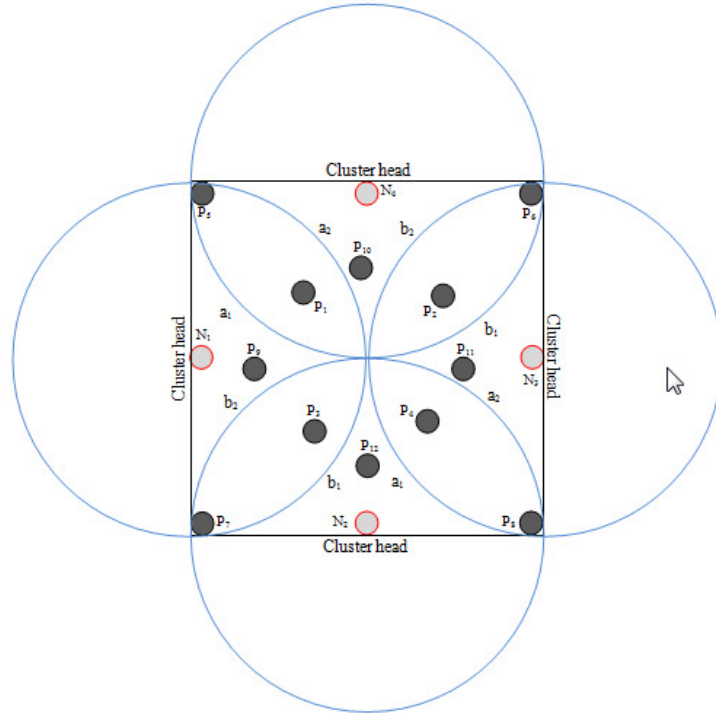
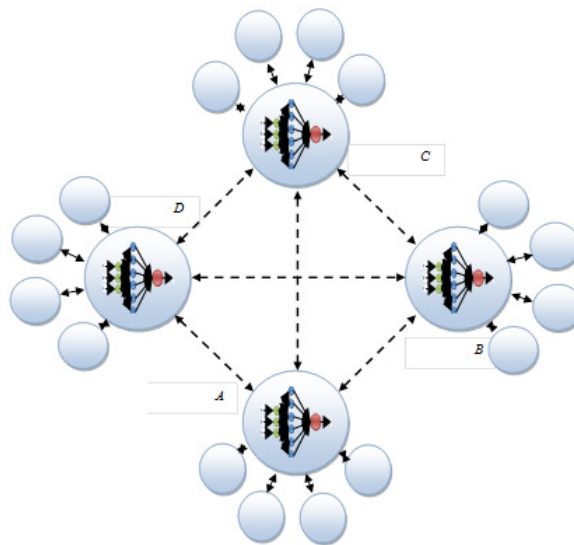


Figure 4.10 Subspace partitioning and their regions

These results will benefit to mainly focus on how the different cluster subspaces would help to predict the temperatures more precisely at the surface boundary. We have four regions, where it can overlap and that form subspaces within the whole space. Each region represents a population of sensors managed by a cluster head. A boundary exists among these four sub regions. The sensor nodes that can cover these boundaries can share information on the given region boundary cluster head. Hence we are looking at a point to analyse the surface of the boundary. The regions are distinguished as  $a_1b_1$ ,  $a_2b_2$ ,  $a_1b_2$  and  $a_2b_1$ . The boundaries can be viewed at  $a_1a_2$  and  $b_1b_2$  as shown in Figure 4.10. The simulation results revealed the effect on the accuracy of thermal predictions at the given infrastructural node and are discussed in the simulation results chapter 7. These scenarios are built to evaluate the effect of subspace on the thermal accuracy. This subspace analysis is meant to identify a difference when utilizing more than one cluster head for computations. The next section describes the methodology of the nWSN structure for subspace partitioning.

#### 4.7.1 *nWSN structure for subspace partitioning*

The confined space can be divided into sub-divisions to improve the prediction accuracy and this also reduces the computational burden on a single node. Each cluster models its own sub-space that contributes its computations to the overall space. An example model scenario has 4 clusters. This intern has N equal to 4 in equation 4.6. The distributed node based nWSN structure is shown in Figure 4.11.



*Figure 4.11 nWSN Structure using 4 clusters*

Assume the sensing nodes that are attached to the objects move continuously or occasionally. There is a possibility that nodes migrate to another region. Some of the data collected by that node may still be useful for the previous cluster for synchronization purposes and hence the communication among cluster heads is essential here. This approach of messaging services between the clusters is discussed in the implementation of the neural net cluster dynamic grouping. A dynamic grouping model assists in gathering information from the required sensing nodes along with the temperature data, even after the carcasses move continuously to synchronise the data at the instant when the training is required. The requirement of synchronized data input to the cluster head for training is discussed in the upcoming section.

The cluster heads within the model are connected in mesh network topology. This service is used for computational and communication purposes. The sensing nodes that are attached to the mobile objects are dynamically connected with one of the cluster heads while they come into their cluster zone. The overall network formulates clusters to facilitate computations and message interactions. The cluster heads are shown in the Figure 4.11, where A, B, C and D can make use of Nodes Message interaction mode for message interactions among other cluster heads.

Dijkstra's algorithm is used to evaluate these models for finding the shortest weighted path from the sensing node to the cluster head. A dynamic programming model is designed to formulate a recursive algorithm, in which the shortest path length ( $L_I$ ) to a cluster head  $I$  can be expressed by equation 4.9.

$$L_I = \min_{(p,I) \in S} [L_p + \text{weight}(p,I)] \quad 4.9$$

where  $p$  represents the sensing (portable) node and  $S$  is the space. When the NMi Pnode interaction executes, the algorithm calculates and identifies the cluster head to join. The NMi Inode and Pnode interactions in the nWSN architecture is shown in the Figure 4.12.

Within the time interval, the algorithm runs to verify any changes. Alternatively, when the NMi Inode interaction executes, the querying system within the node triggers to redirect the relevant cluster head to act accordingly to give a response. The flowchart in Figure 6.9 explains the NMi Inode that responds in deciding the query to execute. The user can query for a specified location and the cluster head

can then evaluate and return the temperature value. The cluster head is the controller that initiates the process. If the specified query location is not within the range of the given cluster head, it will immediately pass it over to the corrected one. Let  $N_i$  be the cluster head (infrastructural node), where  $i$  is the node ID and  $p(x,y,z)$  is any arbitrary point within the boundary of the specified region. If  $p(x,y,z) \in N_i$ , the temperature value is evaluated within the node  $N_i$ . If  $p(x,y,z) \notin N_i$  the query will be forwarded to  $N_j$  where  $p(x,y,z) \in N_j$ .

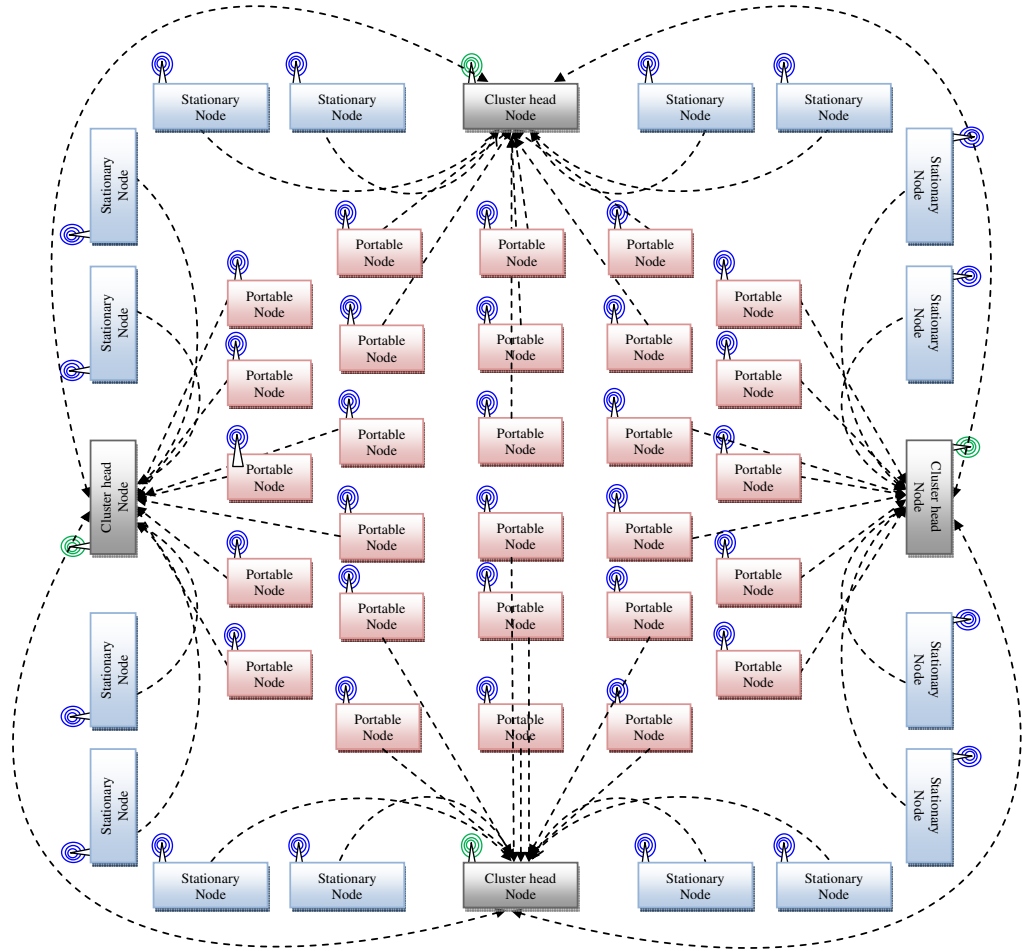


Figure 4.12 NMi Inode and Pnode interaction among the nodes in nWSN

## 4.8 Time Synchronization for nWSN Data Processing

The nodes which are attached to an object can be moved continuously or can be static. When the objects are moving, the data received from a sensor data to a cluster head reaches asynchronously. This reflects an inaccurate training and mapping of the space. Hence there is a requirement to consider a method where

training could be done synchronously at the cluster head. The approach for buffering and query management methodology is proposed to fulfil this requirement and accordingly, the QnDP algorithm has been developed.

#### 4.8.1 An algorithmic approach – QnDP algorithm

The query based data processing model consists of an assignment of the parameters within the nodes assembly. These parameters are dynamically assigned when a request is placed to any node. The queries receive and response state triggers for each epoch in the sensor node. The local variables data is allocated in byte array within the assembly. The cluster head/infrastructural node is allocated a buffer for the aggregation functions of interest. In this study, we have examined the aggregate functions including average [Avg()], maximum [Max()] and minimum [Min()] for temperature data at the cluster head level.

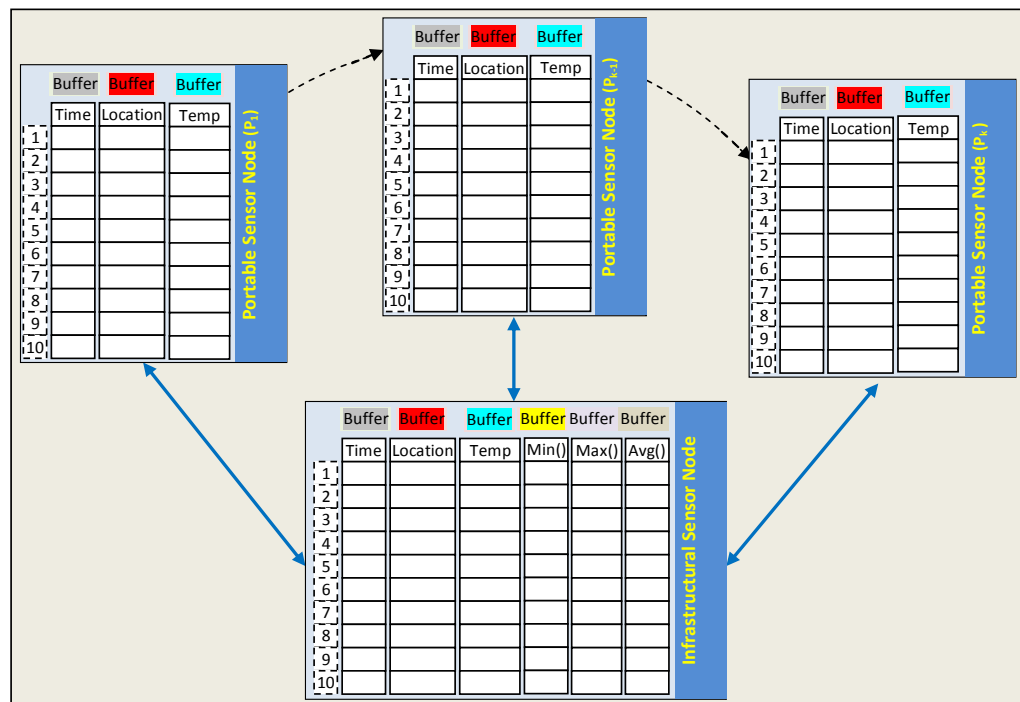


Figure 4.13 QnDP Memory buffer model

Along with these aggregate functions, the main parameters of time, location and temperature are assigned to it. **Error! Reference source not found.** describes the variable assignment data model for the defined parameters.



---

**Algorithm** Query based nWSN Data Processing (QnDP)

---

```
1: For each epoch e at buffer index k
2: Input: Selective message – parameters P ( $N_i$ , Temp, Avg,
   Max, Min, btime)
3: Output: Packet Data pD, Time t, Temperature Tc, Location
   L(x, y, z), epoch e, Buffer B[k], Node id  $N_i$ 
4: Start the application/Acquire Radio/Put into Receive Mode
5:   if (B[k] exists)
6:     set the Buffer data to local variables and update pD
7:     Transmit pD
8:   end if
9: Setup periodic Timer Call back to update B[k]
10:   if( $N_i$  equal to Node id)
11:     Set flag True
12:   end if
13:   check btime equal to btime at index k
14:   switch(parameter)
15:     case equal to Temp
16:       Set Temp buffer data at index k
17:       if(flag True)
18:         set buffer data to pD
19:         Transmit pD
20:       break
21:     case equal to Avg
22:       Set Avg buffer data at index k
23:       if(flag True)
24:         set buffer data to pD
25:         Transmit pD
26:       break
27:     case equal to Max
28:       Set Max buffer data at index k
29:       if(flag True)
30:         set buffer data to pD
31:         Transmit pD
32:       break
33:     case equal to Min
34:       Set Min buffer data at index k
35:       if(flag True)
36:         set buffer data to pD
37:         Transmit pD
38:       break
39:   end switch
40: Set B[k+1] to B[k] every 1Min, where B[k] is the current
   time
```

---

Figure 4.14 QnDP Algorithm

Each column is designated to allocate a parameter for a specified time as a row. Figure 4.14 describes the algorithmic approach for data processing for enhancing nWSN architecture.

# Chapter 5

## 5 nWSN Simulation and Validation

In this chapter, the nWSN structure is constructed in a simulation environment to validate various scenarios. Further testing on a local testbed at SeNSe laboratory is discussed. A Shepard's algorithm is used for comparison with the adaptive neural network solution. Other scenarios are set to test the ANN approach and analyse key parameters involved in the solution.

### 5.1 Introduction

A simulation model is constructed with a space volume  $20 \times 20 \times 3 \text{ m}^3$ . This size can be varied in the modelling environment. The code is written using C++ and custom libraries/functions of Flexsim simulation environment [104]. The nWSN is designed based on the following assumptions for the scenarios discussed in this section.

- 1) The three dimensional location of each deployed node in the simulation environment is known.
- 2) The stationary (infrastructural) nodes are considered to be deployed on the inner surface of the space boundary.
- 3) Both cases of steady-state and transient state are considered, as follows:
  - a) In steady-state mode, the temperature within the space follows a predefined profile. This profile is generated by assuming few temperature points and the volume is mapped within the simulation environment.
  - b) In transient mode, the dynamic behaviour is introduced within the environment by varying the temperature profile at all points between  $\pm 2^\circ\text{C}$  with the simulation time. The temperature fluctuation is assumed to be sinusoidal at a rate of 1 cycle/hour.

### 5.1.1 Temperature profile

In the simulation environment, a function that generates to fit a set of data points (x, y, z), by performing a polynomial regression analysis on each dimension using the least-squares method. Similar methodologies have been applied [107, 108, 109] to formulate the analytical solution for the three dimensional modelling and visualization.

In three dimensions a complete  $n^{\text{th}}$  degree polynomial is given by equation 5.1.

$$P_n(x, y, z) = \sum_{m=0}^n \alpha_m x^i y^j z^k \quad i+j+k \leq m \quad 5.1$$

where  $i, j$  and  $k$  are permuted accordingly. The number of terms in the above polynomial is equal to  $(n+1)(n+2)(n+3)/6$ .

For  $n=1$ :

$$P_1(x, y, z) = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 z \quad 5.2$$

The model uses quadratic polynomials that describe a three dimensional curve parametrically. The polynomials are determined by using Gauss-Jordan elimination on a matrix. Hence the distance between the curve and the input points can be minimized by calculating the partial derivatives.

The thermal profile is generated in the simulation environment and the sensing points are identified to evenly distribute over a specific elevated plane of the space. These sensing points are used as training set to calculate against the testing points. The test set is generated randomly using a uniform distribution and the probability distribution function is given by the equation 5.3.

$$f(x) = \begin{cases} \frac{1}{(b-a)} & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases} \quad 5.3$$

where  $a$  and  $b$  are real numbers with  $a < b$ ,  $a$  is a location parameter and  $b - a$  is a scale parameter.

There are 13 evenly distributed points used for sensing, which are divided into three different patterns as explained in the next section. Out of 60 extracted data points from the map, around 20% are allocated to sensing points and another 20 points are extracted from the 80% balance of the data set. The contour map along with the lattice slice generated based on the sensing points as shown in Figure 5.1.

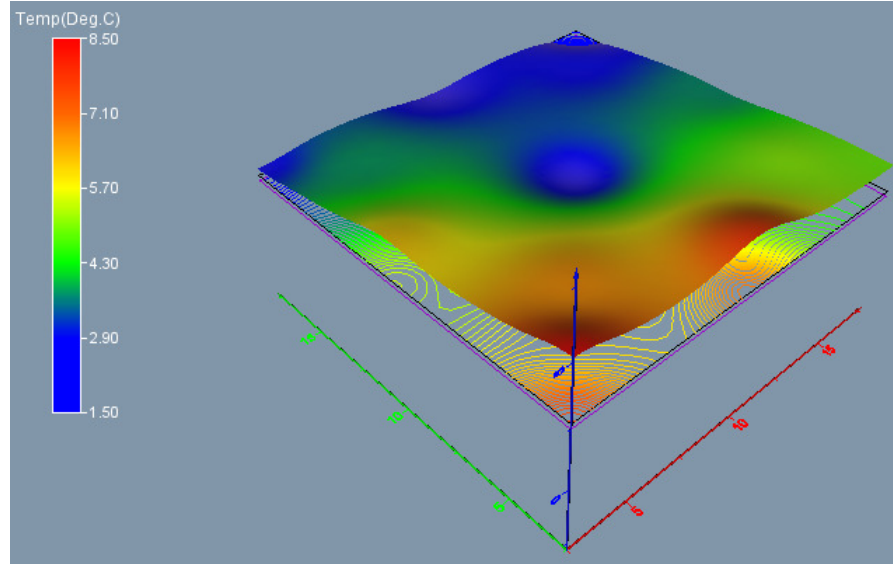


Figure 5.1 Contour map of the thermal profile based on the sensing points

Sno	X	Y	Z	Temp(°C)
1	5	5	1.5	5.9427
2	15	5	2.5	4.472
3	15	15	3.5	2.4708
4	5	15	4.5	3.3208
5	10	10	1.5	3.0154
6	10	2.5	2.5	6.9829
7	17.5	10	3.5	3.2486
8	10	17.5	4.5	2.1952
9	2.5	10	1.5	5.4068
10	2.5	2.5	2.5	7.5734
11	17.5	2.5	3.5	4.3257
12	17.5	17.5	4.5	1.465
13	2.5	17.5	1.5	2.4225

Table 5.1 Temperature profile used for the sensing data

The extracted sensing data are given in Table 5.1. The generated test data set is used to calculate the prediction error and Table 5.2 shows the thermal profile used for testing.

Sno	X	Y	Z	Temp(°C)
1	14.6993	11.1173	3.1163	3.2255
2	5.9293	9.2927	1.8406	4.4851
3	11.7076	18.6014	2.9507	1.7053
4	17.2856	12.3084	1.2442	2.9004
5	9.6011	13.8656	1.6928	3.0306
6	13.7048	11.7056	2.9408	3.1544
7	2.6908	11.4012	2.9139	4.9645
8	5.685	3.9114	2.1175	6.2354
9	17.1877	1.4868	1.028	4.4859
10	6.0663	16.7279	3.4719	2.7449
11	17.2047	7.7001	4.783	3.6627
12	16.7024	8.9071	3.7665	3.4848
13	5.7929	11.6882	1.5704	4.0047
14	7.1481	4.1476	3.7444	5.9532
15	12.7144	6.5196	3.6472	4.3424
16	18.7268	5.2014	1.5808	3.886
17	14.7833	9.0224	2.1189	3.5927
18	13.4463	14.8871	4.9531	2.6607
19	7.4092	16.3307	2.718	2.689
20	8.7183	19.5064	4.3714	1.8097
21	13.1249	18.6296	3.2838	1.8386
22	9.4167	13.5593	2.4651	3.0832
23	7.2684	2.9073	1.6824	6.5491
24	2.1574	13.2209	3.9101	4.3048
25	6.9424	1.3037	3.561	7.0779
26	13.0039	12.2017	1.439	3.1051
27	13.3063	18.2782	4.362	1.9704
28	7.2935	16.0156	2.5716	2.7845
29	10.0521	1.4301	2.964	8.0333
30	12.0799	2.9031	3.5125	5.9336
31	8.4805	7.7563	2.5125	4.2256
32	11.0565	18.5316	3.7132	1.7376
33	14.2504	15.5173	4.3064	2.4481
34	2.0586	4.0332	1.3208	6.8528
35	19.1492	11.627	3.1347	2.9469
36	13.3711	1.7635	4.5506	5.6952
37	15.7469	5.6687	4.5791	4.1934
38	8.7065	15.5604	4.0306	2.7822
39	15.6181	5.842	4.5448	4.1762
40	12.7068	18.0621	1.6719	1.9433
41	10.2478	4.0102	3.1752	5.7976
42	9.4463	3.1857	3.3174	6.3856
43	8.1547	13.9149	1.6684	3.1747
44	12.4025	10.4503	1.4194	3.3274
45	18.7413	18.5767	3.9482	0.3966
46	10.0977	19.3201	3.3241	1.1892
47	2.7012	12.9001	1.0427	4.3322

*Table 5.2 Thermal profile used for the test set*

In order to test the performance assessment of the network, the model involves obtaining the minimum statistical measurement of error between predicted temperature and actual temperature at any arbitrary points within the space. Actual temperature is the one which follows the temperature profile in a simulation environment and the estimated/predicted temperature is the value calculated from the algorithm used. The statistical measures that are evaluated are; MAE, Root Mean Square Error (*RMSE*) and Correlation Coefficient ( $R^2$ ) are expressed by equation 5.4, equation 5.5 and equation 5.6 respectively.

$$MAE = \frac{1}{k} \sum_{i=1}^k |T_{Act,i} - T_{Est,i}| \quad 5.4$$

$$RMSE = \left[ \frac{1}{k} \sum_{i=1}^k (T_{Act,i} - T_{Est,i})^2 \right]^{1/2} \quad 5.5$$

$$R = \sqrt{\frac{\sum_{i=1}^k (T_{Act,i} - \bar{T}_{Act,i})^2 - \sum_{i=1}^k (T_{Act,i} - T_{Est,i})^2}{\sum_{i=1}^k (T_{Act,i} - \bar{T}_{Act,i})^2}} \quad 5.6$$

where  $\bar{T}_{Act,i}$  and  $\bar{T}_{Est,i}$  are the average actual and predicted temperatures for the  $i^{th}$  observation, respectively.  $k$  is the total number of readings taken from the model. Using the statistical measures, a well trained ANN model should always turn out small MAE and RMSE with large  $R^2$  values.

### 5.1.2 Transient model behaviour

In order to generate a dynamic behaviour within the model, it is considered to fluctuate the temperature at any given point between  $\pm 2^\circ\text{C}$  with the simulation time. The temperature fluctuation is assumed as a sinusoidal form of cycling with a period of 1 hour. Hence the fluctuation of temperature at any time is given by the equation 5.7

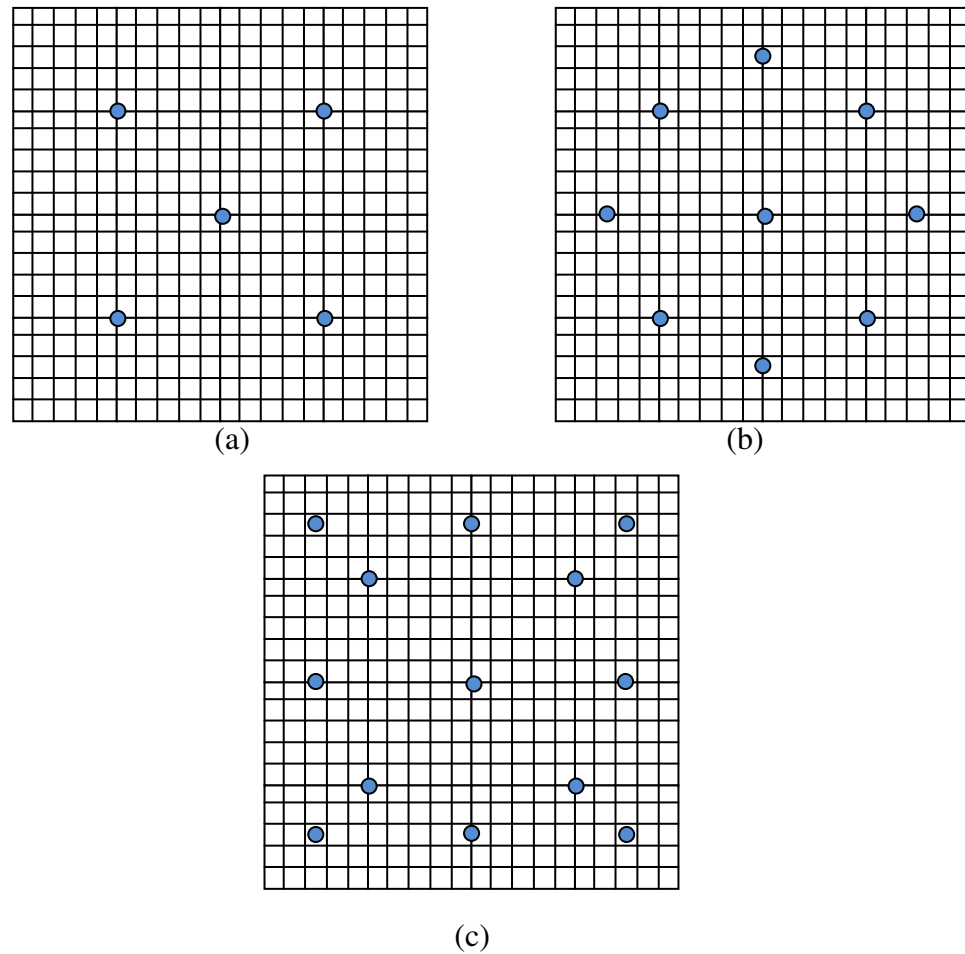
$$T_{(t)} = (\sin[\text{time}() * 2\pi/\text{period}] * h_y + h_y) - h_y \quad 5.7$$

Where  $h_y = 2^\circ\text{C}$ .

The following chapter will give more focus on transient model behavioural study as part of the cold storage within the meat industry (a case study).

## 5.2 Thermal Mapping Based on a Fixed Sensing Points with Single Cluster Head/Infrastructural Node

Initially we have considered sensing points based on a pattern in a given space volume  $20 \times 20 \times 5 \text{m}^3$ . The sensing points are located in three different patterns to examine the variation of thermal prediction error using Shepard's and ANN algorithms. Figure 5.2 shows the sensing point location patterns in a two dimensional view.



*Figure 5.2 Sensing location points (a) pattern with 5 nodes (b) pattern with 9 nodes (c) pattern with 13 nodes*

These sensing points are evenly distributed over the region that can cover the overall space. The pattern with five nodes may not cover the space as much as the pattern thirteen nodes can cover. The extracted temperature data for the given patterns are shown in Table 5.1.

The model is computed and calculated for the location points where temperature is known other than the sensing points given in the Table 5.2. It is observed that

Shepard's method has generated more prediction errors at all the five, nine and thirteen node patterns. There is a significant decrease in error from five to thirteen node patterns. Figure 5.3 shows the MAE and RMSE for Shepard's and ANN methods.

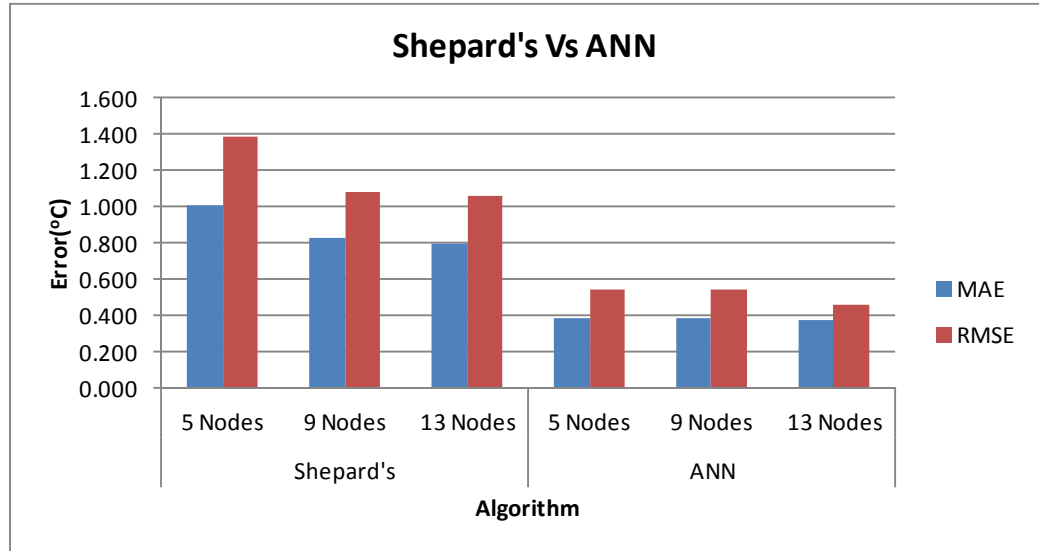


Figure 5.3 MAE and RMSE comparison between Shepards and ANN

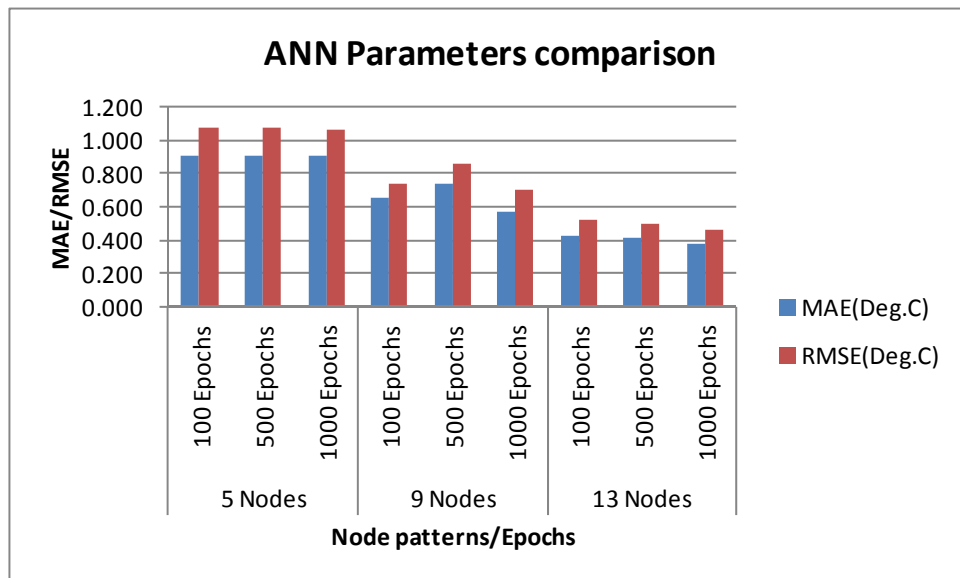


Figure 5.4 ANN Parameters comparison for training

A further analysis has been done for ANN architectural parameters. A decrease in prediction error is observed with the increase of the number of epochs for thirteen nodes pattern compared to the five nodes pattern. There is not much difference in prediction error even with a larger number of epochs for the five nodes pattern.



This is due to the lack of coverage of the nodes. The number of epochs can even lead to converge to local optima instead of global optima. This can be avoided by keeping the model's fitness criteria while training.

Figure 5.4 shows the ANN parameters comparison when changing the number of iterations from 500 to 1000 at five, nine and thirteen nodes patterns.

### 5.2.1 *Impact of NN internal architectural parameters*

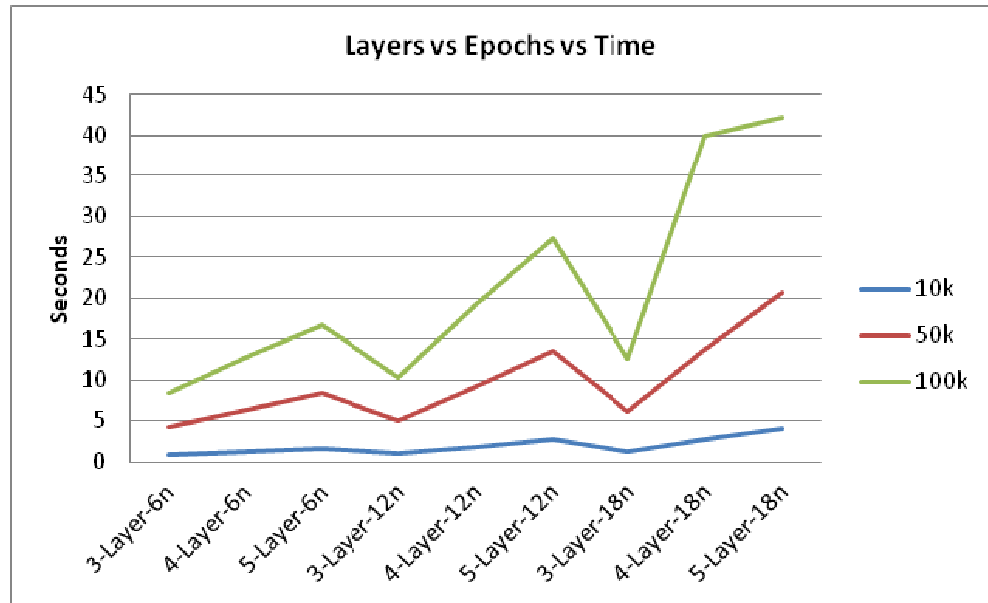
It is important to give attention and identify the best architecture of the neural network. In the experimental scenarios, the models are configured with different neural architectures to study the applicable method. The thermal mapping solution in this work uses the online training. The more internal structure a network has, the better that network will be at presenting complex solutions. But at the same time it may cause the training to diverge, which further leads to over-fitting. This would prevent the network from generalizing well to new data. The model configured with tan-sigmoid activation function in a MLP network has recorded a lower accuracy than those with other functions.

It is also observed that the accuracy didn't improve much with the number of iterations increased. Table 5.3 shows the initial parameters of the neural net architecture for thermal mapping. Initially the various NN architectures are evaluated for suitability of the thermal mapping application.

Minimum weight delta	0.0001
Epochs	10000
Initial weights	0.3
Learning rate	0.3
Momentum	0.6
Activation function	tan-sigmoid
Neurons in hidden layer	3

*Table 5.3 Initial neural net parameters for mapping*

We have used eight different function modelling architectures and among those the MLP has given the best results. It is observed that the MLP three and four layer performed better than any other architecture.



*Figure 5.5 Time comparison at different MLP layers and epochs*

The trade off among the number of layers and the epochs is considered based on the time required to run the algorithm. There is the risk of converging at local optima instead of global optima, when running through a specified number of epochs. Hence the algorithm saves the best system while running by calculating the  $R^2$  value. Figure 5.5 shows the time comparison to run the algorithm when changing the layers from 3 to 4 (at 6, 12 and 18 hidden neurons) and the epochs at 10,000, 50,000 and 100,000.

A three layer MLP model with six neurons at the hidden layer would require at least 0.9 seconds to run 10,000 epochs. To avoid converging to local optima, we have considered setting up the model to run at least 10,000 epochs. The best system identified in a simulation environment is at  $R^2$  value of 0.92. The algorithm run time is important when we have more training input. It is obvious that the increase in the number hidden neurons, layers and epochs has a greater impact on the resultant time based on Figure 5.5.

The Paltarion Synapse software is used to optimize the number of epochs, number of hidden layers and number of neurons in a hidden layer. The Genetic Optimizer module is used to identify the right parameters that can produce a minimum output error within 95% confidence level for training and validation procedures. The algorithm starts with a minimal network, then adds hidden nodes during training.

### 5.3 Thermal Mapping Based on Random Sensing Points

This model is considered to collect data in a steady-state environment. It has a single central data collection point and the sensing nodes are randomly placed. The central point gets the temperature and location data from 20 portable nodes as shown in Figure 5.6. The portable nodes  $P_1$  to  $P_{20}$  feed the temperature and location data to the infrastructural node  $N_1$ . We have analysed the predicted temperatures at any given points within the space by using Shepard's algorithm and the Artificial Neural Network's approach.

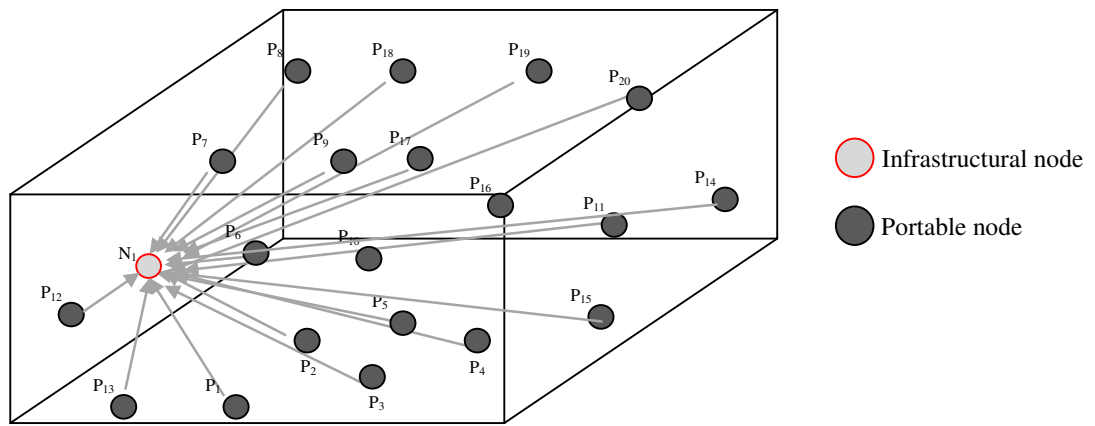


Figure 5.6 Sensing node distribution over the space

#### 5.3.1 Thermal mapping using Shepard's Algorithm

The model is computed initially using the Shepard's algorithm and tested against 20 arbitrary points in the space. MAE and RMSE recorded 0.89 and 1.24 respectively. The  $R^2$  value was recorded 0.80. The following Table 5.4 shows the arbitrary points selected to predict the temperatures.

Sno	X	Y	Z	Actual (°C)	Predicted(°C)
1	14.6993	11.1173	3.1163	3.2255	3.6894
2	11.7076	18.6014	2.9507	1.7053	2.8954
3	9.6011	13.8656	1.6928	3.0306	3.4809
4	2.6908	11.4012	2.9139	4.9645	4.0203
5	17.1877	1.4868	1.028	4.4859	4.4034
6	17.2047	7.7001	4.783	3.6627	4.0441
7	5.7929	11.6882	1.5704	4.0047	3.8144
8	12.7144	6.5196	3.6472	4.3424	4.2938
9	14.7833	9.0224	2.1189	3.5927	3.916
10	7.4092	16.3307	2.718	2.689	3.367
11	13.1249	18.6296	3.2838	1.8386	2.7763
12	7.2684	2.9073	1.6824	6.5491	4.3969
13	6.9424	1.3037	3.561	7.0779	4.4585
14	13.3063	18.2782	4.362	1.9704	2.9248
15	10.0521	1.4301	2.964	8.0333	4.8582
16	8.4805	7.7563	2.5125	4.2256	4.1342
17	14.2504	15.5173	4.3064	2.4481	3.2386
18	19.1492	11.627	3.1347	2.9469	3.7022
19	7.1481	4.1476	3.7444	5.9532	4.378
20	15.6181	5.842	4.5448	4.1762	4.2401

Table 5.4 Actual and Predicted temperatures using Shepard's algorithm

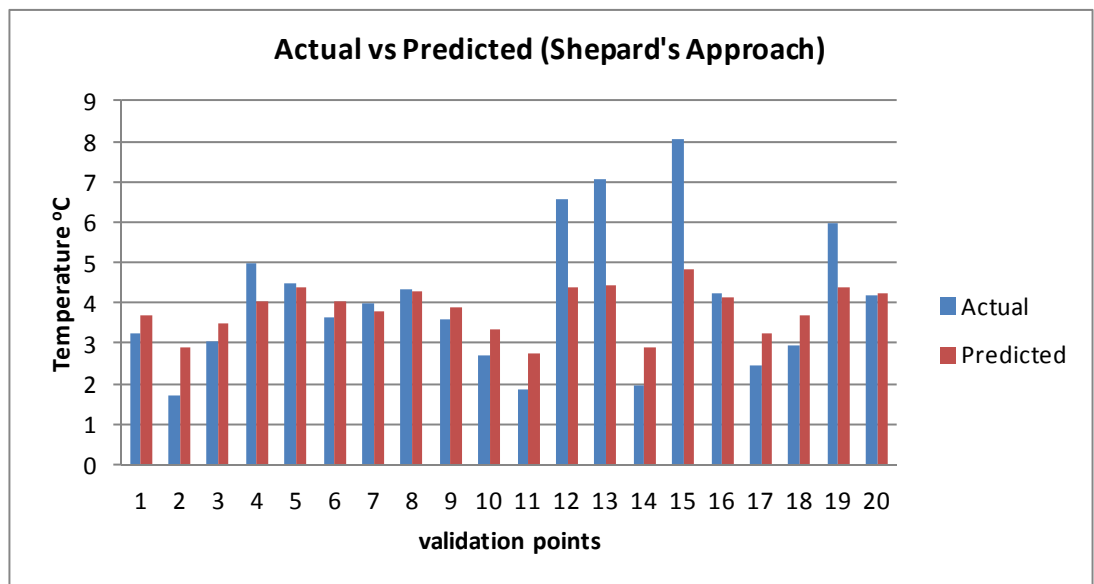
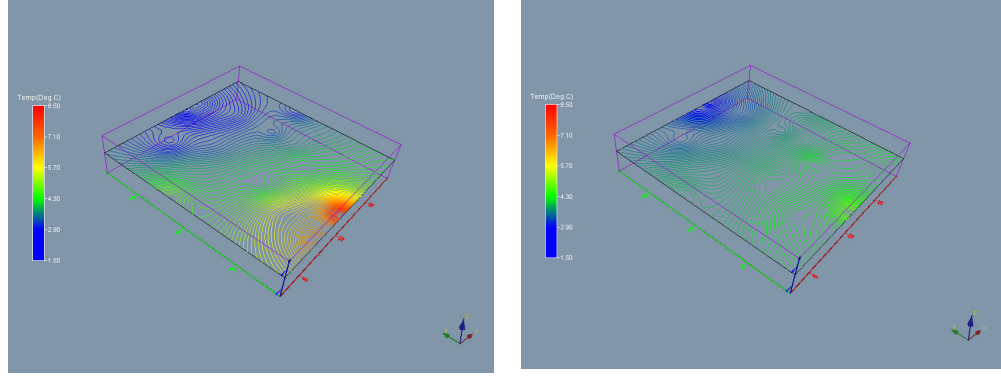


Figure 5.7 Actual vs Predicted temperatures

The contour maps shown in Figure 5.8 give a clear difference between the actual and predicted temperatures in the three dimensional space.



*Figure 5.8 (a) Actual thermal profile (b) Predicted thermal profile*

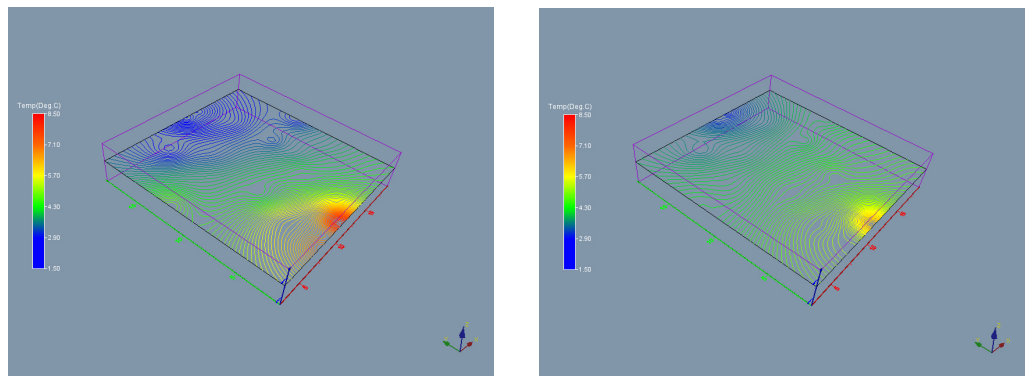
Shepard's algorithm was ineffective at identifying the hot spots which exists in the space. The high and low temperatures are poorly predicted as also shown in Figure 5.7 and Figure 5.8.

Based on the hot spots, a further three nodes were deployed near the hot spot region in the simulation mode and the profiles were recalculated. The identified location and temperatures are given in Table 5.5.

Sno	X	Y	Z	Temp( $^{\circ}$ C)
1	8.00	2.50	1.50	6.8028
2	10.00	1.50	2.50	8.1286
3	11.00	2.50	1.50	6.6687

*Table 5.5 Additional deployed nodes at hot spots*

The contour maps are constructed to identify the differences in the actual and predicted thermal profile as shown in Figure 5.9.



*Figure 5.9 (a) Actual thermal profile (b) Predicted thermal profile*

The predicted profile was improved moderately after placing the nodes at hot spots, increasing  $R^2$  to 0.89. But there are other locations where the hot spots are not still effectively identified. If the hot spots are not identified, there is a probability of deterioration of the products.

### 5.3.2 Thermal mapping using neural net approach

The Neural Net approach was used to compare with the earlier Shepard's algorithm. The MAE and RMSE are recorded as 0.35 and 0.52 respectively. The  $R^2$  value is recorded as 0.92. The following Figure 5.10 shows the arbitrary points selected to predict the temperatures.

Sno	X	Y	Z	Actual (°C)	Predicted (°C)
1	14.6993	11.1173	3.1163	3.2255	3.021
2	11.7076	18.6014	2.9507	1.7053	1.686
3	9.6011	13.8656	1.6928	3.0306	2.851
4	2.6908	11.4012	2.9139	4.9645	4.505
5	17.1877	1.4868	1.028	4.4859	5.201
6	17.2047	7.7001	4.783	3.6627	3.72
7	5.7929	11.6882	1.5704	4.0047	3.94
8	12.7144	6.5196	3.6472	4.3424	4.615
9	14.7833	9.0224	2.1189	3.5927	3.552
10	7.4092	16.3307	2.718	2.689	2.553
11	13.1249	18.6296	3.2838	1.8386	1.582
12	7.2684	2.9073	1.6824	6.5491	6.152
13	6.9424	1.3037	3.561	7.0779	6.605
14	13.3063	18.2782	4.362	1.9704	1.669
15	10.0521	1.4301	2.964	8.0333	6.237
16	8.4805	7.7563	2.5125	4.2256	4.776
17	14.2504	15.5173	4.3064	2.4481	2.081
18	19.1492	11.627	3.1347	2.9469	2.435
19	7.1481	4.1476	3.7444	5.9532	5.968
20	15.6181	5.842	4.5448	4.1762	4.46

Figure 5.10 Actual and predicted temperatures using ANN approach

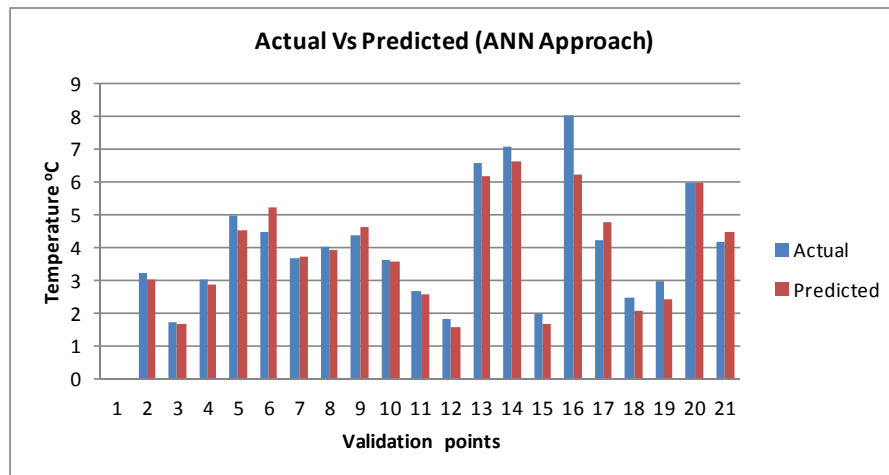


Figure 5.11 Actual and predicted temperatures

The actual and predicted temperatures for 20 sample data are shown in Figure 5.11. The contour maps shown in Figure 5.12 give a clear difference between the actual and predicted temperatures in the three dimensional space. There were few false positive and negatives recorded, among those the validation point 5 and point 15 have given a major difference.

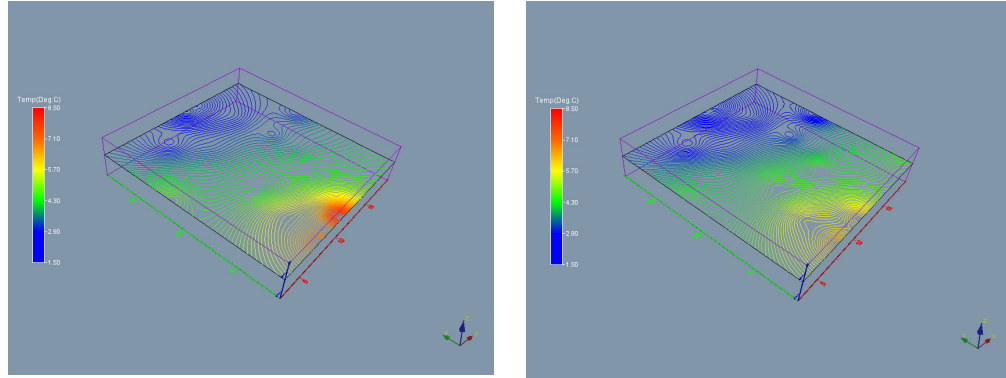


Figure 5.12 (a) Actual thermal profile (b) Predicted thermal profile

The ANN approach has the capability to identify the hot spots and the low and high temperatures are predicted with less error when compared to the Shepard's method. The additional nodes deployed as shown in Table 5.5 are used to identify the improvement for the hot spot prediction. There is a little improvement in the  $R^2$  to 0.93, and a better profile is generated as shown in Figure 5.13.

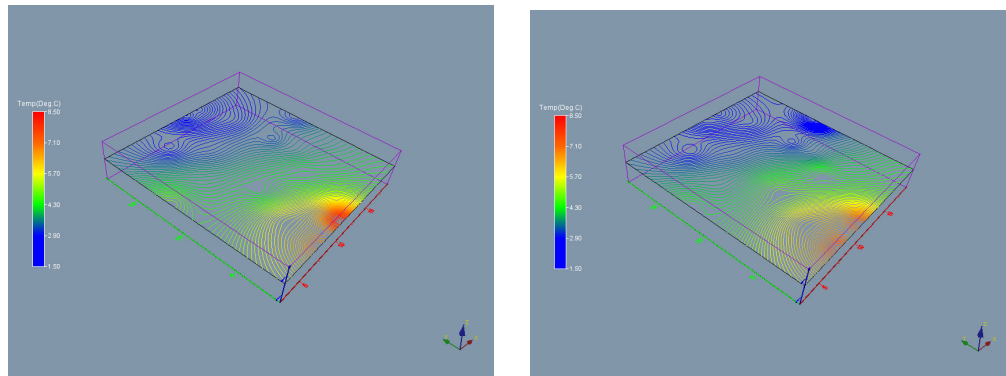


Figure 5.13 (a) Actual thermal profile (b) Predicted thermal profile

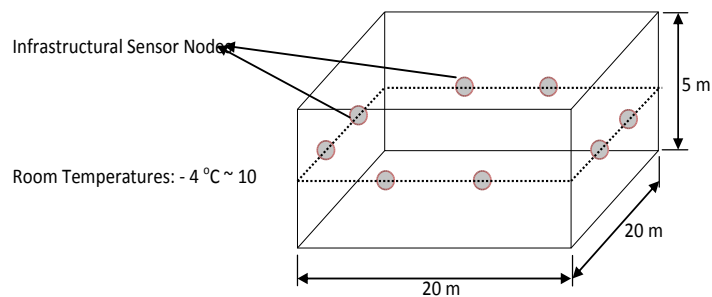
A few more scenarios were conducted to realize the ANN approach by varying the parameters that include the space volume and training data set. The earlier experimental model is a steady-state where the temperature at the given point does not change with time. But in real time applications like meat plant's cool storage the temperature varies with time. Hence it is important to consider a dynamic

environment within the modeling space. We have considered the Meat Industry cool storage as a case study to further compute the experiments in the upcoming chapter.

## 5.4 Thermal Mapping Experimental Scenarios

In these scenarios, the ANN approach is implemented in a simulation modeling environment by varying the number of infrastructural nodes and portable nodes, room volume and training data set. These experiments are meant to identify the applicability of the approach for different configurations. This further helps to study and improve the performance and the effecting factors towards the accuracy of the output.

The neural networks are used to model the temperature distribution in a space designed in a simulation environment. Suppose we have a room where the temperature is distributed with a minimum and maximum temperature ranges  $-4^{\circ}\text{C}$  to  $10^{\circ}\text{C}$  respectively. Figure 5.14 shows the experimental design for the ANN. The room dimensions are assumed as  $20 \times 20 \times 5 \text{ m}^3$ ,  $30 \times 30 \times 5 \text{ m}^3$ ,  $40 \times 40 \times 5 \text{ m}^3$  and  $50 \times 50 \times 5 \text{ m}^3$ .



*Figure 5.14 Schematic diagram of the experiment*

A model is designed using Flexsim software to emulate the relationship between the training data along with infrastructural nodes temperature data at various locations to estimate the temperature distribution at other arbitrary positions. A temperature profile is generated based on the given temperature points at few locations within the designated space.



No. of Scenarios	No. of Infrastructural Nodes used	Room Volume(m3)	Random Data Set	Training Data Set
1	8	20x20x5	20	28
2	16	20x20x5	20	36
3	24	20x20x5	20	44
4	32	20x20x5	20	52
5	40	20x20x5	20	60
6	24	20x20x5	20	44
7	24	30x30x5	20	44
8	24	40x40x5	20	44
9	24	50x50x5	20	44
10	8	30x30x5	10	18
11	8	30x30x5	20	28
12	8	30x30x5	40	48
13	8	30x30x5	80	88
14	8	30x30x5	120	128

*Table 5.6 Model run scenarios*

Table 5.6 shows the model run scenarios to test at different data sets assumed initially. The infrastructural nodes are deployed on the inner surface of the room to get thermal feed from the boundary. These nodes are varied from 8 to 40 based on the designed scenarios. The random data set is the data feed from the portable nodes. This data set also varies as the number of portable nodes change from each scenario.

A GUI has been built to input the required data for each scenario and the training data set is generated from the model. This training data set is generated by using a bounded continuous uniform probability distribution function. This training data set is nothing but the data supplied by the portable nodes.

The uniform distribution is essential in generating random variants from all the other distributions and will return a continuous set of random values inclusive of minimum and maximum variants. This training data along with the infrastructural nodes data is supplied to train the neural net.

The RMS error for each of the scenarios stated above has been calculated. Based on the results, each scenario of the model demonstrated noticeable change of error as a result of varying one of the parameters. The RMS error is given by the difference between RMS actual and estimated values.

Scenarios 1 to 5 are evaluated by changing the infrastructural nodes from 8 to 40 given in Table 5.7. It is observed that the RMS error decreased with the increase of number of infrastructural nodes for a given volume. Figure 5.15 clarifies the change

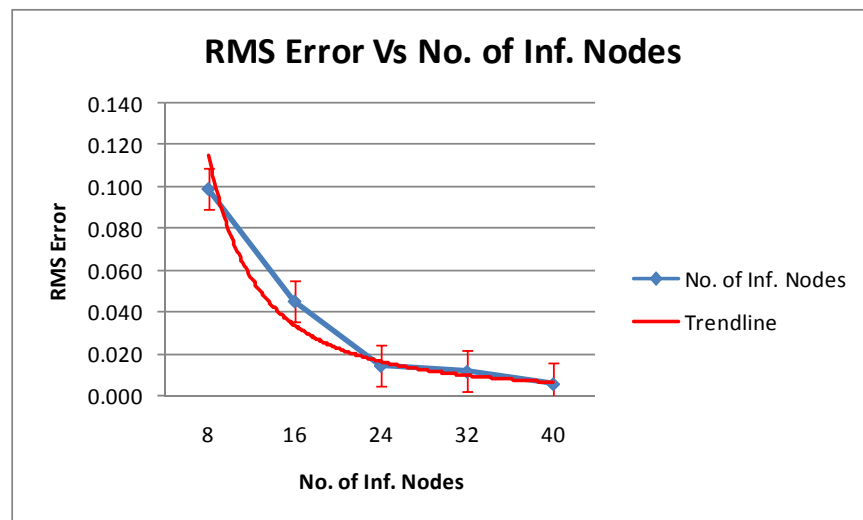
of responsiveness of RMS error with the increase in the number of infrastructural nodes.

	No. Nodes	Volume(x1000m <sup>3</sup> )	No. Training Data	RMS Error
Scenario 1	8	2	20	0.099
Scenario 2	16	2	20	0.045
Scenario 3	24	2	20	0.014
Scenario 4	32	2	20	0.012
Scenario 5	40	2	20	0.006

*Table 5.7 RMS Errors – Inf. Nodes changing 8 to 40 at 2000m<sup>3</sup> Volume and 20 Training data*

The vertical error bars at each point shown in Figure 5.15 are about 0.02°C for the given set of training data. This means an error  $\pm 0.02^\circ\text{C}$  is observed by running a number of iterations at each training of the neural net. At the room volume 2000m<sup>3</sup>, a minimum of 24 nodes are required to get at least 0.02°C RMS error.

Based on these results another set of test scenarios have been conducted in order to evaluate the room volume response by keeping 24 nodes, 20 training data set constant and varying room volume from 2000m<sup>3</sup> to 12500m<sup>3</sup>.



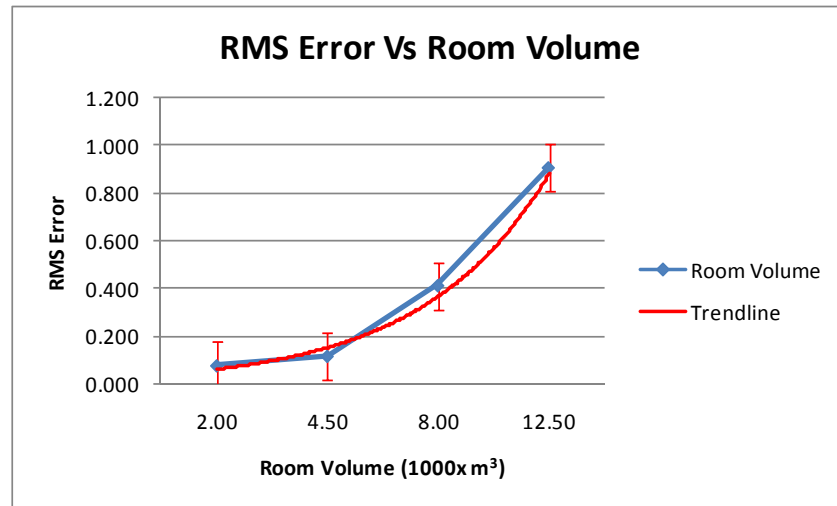
*Figure 5.15 RMS error Vs No. Infrastructural Nodes*

The RMS error has not affected much from 24 nodes to 40 nodes; hence it is considered to keep 24 nodes and 20 training data set constant for the next runs. Scenarios six to nine shown in Table 5.8 describe the RMS error actual and estimated values by changing the room volume.

	No. Nodes	Volume( $\times 1000\text{m}^3$ )	No. Training Data	RMS Error
<b>Scenario 6</b>	24	2	20	0.077
<b>Scenario 7</b>	24	4.5	20	0.116
<b>Scenario 8</b>	24	8	20	0.413
<b>Scenario 9</b>	24	12.5	20	0.907

*Table 5.8 RMS errors – Room volume changing  $2000\text{m}^3$  to  $12500\text{m}^3$  at 24 Inf. Nodes and 20 Training data*

It was also observed that an error  $\pm 0.025^\circ\text{C}$  is resulted for each time we trained the neural net. Figure 5.16 shows the drastic change of RMS error by increasing the room volume. The RMS error is increased from  $0.077^\circ\text{C}$  to  $0.907^\circ\text{C}$  for the  $2000\text{m}^3$  and  $12500\text{m}^3$  respectively.



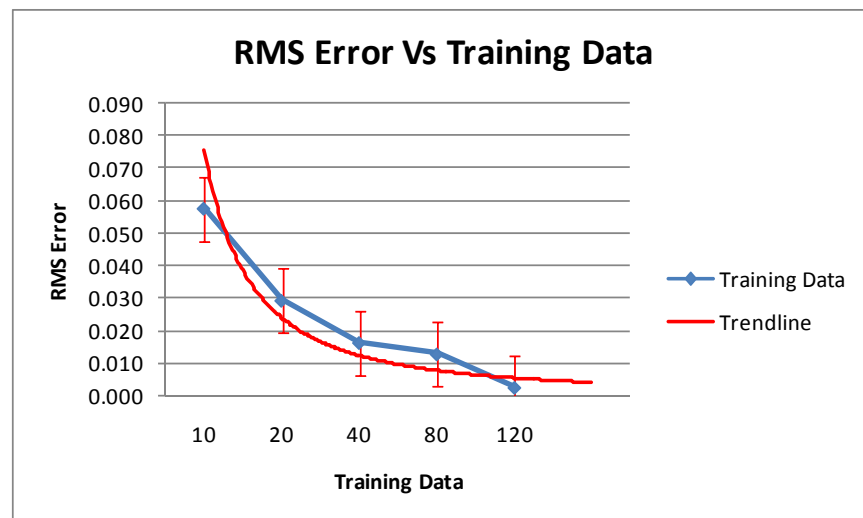
*Figure 5.16 RMS error Vs Room volume*

This clearly states that the given set of nodes and training data was not sufficient to keep the RMS error within the limits with the increase of room volume. Hence, further analysis was conducted to test the model for its suitability for thermal prediction using neural networks. This time another set of scenarios 10 to 14 are conducted by changing the number of training data set at a given room volume  $4500\text{m}^3$  and eight infrastructural nodes. We have chosen room volume  $4500\text{m}^3$ , based on the test results from Figure 5.16.

It clearly shows that the RMS error is somewhat within  $0.05^\circ\text{C}$  at the room volume of  $4500\text{m}^3$ . Table 5.9 shows the decrease of RMS error by increasing the number of training data set. The trend-line is given by the best fit to the RMS error data set.

	No. Nodes	Volume(x1000m <sup>3</sup> )	No. Training Data	RMS Error
<b>Scenario 10</b>	8	4.5	10	0.05747
<b>Scenario 11</b>	8	4.5	20	0.02921
<b>Scenario 12</b>	8	4.5	40	0.01627
<b>Scenario 13</b>	8	4.5	80	0.01276
<b>Scenario 14</b>	8	4.5	120	0.0025

*Table 5.9 RMS Errors – Training data set changing 10 to 120 at 8 Inf. Nodes and 4500m<sup>3</sup> volume*



*Figure 5.17 RMS Error Vs Training Data*

An error of  $\pm 0.01^{\circ}\text{C}$  is observed during the training period of the neural net and the vertical bar reflects the same. Figure 5.17 evidently illustrates that the increasing number of training data set produced a much better improvement of the RMS error. However, 8 infrastructural nodes and the room volume of 4500m<sup>3</sup>, allowed the temperature to be predicted with an RMS error of at least 0.02 $^{\circ}\text{C}$ .

Furthermore, the RMS error is almost reaches zero after 80 training data set for the given room volume 30x30x5 m<sup>3</sup>. However the increase in the number of training data set did not affect the RMS error after the training data set of 40. It is considerably lower than 0.01 $^{\circ}\text{C}$ .

The main conclusion possible from the different set of scenarios assessed in these experiments is that the greatest impact on the RMS error is through increasing the number of infrastructural nodes and training data set. Hence the number of infrastructural nodes can also influence the thermal prediction accuracy, since they are located on the inner surface of the boundary.

## 5.5 nWSN Testbed Experimentation

Based on the nWSN structure, a testbed has been constructed at the Sensor Network and Smart Environment (*SeNSe*) Research centre. We have used Atmel's RZRAVEN and RZUSBSTICK for the experimental setup and implementation. There are eight infrastructural nodes deployed in the room and each node is also responsible to sense the temperature at that point. These eight inputs are fed into a neural net for training. Each node is programmed using IBMs mote runner library. Apart from these eight nodes, there is another node which is connected at the base station to compute the thermal profile. Each node is programmed with a unique short ID and the location information is hardcoded and transmitted along with the temperature data. There are nine testing points selected to validate the model. These points were chosen randomly within the space and their actual temperatures were recorded. The schematic diagram of the room layout is shown in Figure 5.18.

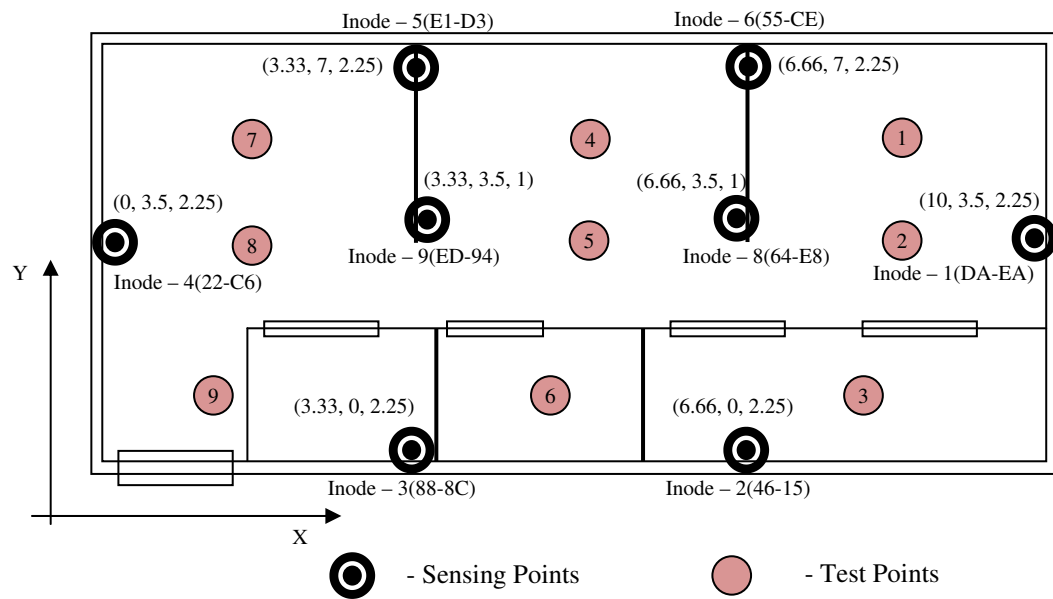
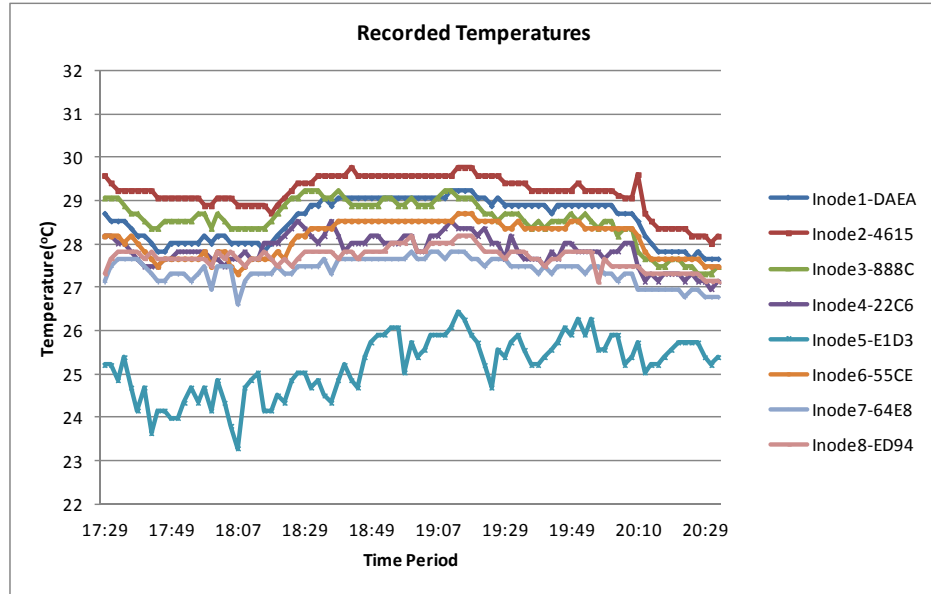


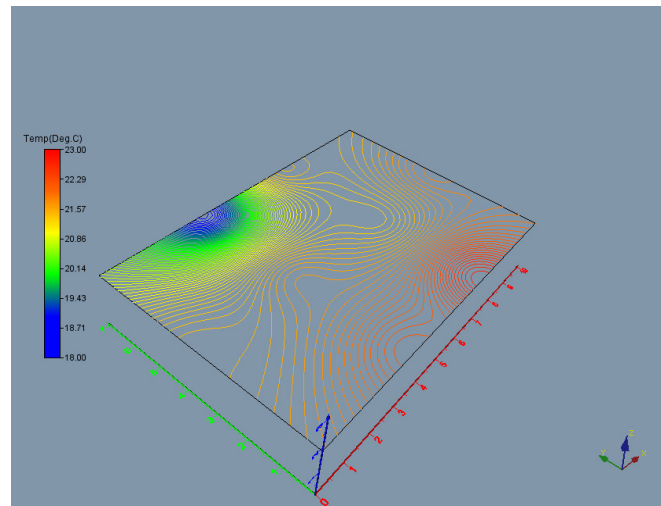
Figure 5.18 Testbed layout at *SeNSe*

Each node from Inode-1 to Inode-8 is assigned a short ID as shown in the layout. We have left the system running to identify the temperature variation during the mid year. The following Figure 5.19 shows a sample the recorded temperature data during the evening hours.



*Figure 5.19 Temperature profile of each sensor node*

The NN program is configured to use a one-layer MLP with 6 hidden neurons to generate the thermal profile of the space. The contour map of the actual room condition at the middle of the cross section is shown in Figure 5.20.



*Figure 5.20 Actual room temperature contour map*

The ANN training was performed at the node connected to the base station. The predicted temperature profiles are generated at a given instance of time. The actual temperatures at the test points are then compared against the predicted temperatures. An RMSE of 1.12°C is recorded for the given data and the one-layer MLP architecture has executed training with a 95% confidence interval of the output within  $\pm 0.012^\circ\text{C}$  as shown in Table 5.10.

Sno	Desired(°C)	Output(°C)	High95%	Low95%
1	22.2	22.200	22.211	22.188
2	21.85	21.850	21.862	21.839
3	20.21	20.203	20.215	20.191
4	18.7	18.698	18.710	18.686
5	21.15	21.138	21.150	21.126
6	22.7	22.695	22.707	22.683
7	21.5	21.501	21.513	21.489
8	20.74	20.728	20.740	20.716
9	20.7	20.693	20.705	20.681
10	21.78	21.777	21.789	21.766
11	21.3	21.299	21.311	21.287
12	21.5	21.500	21.511	21.488

*Table 5.10 Training data confidence plot*

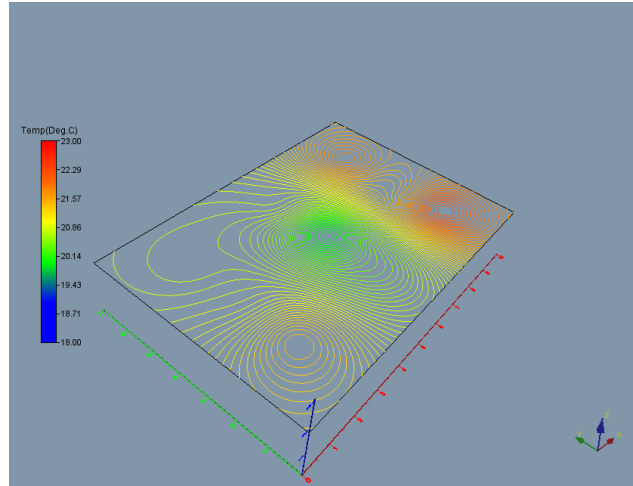
The validation set within the training data set produced with a 95% confidence interval an output within  $\pm 2.07^{\circ}\text{C}$  of the measured data as shown in Table 5.11. This clearly reflects the insufficient data set for the training and validation of the model.

Sno	Desired(°C)	Output(°C)	High95%	Low95%
1	20.36	21.208	23.282	19.133
2	19.14	20.938	23.012	18.864
3	22.72	22.873	24.948	20.799
4	21.8	20.527	22.602	18.453
5	22.1	22.028	24.102	19.954

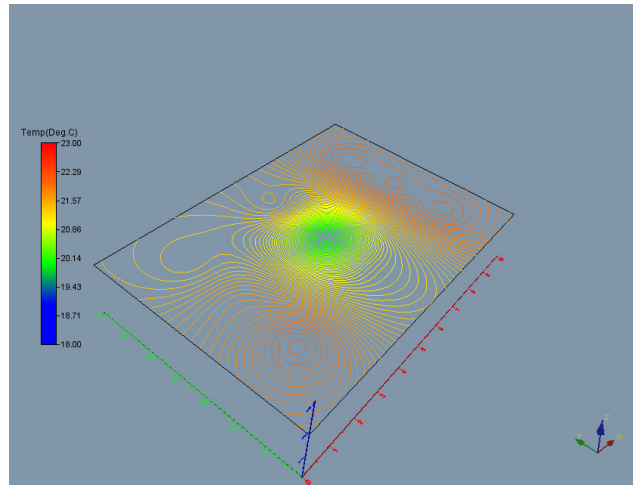
*Table 5.11 Validation data within the training set confidence plot*

The testing data is compared with the predicted data for the selected locations and the contour maps are drawn to examine the differences. Figure 5.21 and Figure 5.22 shows the actual test data contour map and the predicted data contour map, respectively.

The testing and predicted thermal maps have a similar profile on their contours with an accuracy of  $0.8^{\circ}\text{C}$ . There are two test points that have more influence than others due to less training and validation data set.



*Figure 5.21 Contour map of the testing data set*



*Figure 5.22 Contour map of the predicted data set*



# Chapter 6

## 6 nWSN Implementation Factors

This chapter focuses on components that are essential for real time development and implementation based on the methodologies discussed in chapter 4. The required components for nWSN implementation are identified. This includes the query based spatio-thermal mapping, time synchronization, neural net cluster dynamic grouping and node distribution and minimization. Each component has significance to the overall implementation of the nWSN architecture.

### 6.1 Components of the nWSN Implementation

The important components of the nWSN structure are classified below. Each methodology described in the earlier chapter requires an implementation and further study of the performance measure for results analysis.

- 1) Query based spatial thermal mapping,
- 2) Time synchronization,
- 3) Neural net cluster dynamic grouping, and
- 4) Nodes minimization.

In a dynamic environment, there is a requirement to feed the temperature data at a given point of time. In the case of portable nodes where they move with time, it may not be possible to acquire data without an implantation of a query system that fulfils the time synchronization. The implementation of query based spatial thermal mapping details these requirements.

The time synchronization component is essential for a modularised model for mapping a localized data. The solution has been extended to further space partitioning to verify the precision when considering the sensor nodes that are at the boundaries among the sub regions.

In several applications where the nodes deployment is a criterion, this is due to the restrictions of the surroundings and the working environment. It may not be

possible to deploy a node if there an infrastructural object does not exist. Hence the focus was given to look at the opportunity to minimize the number of nodes. The thermal precision is taken as performance measure for the implementation.

## **6.2 Approach for Query Based nWSN Spatial Thermal Mapping**

Based on the case study described in the upcoming chapter, it is identified that the surface temperature of the carcass in a cool storage changes from 30 °C to 4 °C within a period of 10 to 15 hours. This is a rapid change at the beginning of the process before it comes to a steady state. Hence the temperature monitoring during this period is vital, together with identifying hotspots. In the nWSN training process, the infrastructural nodes require the training data set from all the portable nodes at a given time to train the neural net. These nodes can process the data by filtering, aggregating and sharing within the network. The infrastructural nodes have to initiate a query to all the nodes to request the training input data at a synchronized time. To address these requirements we proposed a Query Based nWSN (*QBNWSN*) implementation to support a synchronized data input to the infrastructural node upon a query request. The infrastructural and portable sensor node layers are responsible for data processing.

The neural net algorithm requires training data from all the portable nodes within its range. The input data contains the portable node location ( $x, y, z$ ) coordinates and temperature recorded at that location. The neural net calculates the required parameters that further assist to train the network rapidly. These neural training can be facilitated to build a spatial thermal map. It can further assist to calculate temperatures at any arbitrary point. A memory buffer model is introduced to store the data at infrastructural and portable node layers. When a query propagates to all the portable nodes to request the training data, each node responds with its buffered data stored at that time. The buffer data for each infrastructural and portable node contains location and temperature data. This buffer data loops through every 1 minute to update the buffer table and to accommodate up to 10 minutes of pervious data within the node.

The location and temperature data are the parameters that are dynamically assigned for each query to respond at a given time. The QnDP algorithm executes for each time a query triggers at the portable node. This processing is continuously

performed at a given time period to update the parameters and for further online training of the neural net.

### 6.2.1 Query based nWSN data processing framework

The WSN itself acts as distributed data storage where the data can be stored and retrieved upon query requests. The query based nWSN data processing framework is laid on the data which is stored at different node levels. The dynamic data acquired by each sensor node from the surroundings can be immediately consumed by the application or it can be sent to another peer node. In QBnWSN framework, the infrastructural node disseminates a query to all the portable nodes. The query consists of several attributes.

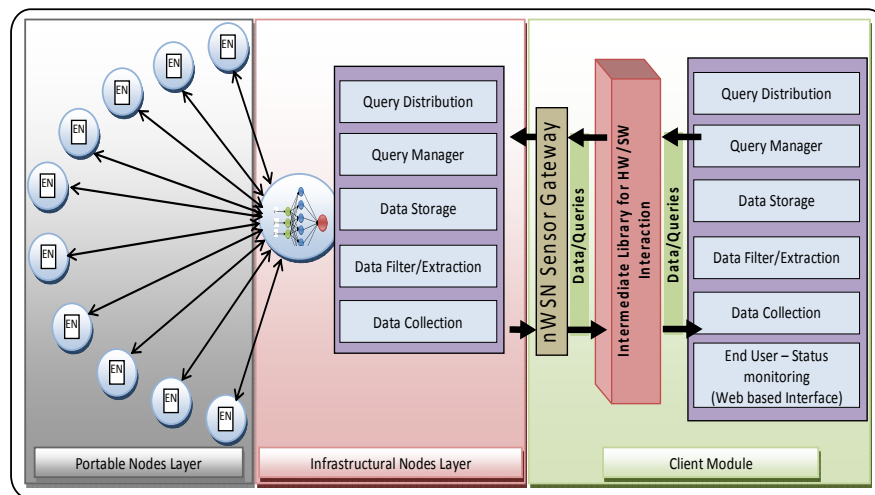


Figure 6.1 QBnWSN Framework

Each attribute has a key value, where it can modify each portable node's assembly module. Figure 6.1 shows the block diagram of the QBnWSN framework and its various components.

The framework has mainly three parts:

- 1) A client module running in a PC, which is typically a base station. This module continuously interacts with the sink to receive and send the data packets. The client side module can be used to parse queries and disseminates into the network.
- 2) An infrastructural node layer module that runs on top of the run time platform. We have used IBM's Mote Runner OS to test the model. One

infrastructural sensor node acts as a sink to communicate with the client side module.

- 3) A portable node layer module that also runs on top of the run time platform Mote Runner. Each portable node in this layer interacts with the infrastructural sensor node.

In this implementation, the distributed sensor nodes constitute a single-hop system. The cluster head/infrastructural node communicates with the end/portable sensor nodes when the data is required. We have considered a query transmission to all the portable nodes within a time interval. The period for each result produced is an epoch. This epoch duration can also be known as sample interval of the query. The sample interval is a parameter that can be changed during the experiments of the model. The infrastructural node layer consists of several components that include query distribution, query manager, data storage, data collection, data filter/extraction; an extra web interface for monitoring is available in the client module. Each component has its own contribution to coordinate with others. We considered a large packet data to be sent from each node, hence there is a requirement of data collection and filtering at the infrastructural node layer.

Every portable and infrastructural node has its own cache/buffer to accommodate data storage for each aggregation parameter. The main important attributes that are considered in this model are time, location and temperature. It can be extended for any other attribute. The data aggregation calculates for temperature at each infrastructural sensor node.

Figure 6.2 shows a simple application scenario of the model. The sink injects a query that consists of a selective message that will be executed at the specified node or at all nodes. An example of the query may be the temperature recorded at a portable node at a given time. In another example, the maximum temperature recorded at a given sub-region. Each sensor node is capable of computing various aggregation functions including, average, maximum and minimum occurrence of the attribute at the specified time slot.

In the first phase, selective messages are spread throughout the network using flooding, but these do not necessarily activate or require responses from all the nodes. In most cases the appropriate nodes are activated based on the query.

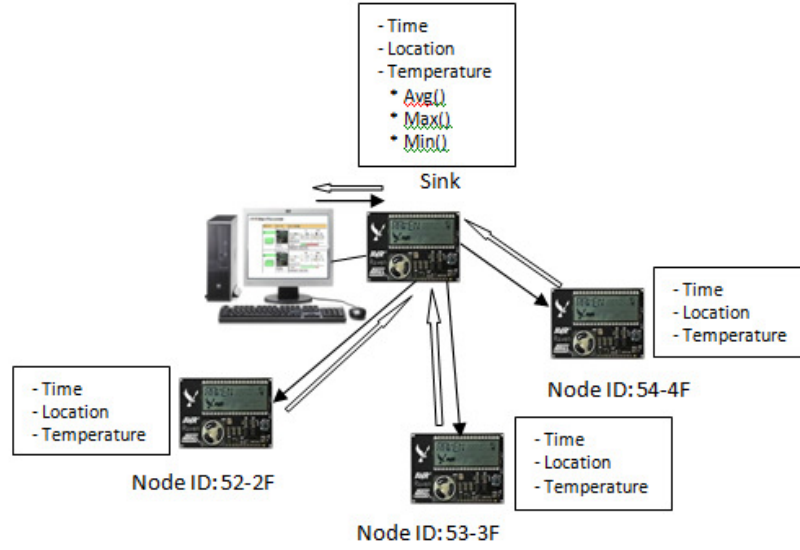


Figure 6.2 WSN Nodes arrangement in the application scenario

Let the infrastructural node  $N_i$ , where  $i$  is the node ID (short address), queries a portable node  $P_k$  to request the input training data (location and temperature) at the time  $t_l$ . If  $P_k \subset N_i$  then it returns a response to  $N_i$ . In QBnWSN framework, the sensor node has the following states.:

- 1) *Query receive state*: This state initiates when the first node of the network receives a query from the infrastructural node. The parameter of the query executes and the assembly returns true if the node belongs to the infrastructural node. It immediately sets the parameter values and calls the response state.
- 2) *Query response state*: This state initiates when the query receive state returns true. This will generate a response and will update it to the nodes local variable and add it to the data packet that transmits back to the infrastructural node.
- 3) *Data aggregation buffer state*: During this state, all the aggregation functions calculate at periodic times and update the local variable data. For every minute (or any configurable time) the mote senses the environment continuously and aggregates to update the variable data. These variable data

fills the buffer and loops within it for every 1 minute to cover for a period of 10 minutes data in the experiment. The volumetric rendered maps are constructed as performance measure to validate the QB-nWSN implementation.

### 6.3 Time Synchronization and its Implementation

In a dynamic environment the acquired sensor data at any given node varies with time. If the location of the sensor node also varies, then there is an issue with the data input to the neural net computations. This is due to the asynchronous data received from the sensor nodes to a cluster head. Hence the sensor data must be a temperature and location specific at a given time. Figure 6.3 shows the node's location change with time while the cluster head triggers its query request state. Assume there is a node  $P_1$  at location  $x_1, y_1, z_1$  at time  $t_1$ , which continuously moves with time. The query request state at the cluster head may trigger at any location before it reaches the state where its location is  $x_2, y_2, z_2$ .

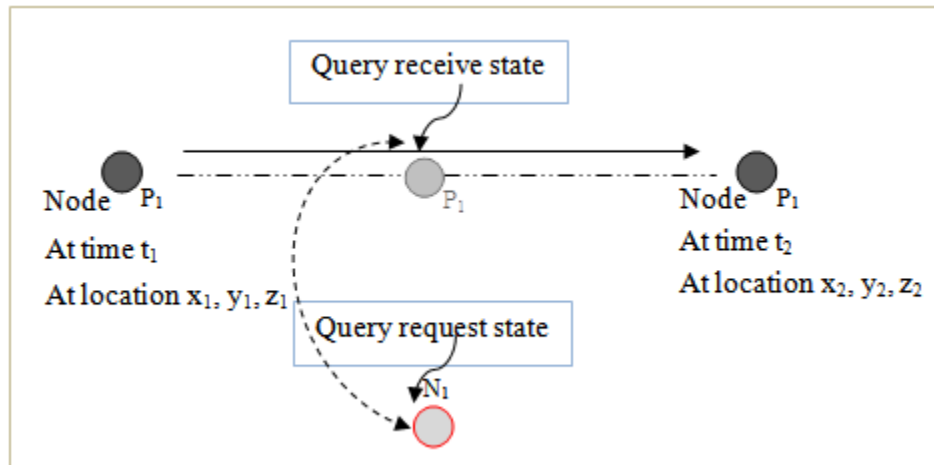


Figure 6.3 Node location change with time

The proposed system contains local variables data in byte array at each sensor node's assembly program. This buffer data is the core element to implement the time synchronization and it contains the location and temperature data. The infrastructural and portable sensor nodes have a similar buffer model. Each function can accommodate up to 10 minutes of previous data. These buffer data is looped through every one minute to update all the functional parameters. This means each

row indicates the previous minute data of the model, i.e. row 1 contains the data for the last minute and row 10 contains the data corresponding to 10 minutes earlier. The algorithm described in the earlier chapter deals the query based nWSN data processing within the sensor nodes. The QnDP algorithm executes for each epoch at infrastructural or portable node level. This depends on the queries that are passed as a selective message to the nodes. The selective message consists of few parameters, including node ID (short address) and time. The time is the target time to query the data.

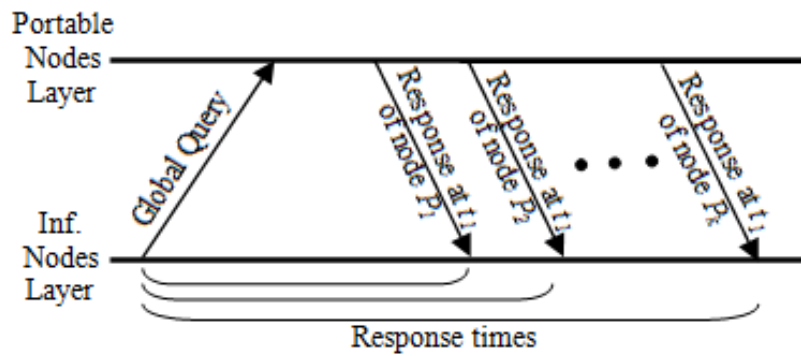


Figure 6.4 Flow diagram for query processing

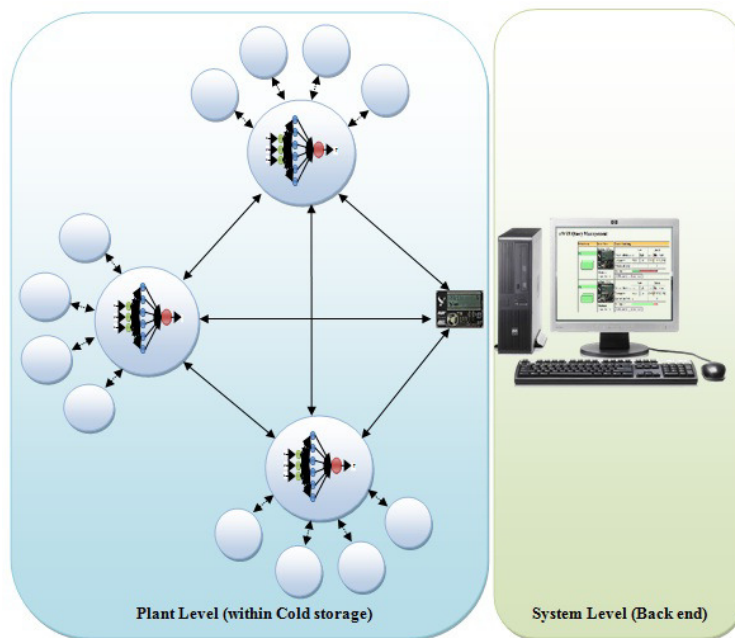
Assume there are  $k$  numbers of portable nodes where all nodes from  $P_1$  to  $P_k$  form a cluster and having a cluster head as shown in Figure 6.4. The neural net that computes temperature profile requests temperature along with its location from all portable nodes as an input data. These data from all the portable nodes have to be collected at a given time  $t_i$  in order to train the system at that time. In an example, a portable node can be attached to an object that continuously moves with time. When the infrastructural node requests all the location and temperature data at time  $t_i$ , each portable node reacts and sends the parameters data from the buffer located at  $t_i$ . In this way the nWSN algorithm within the infrastructural node trains the system at that time to update the parameters. Therefore a synchronized data set can be the input to the neural net and the online training is performed to sustain the parameters of the neural net. The flow diagram shown in Figure 6.4 describes the query flow and response between the infrastructural node and the portable node layers.

The query transmit and receive times along with the average response times are calculated as a performance measure. The volumetric rendered maps are generated

to compare the thermal precision when the data are trained synchronously versus asynchronously.

### 6.3.1 Configuration and setup for time synchronization

The importance of QBnWSN was discussed in the earlier section. The main initiative of the QnDP algorithm is to have synchronized training data available to the infrastructural nodes. This can assure the online training of the neural net at specific time periods with consistent data sets. The location and temperature data is fed to train the system.



*Figure 6.5 nWSN configuration setup*

In the experimental setup, there are four portable nodes used along with cluster head to query the data. Figure 6.5 shows the nWSN setup configuration for the experimental test bed. There are mainly two different levels that can be considered in the setup. At plant level all the sensor nodes are deployed in the given space where the temperature is monitored.

A gateway is connected to the nWSN at the system level. We have used IBM's Mote Runner to setup this scenario within the simulation environment. The socket programming functionality provided by Mote Runner can further facilitate to interact with the sensor network using the web browser. Using the interface it is



possible to send a query manually to all the portable nodes. But this could be done programmatically in any infrastructural node within a time period. For each time period, when a call back function initiates its query propagation phase, a query propagates to all the portable nodes connected to the infrastructural node. Upon receiving a query, the portable node verifies all the parameters of the query to respond accordingly. This means that a query requiring the temperature of all the portable sensor nodes within five minutes will return all the temperatures of each portable node along with its location data. This will make sure the neural net trains the system at a synchronized time. The current time of each node can be synchronized through the web interface. This will ensure the query statement time is valid at all the portable nodes.

In the experimental setup, we have fixed a timing of 10 minutes for updating the memory buffer that includes temperature, location and other aggregate functions data. During this period the queried data is trained at each infrastructural node. The sensing time is independent of updating memory buffer time and it is given one minute. Figure 6.6 shows the WSN query management results within the Mote Runner simulation environment setup. The queries transmit and receive times are recorded from the number of samples. The collected result show that an average of 25 seconds is consumed to respond to each portable node.

### Query Data Processing: QB-nWSN Framework

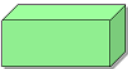

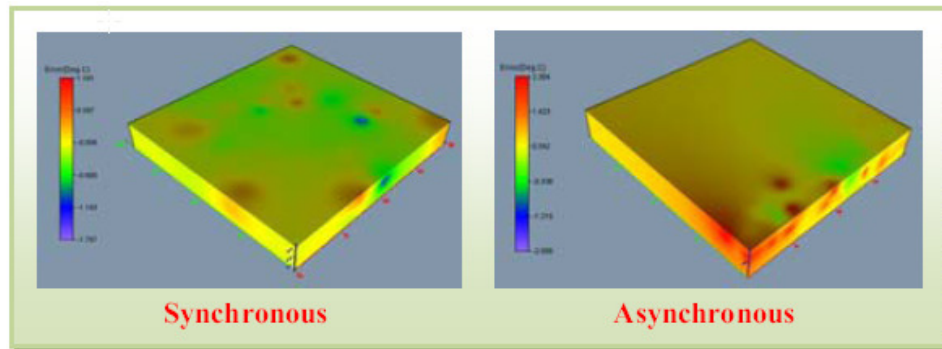
Packet Status	Sensor Node	Thermal Monitoring																																															
<div>OK</div> 	Address: -52-28  Select Function: Minimum	<table border="1"> <thead> <tr> <th></th> <th colspan="2">New</th> <th>Current</th> </tr> <tr> <th>Sensor / Thresholds</th> <th>Low</th> <th>High</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Temperature</td> <td>20.00</td> <td>30.00</td> <td>23.55 C</td> </tr> <tr> <td>Sample Rate (sec):</td> <td colspan="2">15</td> <td>15</td> </tr> <tr> <td></td> <td>Time:</td> <td>Temp:</td> <td>nWSNTemp</td> </tr> <tr> <td>Result</td> <td>0:4:4</td> <td>8.64 C</td> <td>25.72</td> </tr> <tr> <td>Location:</td> <td colspan="3">X=10 Y=15 Z=5</td> </tr> <tr> <td>Next update:</td> <td colspan="3"><div></div></td> </tr> <tr> <td>Buffer row =</td> <td colspan="3">1</td> </tr> <tr> <td>Response Time:</td> <td colspan="3">           Node1=20:2:24:863 10 15 5 23.55            Node2=20:2:25:140 20 10 5 18.59            Node3=20:2:22:596 5 20 5 12.68            Node4=20:2:22:754 5 15 5 32.74         </td> </tr> <tr> <td>Query Time:</td> <td colspan="3">Query Time=20:2:2:234</td> </tr> <tr> <td colspan="3"> <div>Send Query</div> <div>Maintenance</div> <div>QBnWSN</div> </td> </tr> </tbody> </table>		New		Current	Sensor / Thresholds	Low	High	Value	Temperature	20.00	30.00	23.55 C	Sample Rate (sec):	15		15		Time:	Temp:	nWSNTemp	Result	0:4:4	8.64 C	25.72	Location:	X=10 Y=15 Z=5			Next update:	<div></div>			Buffer row =	1			Response Time:	Node1=20:2:24:863 10 15 5 23.55 Node2=20:2:25:140 20 10 5 18.59 Node3=20:2:22:596 5 20 5 12.68 Node4=20:2:22:754 5 15 5 32.74			Query Time:	Query Time=20:2:2:234			<div>Send Query</div> <div>Maintenance</div> <div>QBnWSN</div>		
			New		Current																																												
		Sensor / Thresholds	Low	High	Value																																												
		Temperature	20.00	30.00	23.55 C																																												
		Sample Rate (sec):	15		15																																												
			Time:	Temp:	nWSNTemp																																												
		Result	0:4:4	8.64 C	25.72																																												
		Location:	X=10 Y=15 Z=5																																														
		Next update:	<div></div>																																														
		Buffer row =	1																																														
Response Time:	Node1=20:2:24:863 10 15 5 23.55 Node2=20:2:25:140 20 10 5 18.59 Node3=20:2:22:596 5 20 5 12.68 Node4=20:2:22:754 5 15 5 32.74																																																
Query Time:	Query Time=20:2:2:234																																																
<div>Send Query</div> <div>Maintenance</div> <div>QBnWSN</div>																																																	

Figure 6.6 QB-nWSN Framework results interface

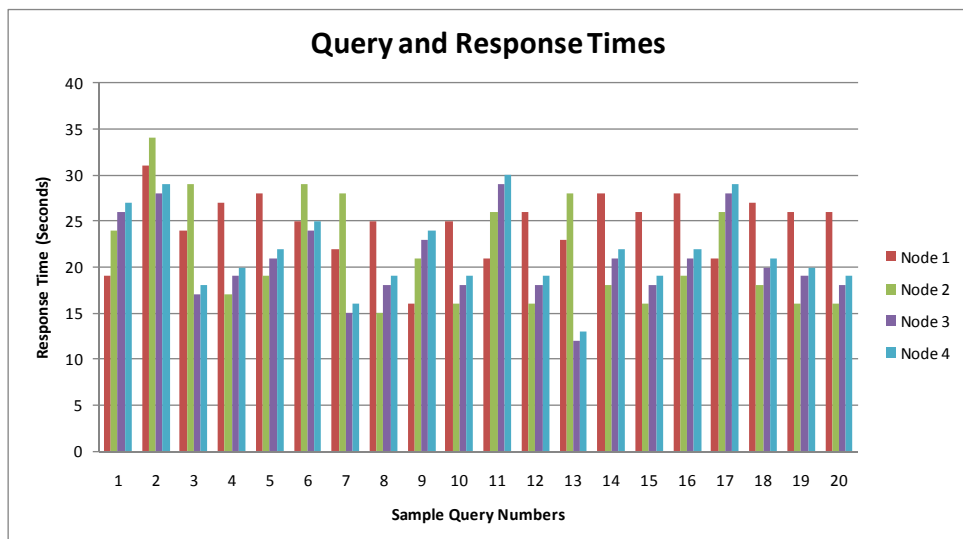
These times for the selected four nodes are given in Figure 6.8. There are 20 numbers of samples taken to evaluate the results. These response times are quite

enough to compute the neural net algorithm in any of the infrastructural nodes for the given times.



*Figure 6.7 Volumetric temperature precision*

A comparison of the thermal precision maps was also examined based on the simulation model results. Figure 6.7 shows the volumetric rendered temperature precision data when the infrastructural node receives synchronous and asynchronous data from the portable nodes. The map variation clearly represents more error when data is asynchronous. The query and sensing periods can be configured through the web interface.



*Figure 6.8 Query and response times for 4 nodes*

Increasing the number of portable nodes that can feed data to the cluster head may present challenges. However the viability of the model shows that the synchronized data can be fed to the infrastructural node to fulfil the application requirements.

## 6.4 Neural Net Cluster Dynamic Grouping

The nWSN structure has been extended based on the modularization of cluster zones. Each cluster head is responsible to the given region. These regions are constructed dynamically and hence the sensor nodes are automatically grouped as sub regions. The information sharing among these regions could be done by utilizing a Nodes Messages interaction (NMI) mode implementation. The NMI mode acts upon triggering a node for information sharing. This could be between any two cluster heads and between cluster heads and the portable sensing nodes. The messaging is divided into two typical interaction modes. These are given as NMI Pnode interaction and NMI Inode interaction.

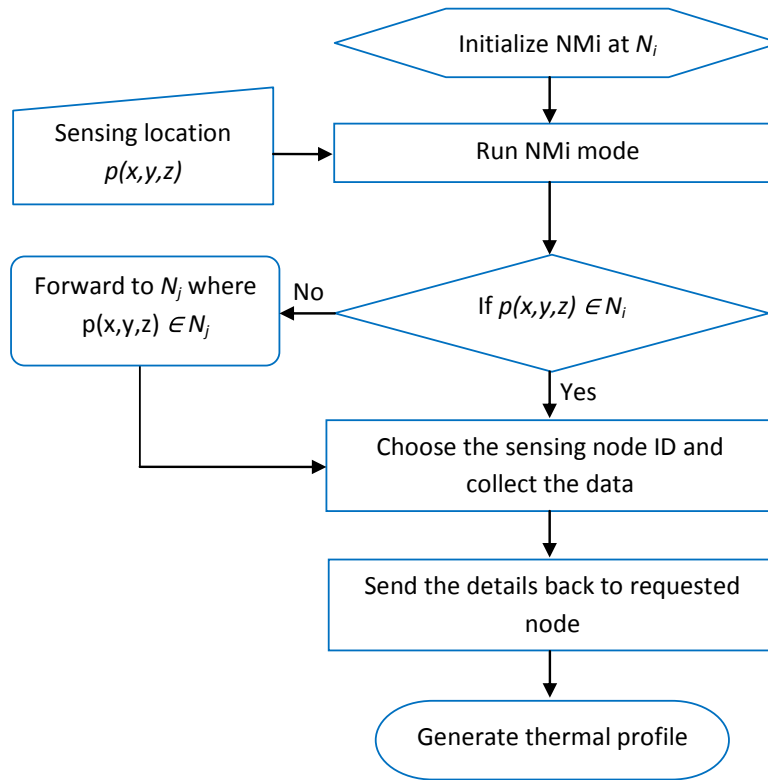


Figure 6.9 NMI Inode process flow

The core elements of the NMI are the Message Structures and Triggers in a node. Whenever a trigger is raised, it causes the relevant message structure can be activated. The NMI Inode process flow is given by Figure 6.9.

### 1) NMi Pnode interaction

In NMi Pnode interaction, a trigger is raised by the sensing node within a time interval. The sensing node transmits the information to its neighbour cluster head. This interaction may also cause the sensing node to join and leave the neighbour cluster head.

### 2) NMi Inode interaction

In this type of interaction, an automated mechanism plays an important role within the cluster head. It can raise a trigger for each time a sensing node joins the cluster head. This interaction affects cluster data collection messages for computations. The algorithm for NMi Inode initiates to run the neural net for online training of the model.

#### 6.4.1 Sequence diagram and process flow

The sequence diagram of the dynamic grouping and further sensing thermal mapping is described as shown in Figure 6.10.

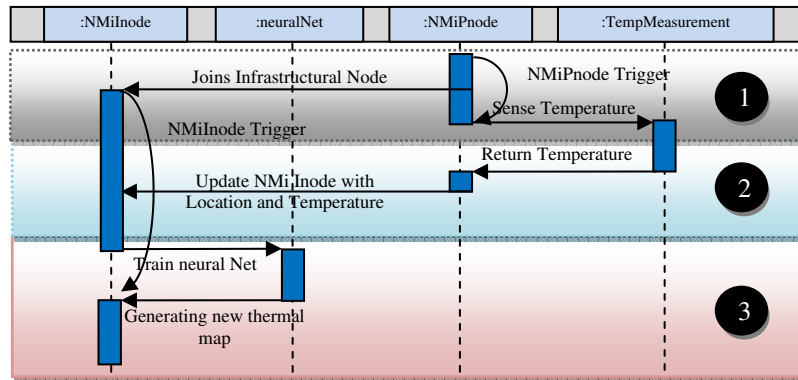


Figure 6.10 Sequence diagram of the nWSN process flow

The main classes in this part of the implementation are NMi Inode, neuralNet, NMi Pnode and TempMeasurement. The links between the elements can facilitate better understanding of the process flow for implementation. The sequence diagram contains mainly three tasks:

- 1) Grouping of sensor nodes,
- 2) Thermal sensing, and

- 3) Training the NN on the spatial thermal map.

These tasks execute continuously in the sequential order upon identifying a sensor node in the group. Due to the NMi mode interactions the sensing nodes join and leave the nearest cluster head when moving within the given space. The location and temperature data is updated with NMi Inode to train the neural net which further generates a new thermal map. The implementation of the cluster dynamic grouping has been proved and supported by a case study in the upcoming chapter. The space partitioning and sub region analysis has been conducted by looking the thermal precision in terms of MAE, RMSE as performance measures.

## 6.5 Minimizing Nodes Approach

The nodes minimization approach follows by constructing a model where it is considered that  $k$  is a dynamic element as it varies from node to node. We have used the  $k$ -NNA and Bayesian methods to identify the number of nodes can be deployed for a given accuracy. Each node in the model has its own  $k$  nearest neighbours calculated by that node. The selection process to decide the  $k$  nearest neighbours is described in the next section. In  $k$ -NNA model, there are two inputs required:

- 1) The number of neighbours to be considered -  $k$ , and
- 2) Total number of sensor nodes in the space.

In the simulation environment, a basic reusable object (a sensor node) is designed and customized. This sensor node acts as an intelligent agent in the model. The following assumptions have been taken into consideration while deploying the nodes into the model:

- 1) The locations of each node deployed in the simulation environment know the coordinates  $(x, y, z)$  of its location,
- 2) The nodes can be arranged on one layer at specified grid points or randomly distributed, and
- 3) Environmental dynamics are not included and the space is considered homogeneous.

The simulation environment can facilitate the deployment of a number  $N$  of nodes. The basic building block of the model is the sensor node, which can be replicated  $N$  times within the boundary of the specified region  $\mathbb{R}$ . During the node generation, the simulation environment will set an attribute value of the node's location and temperature. It is assumed that a uniform distribution of the room temperature values ranging between  $T_{min}$  and  $T_{max}$  and the location values between  $(x_0, y_0, z_0)$  and  $(x_b, y_m, z_n)$  will be generated and assigned. We have designed an approach to minimize the number of nodes that would be required to map the given space. These results are discussed in the next chapter.

### 6.5.1 *Sequential search and nodes minimization*

The nodes minimization process follows a sequential search from the first node in the tree. After nodes deployment, it can be chosen to minimize the model for reducing the node count while keeping the temperature distribution throughout the volume within the given precision (i.e. threshold value). The flowchart shown in Figure 6.11 explains the various steps involved for implementation. This process can take place by deactivating each node and immediately queried at the same location to get the predicted temperature based on its neighbour nodes.

The predicted value at that location will then be compared to the actual node temperature. Obviously there is an error, which is equivalent to  $\varepsilon$  (the difference between the actual and predicted temperatures). This error is then compared to the threshold value and if the error is beyond the specified tolerance, then the selected node will be flagged with a value '1' in a built-in table named 'A' (the nodes that can't be redundant).

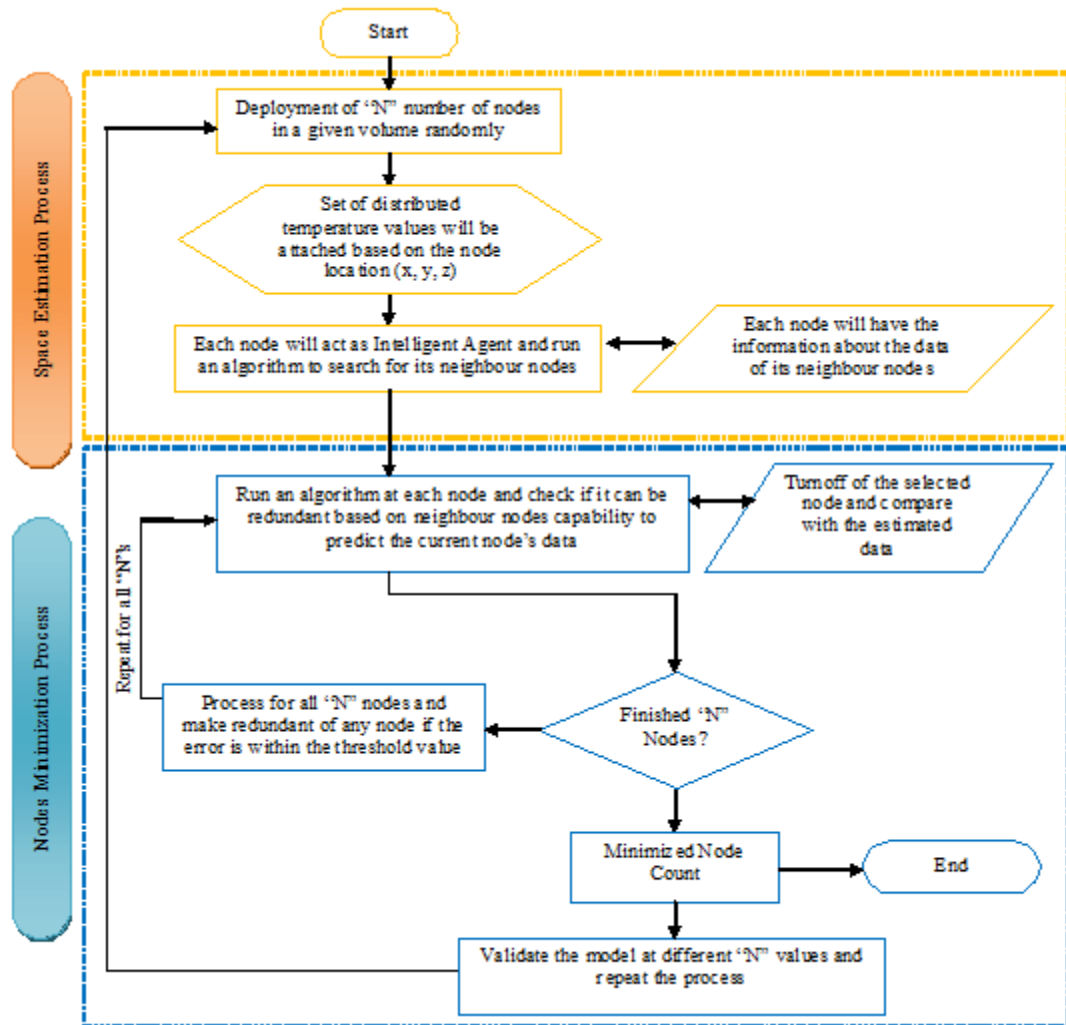


Figure 6.11 Minimizing nodes approach flowchart

If the  $\varepsilon$  is within the threshold value, the selected node will be flagged with a value '0' in the table 'A' (the nodes that can be redundant). Considering all the nodes in a tree followed by a starting node will make a hierarchy shown in the Figure 6.12. The table will have the nodes data containing flags '1' and '0', which further summed up and distinguished the influences and dependability of each node.

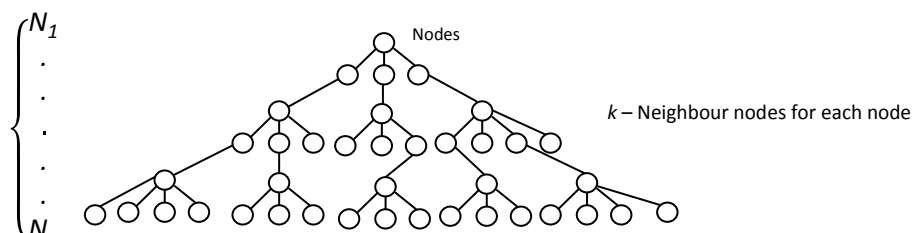


Figure 6.12 Nodes hierarchy

The probabilities of each node on its neighbouring node are calculated to identify the influence of a given node. We have applied a Bayesian theory (belief theory) to identify the nodes that have more influence on each other. The redundant nodes have been eliminated after applying the Bayesian classification. Many real world applications deal with uncertain knowledge and input data that is insufficient to make a decision. In addressing such issues of uncertain information, classical probability theory has been found to be very useful. Hence, a Bayesian network is a probabilistic representation for uncertain relationships and is useful for modelling such real world problems.

#### 6.5.2 Bayesian approach to identify more mutually influenced nodes

A Bayesian network encodes a joint probability distribution over a set of random variables that expresses the belief regarding how likely the different predictions are in order to quantify uncertainty in inferences. A Bayesian network (B), given a set of variables  $X=\{X_1, X_2...X_n\}$  are the discrete variables (nodes). Assume that the node  $X_j$  is the child (neighbour) of the node  $X_i$ , which means  $X_i \rightarrow X_j$ . The conditional probability can be calculated by utilizing the fundamental formula as in equation 6.1.

$$P(X_i/X_j) = \frac{P(X_i, X_j)}{P(X_j)} \quad 6.1$$

For individual probabilities, the number of occurrences of a state variable (1 or 0) can be counted. Let  $n_{ij}$  be the number of occurrences of the state  $j$  of the  $i^{th}$  variable in the table and  $n$  is the total number of data cases from the table. Using these frequency values, we can calculate the probabilities by using the equation 6.2.

$$P(X_i = x_j) = \frac{n(X_i = x_j)}{n} = \frac{n_{ij}}{n} \quad 6.2$$

Thus the conditional probabilities can be calculated by using the individual probabilities in equation 6.2. The conditional probability  $P(X_i \rightarrow X_j)$  can be obtained as in the following equations.



$$P(X_i, X_j) = \frac{n(X_i, X_j)}{n(X_j)} \quad 6.3$$

$$P(X_j) = \frac{n(X_j)}{n} \quad 6.4$$

By substituting equations 6.3 and 6.4 into equation 6.1 we get,

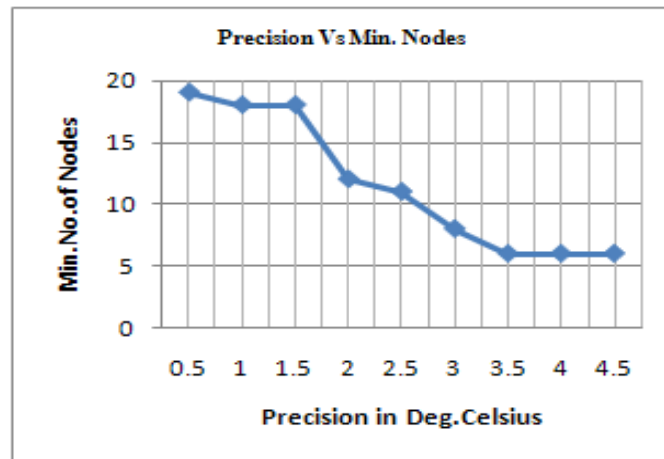
$$P(X_i/X_j) = \frac{n(X_i, X_j)}{n(X_j)} \quad 6.5$$

The resulting equation 6.5 can be used to calculate the conditional probability by counting the corresponding frequencies or influences of the nodes to each of its neighbours. After evaluating the probabilities, the nodes with higher probability will be more vital. The group of nodes with higher probability will be the subset of the group of lower probability nodes. The final count of all the vital nodes can't be redundant. From these nodes, the minimum number of nodes can be estimated, along with their location measured over the full space based on the accuracy discrepancy. This model is further examined by changing the value  $N$  and repeats the process to see if the algorithm is valid.

### 6.5.3 Nodes minimization simulation results and discussion

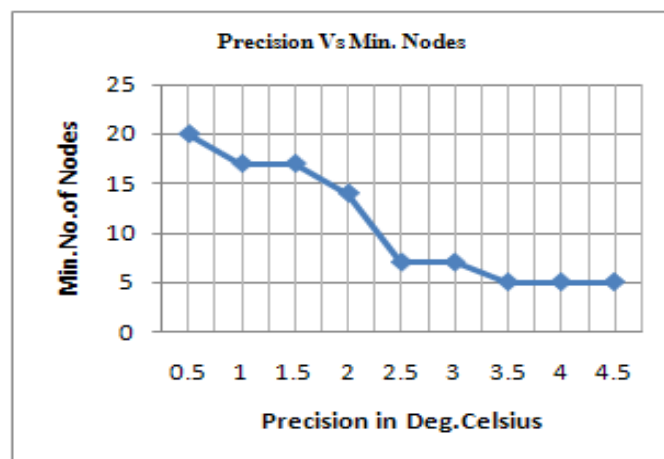
We have considered an algorithmic based approach using Bayesian theory to identify the minimum number of nodes required by keeping the thermal prediction error within the given range. The algorithmic approach was described in chapter four.

The nodes are deployed in a  $50 \times 50 \times 10 \text{ m}^3$  volume of space. This size can be varied in the user interface control. The charts in Figure 6.13 and Figure 6.14 show precision versus minimum number of nodes required when deploying 50 and 100 nodes respectively. It is observed that the minimum number of nodes resulted from the model is same when 50 and 100 nodes are deployed. The model parameters can be submitted to the GUI control for experimentation. The code has been written in C++ for the algorithm development. The simulation environment can be customized to suit, where the nodes can be deployed for further examination of the given space.



*Figure 6.13 Precision Vs Nodes (at 50)*

The model is tested by varying the density of the nodes to recognize the current methodology for its accuracy. Irrespective of the number of deployed nodes to conduct the experiment, a minimum of 20 nodes are required to get at least 0.5°C of precision to map the space based on the configuration. A number of nodes greater than 20 can lead to further increase the precision. But this should be dependent on the spatial distribution of the nodes, size of the space and also the thermal profile. The nodes minimization and their location are important for any application domain, and it is considered for study further. The optimization of nodes can resolve the minimum number of nodes and their location information by considering the aspects including the room space, location constraints with an object function to minimize the prediction accuracy.



*Figure 6.14 Precision Vs Nodes (at 100)*

# Chapter 7

## 7 Cool Storage in a Meat Plant: A Case Study

New Zealand is a world leader in farming for lamb and beef production. The New Zealand economy derives \$3.8 billion from meat exports. The meat companies have to maintain high quality to meet the market demands. The temperature variation in cool store has a significant effect on meat tenderness, colour and on the microbial status of the meat. Hence the thermal mapping during the chilling process and further shipment in real time is very vital.

### 7.1 Introduction

In a dynamic environment, the temperature varies with time and that reflects the characteristic within the meat. The real time monitoring of the meat temperature gradient will further assist to predict the food quality. Figure 7.1 shows the temperature variation of beef at its surface, middle and core within 24hrs time in cool store [111].

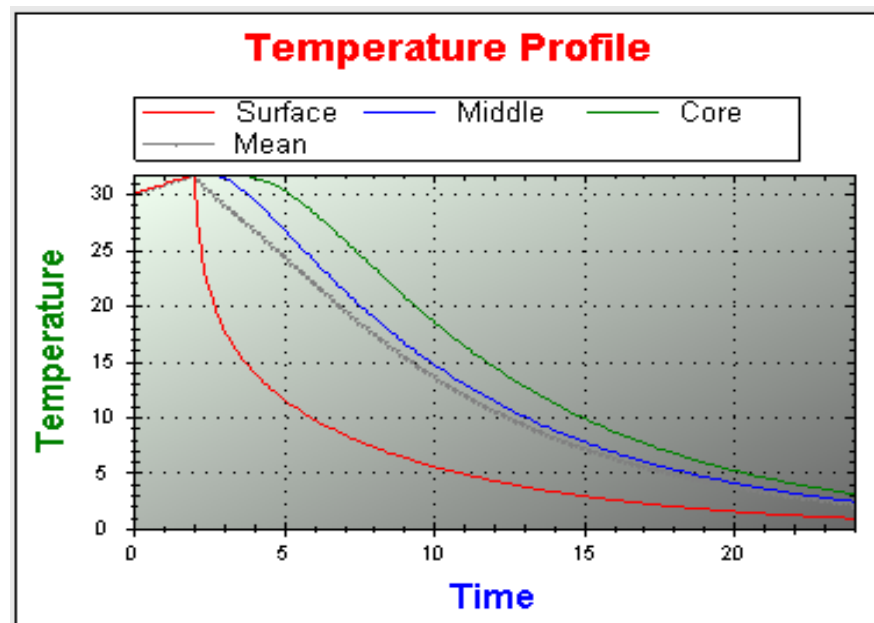
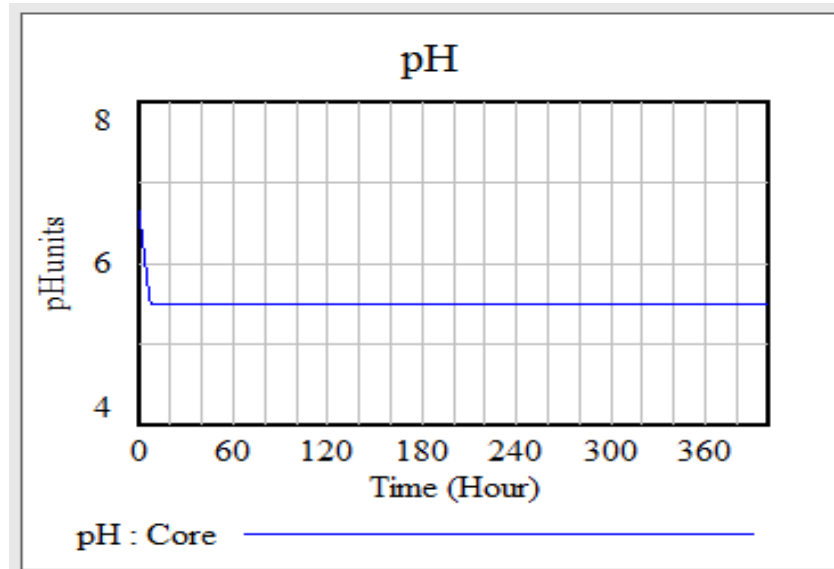
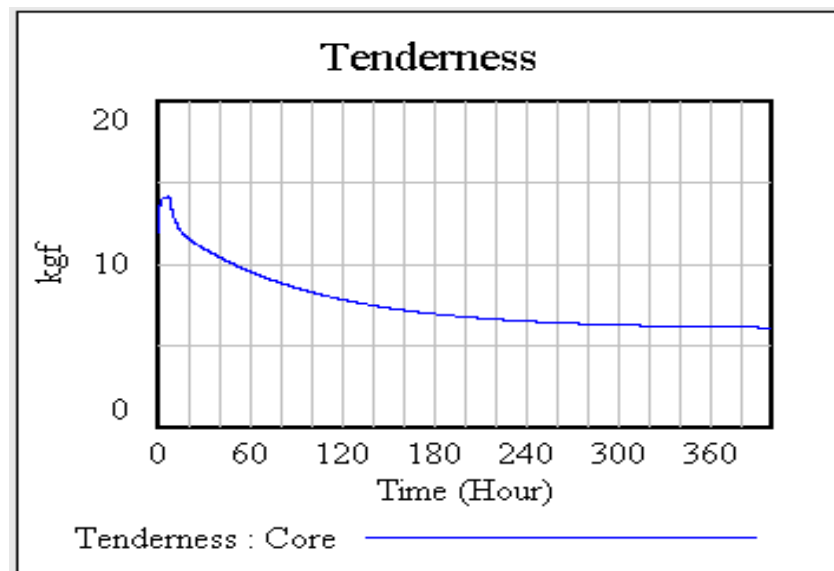


Figure 7.1 Thermal profile of a beef within the first 24hrs

The meat pH and tenderness are also affected with the temperature variation as shown in Figure 7.2 and Figure 7.3 [111]. This reflects the importance of monitoring the thermal exposure history when precision is required in identifying the quality status.



*Figure 7.2 pH variation of a beef carcass*

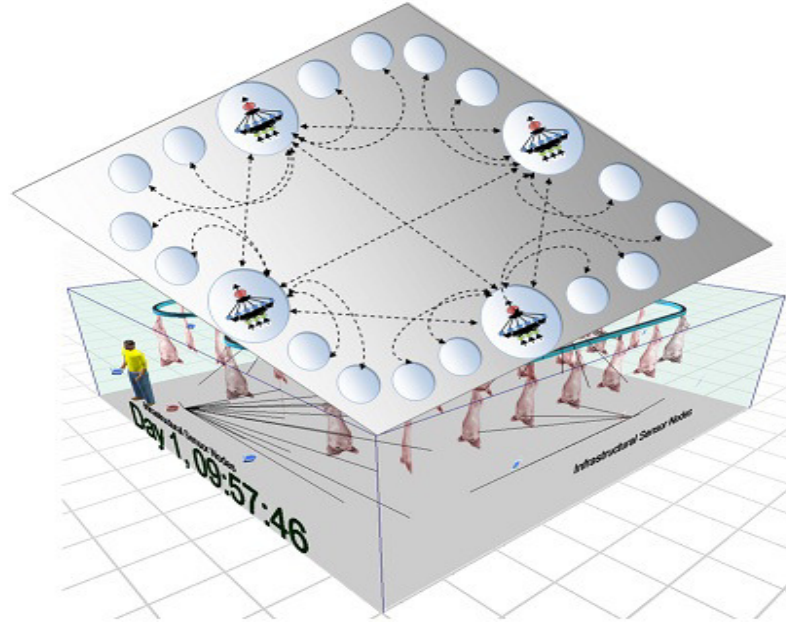


*Figure 7.3 Tenderness variation of a beef carcass*

## 7.2 Modeling Scenario in a Cool Storage

In this modeling scenario, the carcass hangers are used as sensor holder. Each sensor node (a portable node) is attached to a carcass for measuring the nearby

environmental temperature while it is transported on an overhead conveyor. The sensor nodes feed temperature data to their cluster head (one of the infrastructural nodes) for processing.



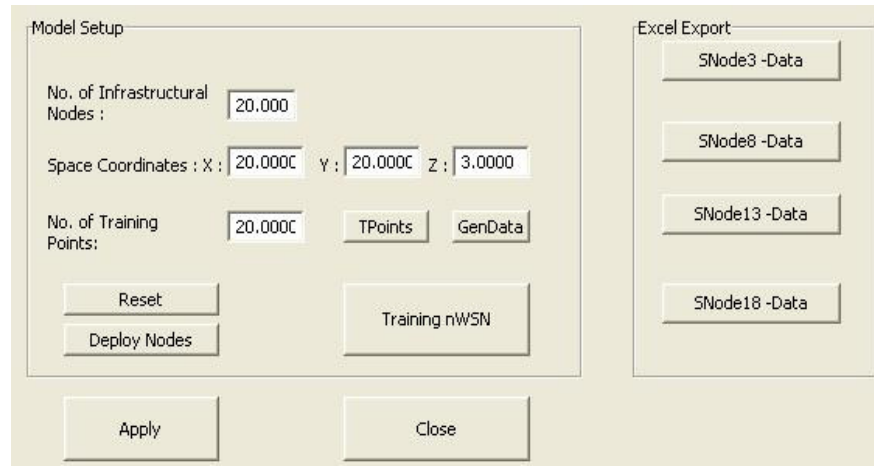
*Figure 7.4 Schematic diagram of the cool storage*

The neural net algorithm is embedded in each cluster head to compute the thermal map. Figure 7.4 shows the schematic diagram of a cold storage for nWSN. The top layer shows the infrastructural and portable nodes placement in a given space. The surface temperature of the carcass in a cold storage changes from 30°C to 4°C within a period of five hours as shown in Figure 7.1. This is a rapid change at the beginning of the process before it comes to a steady state. The temperature monitoring during this period is vital for overall coverage of the space and to identify the hotspots.

### **7.3 Simulation environment setup and experimentation**

A model is designed within a discrete event simulation environment to test the nWSN architecture using the NMi model. For the experiment, we assume a cold storage where the temperature distributed with minimum and maximum ranges between -2°C to 8°C. A dynamic behaviour is introduced within the modelling environment by varying the temperature profile at all points between  $\pm 2^\circ\text{C}$  with the simulation time. There are 174 beef carcasses used in the model. A simulation

model is constructed with a space volume  $20 \times 20 \times 3 \text{ m}^3$ . A GUI is built to ease the use of the experimental setup as shown in Figure 7.5.



*Figure 7.5 GUI to setup experimentation*

Four different scenarios have been analysed. The subsections are divided as follows. A dynamic temperature of  $\pm 2^\circ\text{C}$  is used for sections A, B, C and the section D is used with an increased temperature to  $\pm 4^\circ\text{C}$ .

- 1) *Section A*: The prediction of the temperature for better coverage within the confined space by placing sensor nodes in three layers. Based on the methodology described in chapter 4, as shown in Figure 4.8, the space is divided into a number of vertical layers. The three layers are divided for better coverage to place the sensor nodes.
- 2) *Section B*: The temperature variation analysis at the surface of the each subspace. The infrastructural nodes placed on the inner surface of the space can cover a subspace described in Figure 4.10. The overlap among the subspaces is analysed to study the variation of temperature at the surface of each subspace.
- 3) *Section C*: The prediction analysis where the number of carcasses is reduced to 86, while all other parameters are unchanged. In this scenario, the inter-arrival time of each carcass is increased, hence reflected into a reduced number of carcasses to 86 to fill the given space on the overhead conveyor.
- 4) *Section D*: Thermal mapping of the space where the dynamic temperature behaviour is taken between  $\pm 4^\circ\text{C}$ . In this section, the temperature fluctuation is increased to  $\pm 4^\circ\text{C}$  per hour. This is due to test the feasibility of the nWSN, where the application involves a rapid fluctuation in temperature.

These different scenarios have been constructed to study the viability of the model.

The estimated time to fill the carcasses in the meat cool store is about 5 hours for 174 and same for 86 but the inter-arrival time is increased from 120 seconds to 240 seconds for the given 180 meters of the overhead conveyor.

### 7.3.1 (A) Predicting the temperature for better coverage by placing nodes in three layers.

All the parameters of the model can be configured using the GUI including deploying the infrastructural nodes at specific locations around the wall. The temperature fluctuation is considered between  $\pm 2^{\circ}\text{C}$  with the simulation time. The temperature fluctuation is assumed as a sinusoidal form of cycling with a period of 1 hour.

The space can be fully occupied with 174 carcasses which are equally distributed and hanging on the overhead conveyor. The conveyor moves the carcasses with a constant speed. The schematic diagram of the conveyor arrangement within the cold storage is given by Figure 7.6. The infrastructural nodes 3, 8, 13 and 18 are placed in the middle of the four side walls of the room. Each carcass that entered the cool storage is attached to a portable node. The interaction between the carcass node and infrastructural node is essential for effective coverage of operational requirement of nWSN.

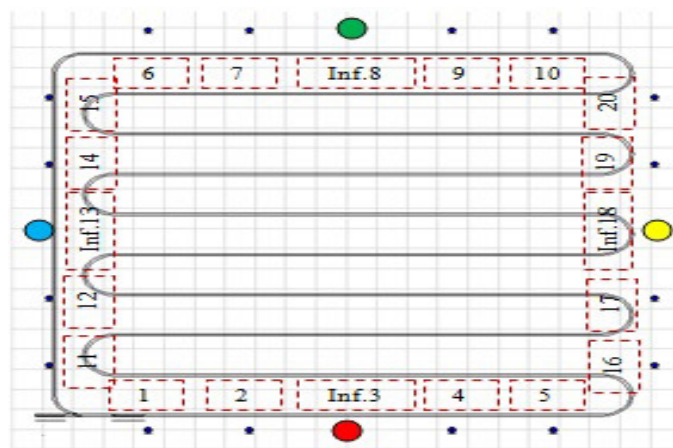
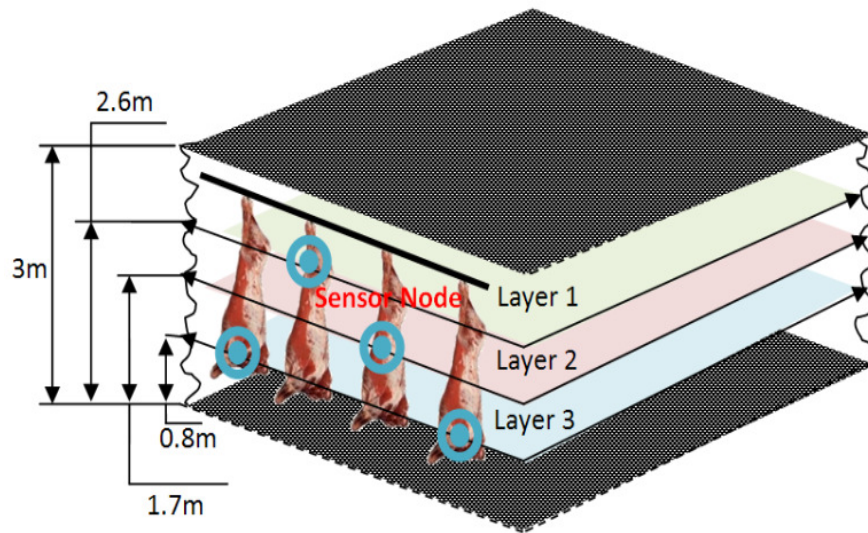


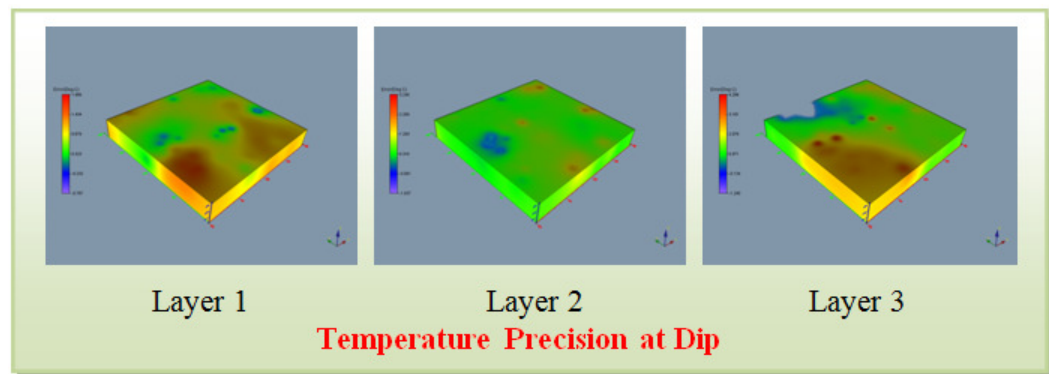
Figure 7.6 Schematic arrangement of infrastructural sensor nodes

The carcass node joins the nearest infrastructural node for information sharing as soon as it enters that particular infrastructural node's domain. The NMi modes act upon joining of any portable node into any infrastructural node as a cluster head.



*Figure 7.7 Portable node layers within the cool store*

The portable nodes are placed on a carcass where they can form three layers (Layer 1, Layer 2 and Layer 3) as shown in Figure 7.7. This placement strategy has been chosen to identify the better coverage of the overall space for thermal tracking. While the carcass is moving from one location to the other on the overhead conveyor, the portable node joins and leaves the infrastructural nodes one after the other. The model continuously drives the nWSN to train the system. In order to test the performance assessment of the nWSN architecture, an MAE is measured. The minimum statistical measurement of error reflects the viability of the model. Hence the estimated temperature and actual temperature at any arbitrary points within the space are measured. The temperature precision data is collected at two different conditions of the thermal profile. The profile at Dip and Peak conditions are considered when collecting three layer nodes temperature data. The volumetric rendered temperature precision data is shown in Figure 7.8 for the Layer 1, Layer 2 and Layer 3 when the temperature profile level went down (dip).



*Figure 7.8 Volumetric temperature precision at dip*



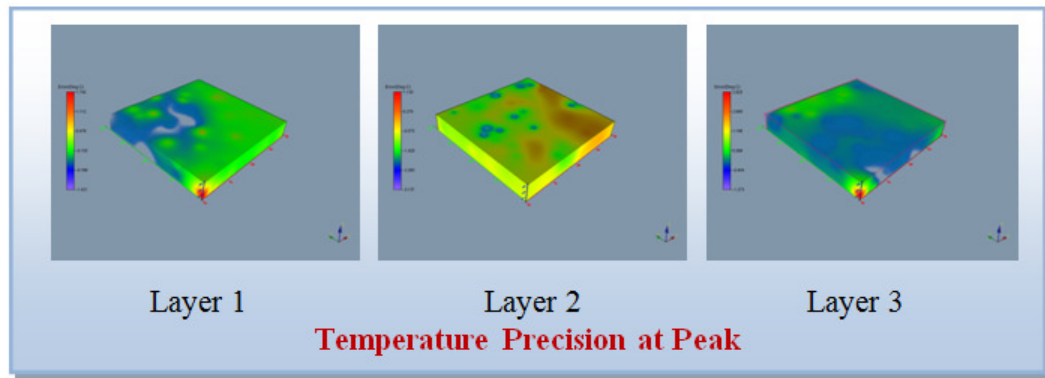


Figure 7.9 Volumetric temperature precision at peak

Figure 7.9 shows the volumetric rendered temperature precision data for Layer 1, Layer 2 and Layer 3 when the temperature profile level is up (peak). It can be easily distinguished by looking at these two volumetric maps that the Layer 2 nodes have a greater coverage than the Layer 1 and Layer 3. The temperature profile of the cold storage within the simulation environment is the actual temperature and the estimated temperature is calculated from the nWSN. Arbitrary testing points are considered to verify the model at infrastructural nodes 3, 8, 13 and 18.

	MAE at Peak			MAE at Dip		
Node	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3
SNode3	-0.02764	-0.27374	-0.19867	-0.6174	-0.09339	-0.15047
SNode8	-0.58955	0.016004	0.361476	0.025936	-0.02975	0.025783
SNode13	-0.61867	0.135717	-0.01649	-0.55477	-0.03517	0.393174
SNode18	-0.0564	0.061493	-0.00487	-0.05323	0.010505	0.190446

Figure 7.10 MAE at peak and dip for Inf. sensor nodes

Figure 7.10 represents the MAE at Peak and Dip for infrastructural sensor nodes 3, 8, 13 and 18. The temperature precision is higher at the layer 2 for both peak and dip. The MAE of  $\pm 0.5^{\circ}\text{C}$  is observed among all the infrastructural nodes. This clearly states that the nWSN model prediction for temperature variation in a dynamic environment is suitable for meat industry applications. However, the feasibility of the nWSN using the NMi model would raise further more challenging issues to be tackled in the future work. A maximum error presents at node 8 and node 13 at peak in addition to node 3 and node 13 at dip as shown in Figure 7.11 and Figure 7.12.

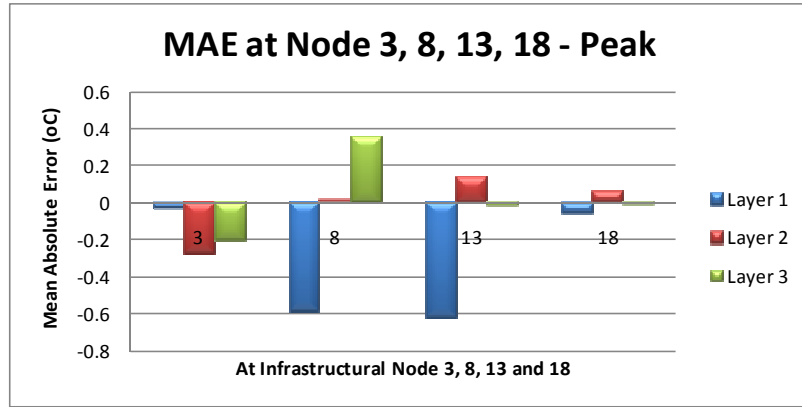


Figure 7.11 MAE at nodes 3, 8, 13, 18 at peak

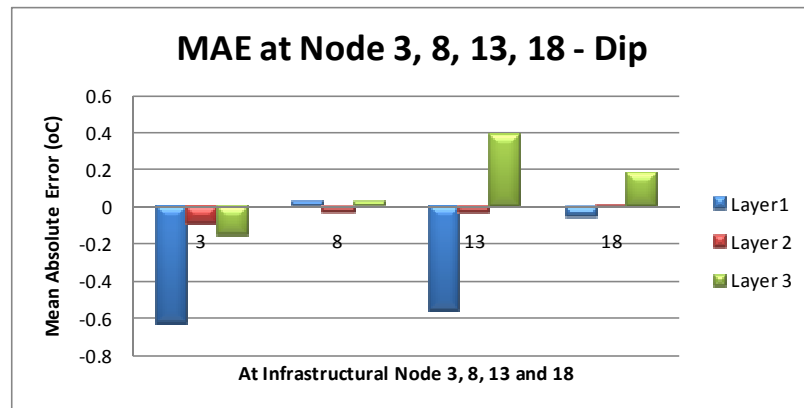


Figure 7.12 MAE at nodes 3, 8, 13, 18 at dip

This is due to the overhead conveyor design and flow direction of the carcasses. The conveyor layout factor also affects the error, since the number of carcasses that join and leave the infrastructural nodes varies. On the other hand, the location of the infrastructural nodes also affects the MAE of the model. These experiments identify future directions where nWSN with NMi model can be deployed for real time measurements.

### 7.3.2 (B) Thermal analysis at the subspace surface region.

We have examined the thermal variability of the surface of each subspace in each cluster head discussed in the chapter 4. The experimental results show that the MAE is considerably high at these regions for both peak and dip conditions as shown in Figure 7.13.

	MAE at Peak			MAE at Dip		
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3
<b>SNode3</b>	1.060	1.070	0.800	-1.600	0.140	-0.420
<b>SNode8</b>	1.160	0.840	0.980	-1.560	-1.430	0.560
<b>SNode13</b>	1.230	1.020	0.680	-1.570	0.170	-0.200
<b>SNode18</b>	1.410	0.460	0.820	-1.350	-0.260	-1.110

Figure 7.13 MAE at peak and dip for Inf. sensor nodes

The MAE of  $\pm 1.5^{\circ}\text{C}$  is observed among all the infrastructural nodes. The maximum error presents at node 3, 8 and 13 at peak and dip as shown in Figure 7.14 and Figure 7.15.

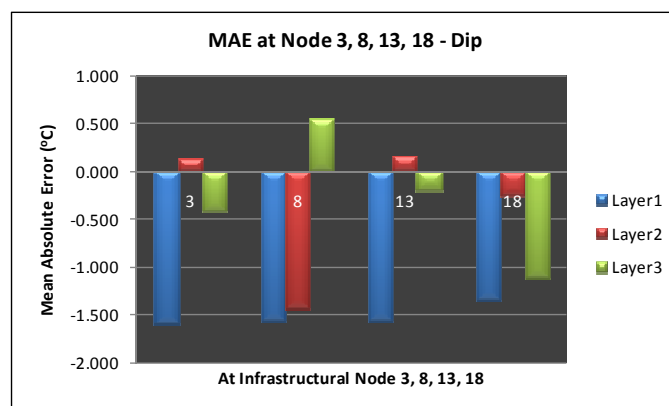


Figure 7.14 MAE at nodes 3, 8, 13, 18 at peak

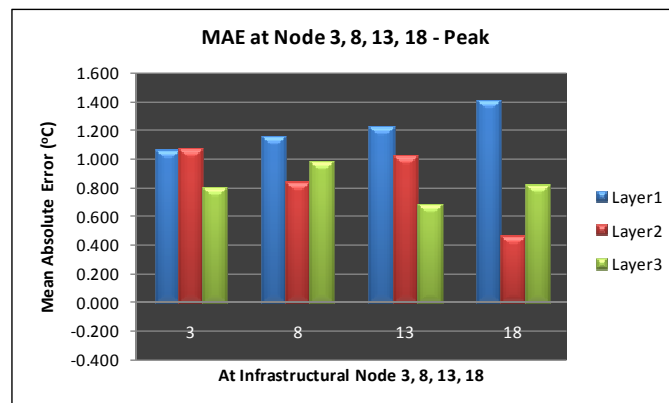
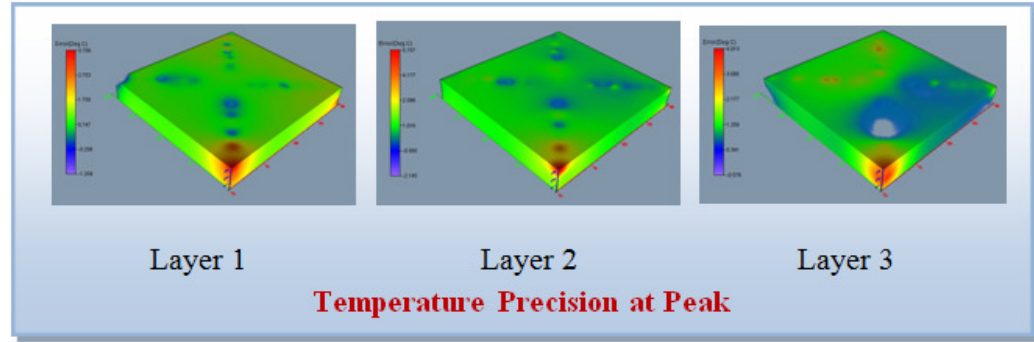
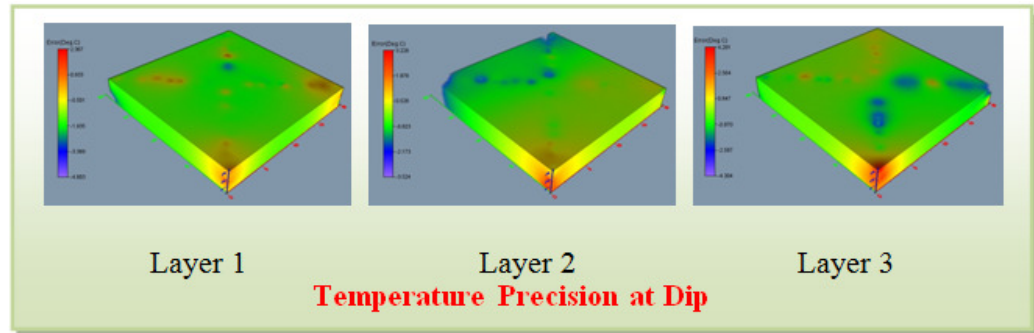


Figure 7.15 MAE at nodes 3, 8, 13, 18 at dip

The volumetric rendered temperature precision data is shown in Figure 7.16 for Layer 1, Layer 2 and Layer 3 when the temperature profile level went up. Figure 7.17 shows the volumetric rendered temperature precision data for Layer 1, Layer 2 and Layer 3 when the temperature profile level went down.



*Figure 7.16 Volumetric temperature precision at peak*

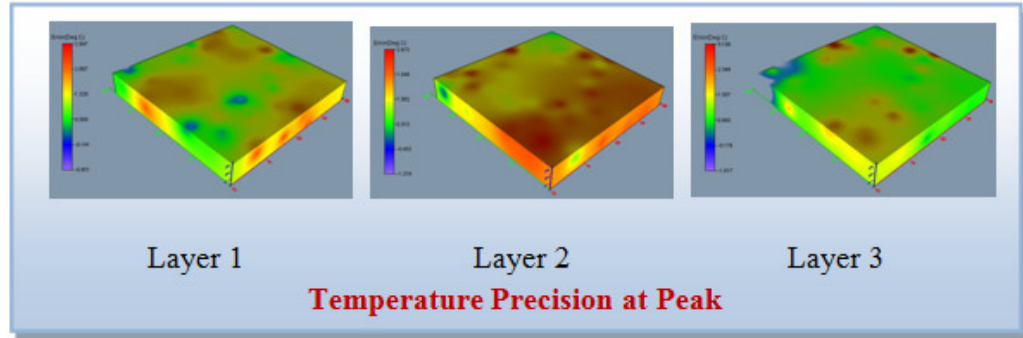


*Figure 7.17 Volumetric temperature precision at dip*

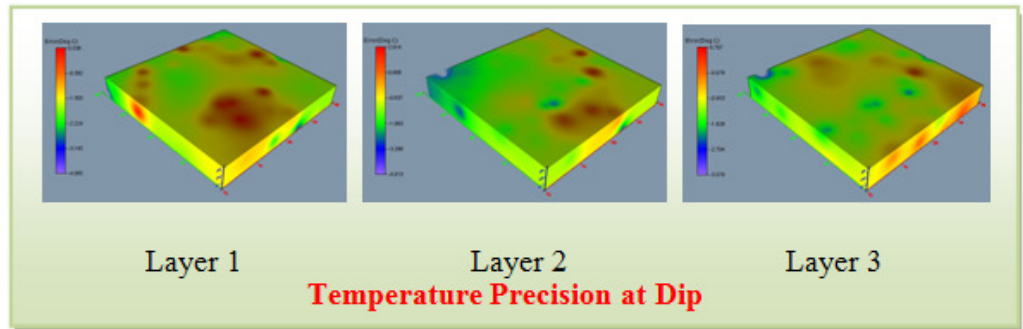
It can be easily distinguished by comparing the maps that there is an error around the subspaces. Hence the integration of multiple subspaces needs to be addressed to map the thermal influence precisely.

### 7.3.3 (C) Thermal analysis at increased carcass inter-arrival time. i.e. Carcass count reduced to 86.

In this scenario the inter-arrival rate of the carcass is increased, as a result of which, the carcass count is reduced to 86 in the cool storage. Therefore, all the carcasses are evenly distributed in the plant. All other parameters are similar to section (A). Three sensor node layers are considered in the model. The experimental result shows that there is a greater increase in MAE for both peak and dip conditions. The outcome clearly identifies the insufficient training data set for the neural net. The Figure 7.18 and Figure 7.19 shows volumetric rendered temperature precision data for Layer 1, Layer 2 and Layer 3 at peak and dip, respectively.



*Figure 7.18 Volumetric temperature precision at peak*

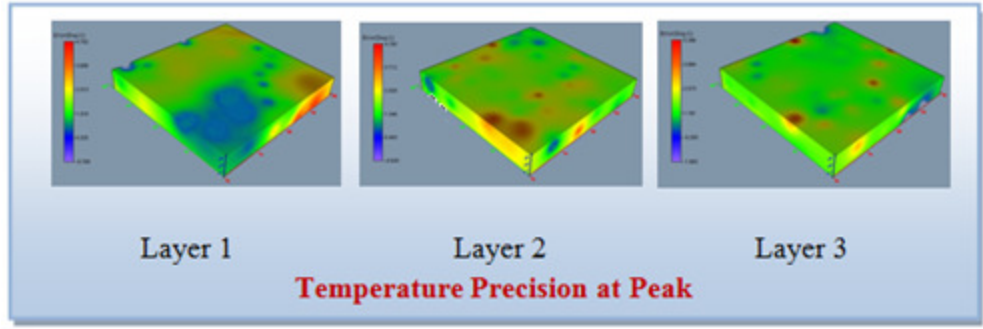


*Figure 7.19 Volumetric temperature precision at dip*

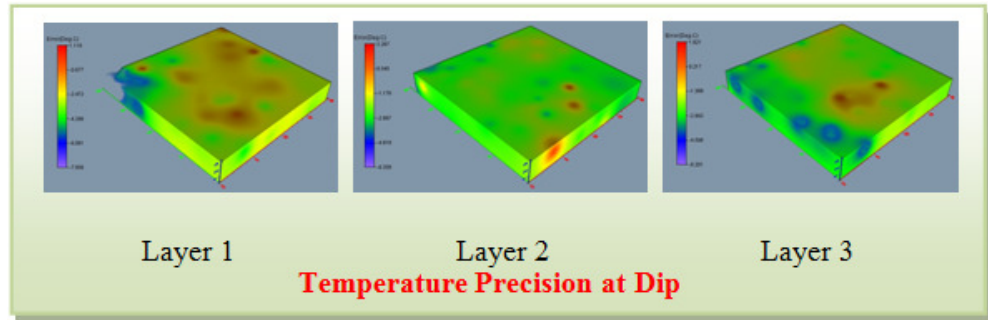
A MAE of  $\pm 2^{\circ}\text{C}$  is observed and Layer 2 has recorded more errors than other layers. Hence, it is important to consider and re-evaluate the algorithm in these conditions. Evaluating other neural net architectures for these types of conditions especially where there is less training data available may be worth considering. Future studies need to address these challenges.

#### 7.3.4 (D) Thermal mapping of the space where the temperature fluctuation is taken between $\pm 4^{\circ}\text{C}$ .

The dynamic behaviour within the cool storage is introduced by fluctuation of the temperature between  $\pm 4^{\circ}\text{C}$  with the simulation time. In all the scenarios we have a temperature fluctuation of  $\pm 2^{\circ}\text{C}$ . In this scenario, we deliberately introduced more rapid temperature change to study the behaviour of the model. However, a temperature change of  $\pm 4^{\circ}\text{C}$  in a one hour period may not be a common situation in a cold storage environment.



*Figure 7.20 Volumetric temperature precision at peak*



*Figure 7.21 Volumetric temperature precision at dip*

Nevertheless, in our study, it was an interesting task as it may reflect the viability of the neural net for some kind of applications. The experimental results show that there is a rapid increase of MAE for both peak and dip conditions. Figure 7.20 and Figure 7.21 show that the volumetric rendered temperature precision data for Layer 1, Layer 2 and Layer 3 at peak and dip, respectively. There is an MAE of  $\pm 4^{\circ}\text{C}$  is observed at node 13 and 18 at dip and peak, respectively. Figure 7.22 and Figure 7.23 show the variation of MAE between the infrastructural nodes 3, 8, 13 and 18 at Layer 1, Layer 2 and Layer 3.

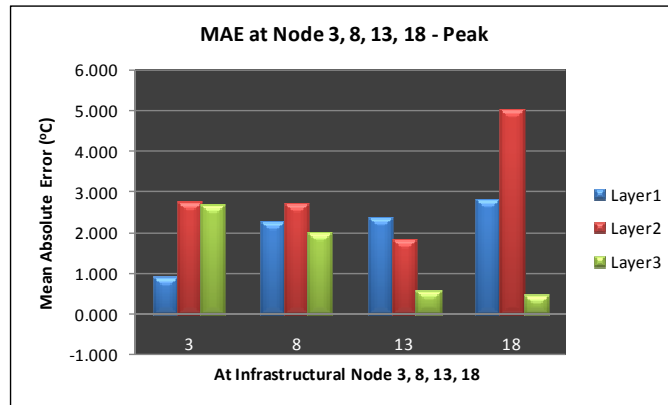


Figure 7.22 MAE at nodes 3, 8, 13, 18 at peak

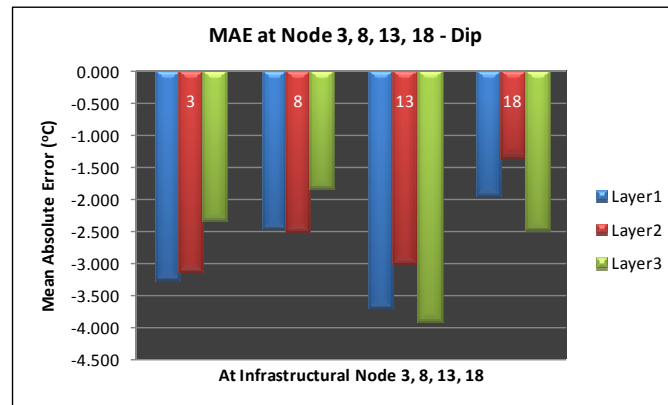


Figure 7.23 MAE at nodes 3, 8, 13, 18 at dip

This variation clearly identifies that the system is unstable in conditions of very rapid temperature changes. The temperature at any arbitrary points varies rapidly and hence the prediction error reflects the same. The mean temperature variation is very high between any two points within the space. Using WSN to respond to a very rapid dynamic environment with the nWSN architecture may be challenging and will need to be addressed in future studies.

# Chapter 8

## 8 Conclusions and Future Directions

This chapter concludes the work that has been achieved by the research objectives specified in the first chapter. We have defined the motivation of the current research work, contributions and the approach in the same chapter. This chapter concludes the overall work which includes the various scenarios built to analyse the results based on the methodologies implemented. Future work and directions are discussed in a later section.

### 8.1 Conclusions

The focus of this work is to develop a method and system towards building an object-centric environment using WSNs for spatial environmental mapping. A thorough study of the background in the field of WSNs, ANNs and thermal mapping is discussed in the earlier chapters. It is also identified from the literature that the research on thermal mapping using ANN and WSN areas is not mature and need further attention.

Thermal mapping is not a new area of research and there are several conventional methods that are published including CFD and FEM [45, 46]. Most of these methods require intensive computational power and hence are not suitable for the limited resources of wireless sensors. The data loggers are the most convenient devices to log the temperature data for the applications that include food and agricultural industries. These devices can't provide any real time thermal mapping to identify the hotspots. There is a demand for applications that allow for real time thermal analysis and also to predict the temperature at any arbitrary position. On the other hand, WSN are becoming very popular and they decrease in size and increase in computational power with a lower price. The soft computing methods, which are ANN and their applications are explored and used to address the research problem. Hence, our research is focussed on identifying the methodologies that can fulfil these requirements in the WSN area.



Initially the proof of concept is designed to compare the prediction error between the Shepard's algorithm and the ANN approaches in relation to spatial analysis and thermal mapping. The ANN approach resulted in mean absolute error of 0.35 °C compared to 0.8 °C with Shepard's algorithm with modified Euclidian distance. The sensing points are located based on a distribution pattern within the given space. The neural parameters are compared to identify the best network for training. Randomly placed sensing points are also analysed to verify the prediction analysis. Further viability scenarios are examined by varying the confined space, number of infrastructural nodes and the number of portable nodes. It has been observed that there is a drop in RMS error while increasing the number of portable nodes at a given room volume and the infrastructural nodes deployed on inner side of the boundary, but there is a maximum number of portable nodes beyond which the error does not improve. The infrastructural nodes deployed on the inner surface of the boundary have an impact on the thermal prediction.

The core module of this research is the implementation of nWSN architecture and its components for building an object centric thermal mapping environment. The NMi model is developed to organize the communication between the infrastructural nodes and portable nodes. A QnDP algorithm is proposed to fulfil the data synchronization for training in a dynamic environment. A volumetric temperature precision is given to compare the data among synchronous and asynchronous input. Larger errors are recorded when the data is asynchronously submitted into the model. This work disclosed the viability of nWSN architecture to execute further on a real time test bed.

A test bed is constructed at SeNSE lab using Atmel's RZUSBSTICK as a gateway and AVRRAVEN as motes to conduct the experiments. This experimental test bed has confirmed the viability of the proof of concept. The deployed nodes have given a great correlation on volume rendered maps and these results revealed a good accuracy between the testing and predicted data sets.

The nodes minimization approach is proposed to identify the number of nodes required in a given space. The K-Nearest Neighbour Algorithm has been used together with Bayesian maps for evaluating more influenced nodes. This work exploits a spatial correlation of temperature data in a given space. The minimum number of nodes can be identified for any given space.

Finally we have analysed a meat industry case study to mimic a cool storage where the temperature varies with time. We have used 174 carcasses in the model and the experimental scenarios are categorized broadly into four sections:

- 1) The first scenario describes the placement of nodes at three different layers in the space to identify the nodes deployment for better coverage. A dynamic temperature is introduced at each point where it fluctuates between  $\pm 2^{\circ}\text{C}$  with the simulation time. At both peak and dip, the layer 2 nodes deployment has given a high correlation compared to the layer 1 and layer 3.
- 2) The second scenario focussed on the temperature variation analysis at the surface of the each subspace based on the modularization of the space into sub-spaces. There is more error recorded around the subspace. Hence the integration of multiple subspaces needs to be addressed.
- 3) In the third scenario, the inter-arrival time of the carcasses is increased, which reduced the number of carcasses to 86. This result clearly identified inadequate training data at each infrastructural node/cluster head for the given space.
- 4) The final scenario has looked into an increase of dynamic temperature variation to  $\pm 4^{\circ}\text{C}$  with the simulation time to identify the effect on the prediction. The system became very unstable at this rapid change in temperature and reflected an increased prediction error at all the infrastructural nodes.

The concept development and the test bed analysis showed that there is an influence of the nodes placement and the number of training datasets. The results of the various scenarios built lead to the conclusion that the concept is valid for a similar kind of applications.

## **8.2 Future Work and Directions**

The nWSN architecture has delivered promising results for further execution in real time domains. During the implementation of this research, a few assumptions were introduced to allow assessment and development of the concept and some of these

required for further attention. In this section some of these areas will be discussed as a potential future work.

1. *Neural parameters*: The neural network architectures need to be reviewed for other applications. Other than the location and temperature, the variables like the air flow direction could influence the thermal profile predictions. In the Heating, Ventilation and Air-Conditioning (HVAC) systems, we can identify the nearby locations of the source to deploy the sensor nodes for the overall coverage of the space. The temperature gradients vary rapidly towards the direction of the air flow. Hence, the direction of the air flow would give a high precision mapping for applications that require greater accuracies.

2. *Fault detection and isolation*: The training data received from the portable nodes should be identified at the cluster node level to determine, if any wrong information was provided by the node. For example, if nodes fail to detect or sense the temperature, this leads to generating incorrect data in the training process. Hence the implementation of the AI at the cluster node level can block the transmission of erroneous data for further training or validation.

3. *Time synchronization*: The inclusion of time as a neural parameter could resolve the time synchronization issue which is required within a transient space where the portable nodes move with time. The time as an input parameter would give a fourth dimensional mapping of the space.

4. *Shelf life prediction*: Implement and test better models for shelf life prediction during transportation of goods. Tracing and tracking systems during transportation would help taking certain decisions. This would require implementation of the real time data streaming algorithms.

4. *Optimization algorithms*: The minimum number of nodes that are required to map the given space is analysed by using  $k$ -NNA along with Bayesian maps to identify the influenced nodes in this work. This analysis has given the minimum nodes for the defined configuration. But it depends on the spatial distribution of the nodes, size of the space and thermal profile. Locating the nodes is an important criterion along with the number of nodes. In an ideal condition, we assume that

there are no obstacles within the space. But in real world scenarios there might be walls, pillars in the middle of the space. The optimal nodes placement varies from one infrastructure to the other. Hence, the sensor nodes placement needs to be modified for a given infrastructure. An optimization problem needs to be constructed with an objective function to minimize the prediction error and considering all the constraints. Optimization algorithms would be required for future study.

4. *Data mining*: In this study we have used the nWSN modularized system where the space is divided into subspaces. A cluster head is responsible for each subspace. When dealing with greater number of modularized systems, it is important to focus on data streaming for the query system to identify the hot spots and also trigger the alarm for the given thermal boundaries of any individual subspace.

5. *Portable Nodes*: In the experimental scenarios, we have used the motes from Atmel Corporation. These motes may not be suitable for the portable nodes, based on their size and power requirements. Hence it is important to give a focus on hardware development that is suitable to work in any environment.

## 9 References

- [1] D. Estrin, D. Kuller, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," *IEEE Pervasive Computing*, vol. 1, pp. 59-69, 2002.
- [2] J.J Garrahan, P.A. Russo, K. Kitami, and R. Kung, "Intelligent network overview," *IEEE Communication Magazine*, vol. March, pp. 30-36, 1993.
- [3] J. Stankovic, T. Abdelzaher, C. Lu, L.Sha and J. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, pp. 1002-1022, 2003.
- [4] C.Y. Chong and S. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, pp. 1247-1256, 2003.
- [5] F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayircia, "Wireless Sensor Networks: A survey," *Computer networks*, vol.38, issue.4 pp.393-422, March 2002.
- [6] E. Yoneki, "Evolution of ubiquitous computing with sensor networks in urban environments," *Ubicomp-Workshop on Metapolis and Urban life*, pp. 56-60, September 2005.
- [7] B. Hong and V.K. Prasanna, "Constrained flow optimization with applications to data gathering in sensor networks," *First International workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.
- [8] L. Huang-Chen, "Towards a general wireless sensor network platform for outdoor environment monitoring," *IEEE Sensors* , pp.1-5, October 2012.
- [9] S. Qingshan, L. Ying, D. Gareth, and D. Brown, "Wireless Intelligent Sensor Networks for Refrigerated Vehicle," *IEEE Symposium on Emerging Technologies: Mobile and Wireless Communication*, China, 2004.

- [10] N. Wang, N. Zhang, and M. Wang, "Wireless sensors in agriculture and food industry – Recent development and future perspective," *Computer and Electronics in Agriculture*, vol.50, issue.1, pp.1-14, January 2006.
- [11] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring and D. Estrin, "Habitat monitoring with sensor networks," *Journal of Communications*, vol.47, no. 6, pp. 34–40, 2004.
- [12] Z. Butler, P. Corke, R. Peterson, and D. Rus, "Networked Cows: Virtual Fences for Controlling Cows," *The International Journal of Robotics Research*, vol. 25, no. 5, pp. 485-508, 2006.
- [13] A. Ipema, D. Goense, P. Hogewerf, W. Houwers, and H. Roest van, "Real-time monitoring of the body temperature with a rumen bolus," *4th international workshop on smart sensors in livestock monitoring*, pp. 13-14, 2006.
- [14] M. Schwager, D.M. Anderson, Z. Butler and D, Rus, "Robust classification of animal tracking data," *Journal of Computers and Electronics in Agriculture*, vol.56, no.1, pp. 46–59, 2007.
- [15] Wireless medium access control (MAC) and physical layer (PHY) specifications. Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Standard 802.15.4, *The institute of Electrical and Electronics Engineers Inc*, 2003.
- [16] J.N. Burdon and A.F. Bollen, "Hort 16A Coolchain," Report prepared by the *Horticulture and Food Research Institute of New Zealand Ltd* (HortResearch), January 2002.
- [17] D. Tanner and N.D. Amos, "Modelling product quality changes as a result of temperature variability in shipping systems," *International Congress of Refrigeration*, Washington, D.C, 2003.

- [18] J. Choi, M. Bouchard, and T. Yeap, "Decision feedback recurrent neural equalization with fast convergence rate," *IEEE Transactions on Neural Networks*, vol.16, pp. 699–708, 2005.
- [19] A.C. Tsoi and A. Back, "Locally recurrent globally feedforward networks: a critical review of architectures," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 229–239, 1994.
- [20] J. Perez-Ortiz, J. Calera-Rubio and M. Forcada, "A comparison between recurrent neural architectures for real-time nonlinear prediction of speech signals," *Neural Networks for Signal Processing XI*, pp. 73–81, 2001.
- [21] D. C. Psychogios and L. H. Ungar, "A hybrid neural network-first principles approach to process modeling," *AIChE Journal*, vol. 38, pp. 1499 – 1511, 2004.
- [22] Y. Yao, G. Marcialis, M. Pontil, P. Frasconi, and F. Roli, "Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines," *Pattern Recognition*, vol. 36, pp. 397–406, 2003.
- [23] E. J. Hartman, J. D. Keeler and J. Nowalski, "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Computation*, vol. 2, pp. 210–215, 1990.
- [24] K. Watanabe, J. Tang, M. Nakamura, S. Koga and T. Fukuda, "Fuzzy-Gaussian neural network and its application to mobile robot control," *IEEE Transactions on Control Systems Technology*, vol. 4, pp. 193–199, 1996.
- [25] P. Angeline, G. Saunders and J. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 54–65, 1994.

- [26] J. Sum, C. Leung, G. H. Young and W. Kan, "On the Kalman filtering method in neural network training and pruning," *IEEE Transactions on Neural Networks*, vol. 10, pp. 161-166, 1999.
- [27] J. Sum, L. wan Chan, C. sing Leung and G. H. Young, "Extended Kalman filter-based pruning method for recurrent neural networks," *Neural Computations*, vol. 10, no. 6, pp. 1481-1505, 1998.
- [28] Y. Sarig, "Traceability of Food Products," *Agricultural Engineering International: Journal of Scientific Research and Development*, vol. 5, 2003.
- [29] I. Gustafsson, "Information for transparency in transport chains," *Doctoral Thesis*, Blekinge Institute of Technology, Sweden.
- [30] R. Van Hoek, "Using information technology to leverage transport and logistics service operations in the supply chain: An empirical assessment of the interrelation between technology and operations management," *Journal of International Technology and Information Management*, vol.1, no.1, pp. 115-130, July 2002.
- [31] L. Ruiz Garcia, P. Barreiro and J.I. Robla, "Monitoring intermodal refrigerated fruit transport using sensor networks: A review," *Journal of Agricultural Research*, vol.5, issue.2, June 2007.
- [32] M. F. Weinhaus and D. Vankat, "Texture mapping 3D models of real-world scenes," *ACM Computing Surveys (CSUR)*, Vol 29, Issue 4, pp.325-365, 1997.
- [33] K. Kristian, "Spatial sampling and interpolation methods – comparative experiments using simulated data," *Computer Vision and Media Technology Laboratory*, Aalborg University, 2003.
- [34] Lu X, "Modeling heat and moisture transfer in buildings – (I) model program," *Energy Build 34*, pp. 1033-1043, 2002.



- [35] V. T. Thanh, V. Eric and B. Daniel, "Data-based mechanistic modeling of three-dimensional temperature distribution in ventilated rooms filled with biological material," *Journal of Food Engineering* 86, pp.422-432, 2007.
- [36] C. Tedoisu, R. Hohota, G. Rusaouen and M. Woloszyn, "Numerical prediction of indoor air humidity and its effect on indoor environment," *Build Environ* 38(5), pp.655-664, 2003.
- [37] A.E. Ruano, E.M. Crispim and Lucio, "Prediction of buildings temperature using neural networks models," *Energy Build* 38, pp.682-694, 2006.
- [38] I. Maqsood, M.R. Khan and A. Abraham, "Intelligent weather monitoring systems using connectionist models," *International Journal of Neural, Parallel and Scientific Computations*, v.10, pp.157-178, 2000.
- [39] M. Hayayi, T.Yousefi, M.Ashjaee, A.S. Hamidi and Y.Shirvany, "Application of Artificial Neural Network for Prediction of Natural Convection Heat Transfer from a Confined Horizontal Elliptic Tube," *International Journal of Applied sciences, Engineering and Technology*, 4(3), pp.157-162, 2007.
- [40] Z. Shimeld, J.Willianson, M. and J. Katsube, "Permeability prediction with artificial neural network modeling in the Venture gas field," *Offshore eastern Canada: Geophysics*, v.61, no.2, pp. 422-436, 1996.
- [41] H. Salu, and J. Tilton, "Classification of multi-spectral image data by the binary diamond neural network and by nonparametric, pixel by pixel methods," *IEEE Trans. Geo-science, Remote Sens.*, v.31, no.3, pp. 606-617, 1993.
- [42] D.M. Miller, E.J. Kaminsky and S. Rana, "Neural network classification of remote-sensing data," *Computers & Geosciences*, v.21, no.3, pp.337-386, 1995.
- [43] I.K. Kapageridis and B. Denby, "Ore grade estimation with modular neural network system: A case study," *Information technologies in the mineral industry*, pp.52, 1998.

- [44] M. Kanevsky, R. Arutyunyan, L. Bolshov, V. Demyanov and M. Maignan, "Artificial neural networks and spatial estimations of Chernobyl fallouts," *Proceedings IAMG Annual conference*, Osaka, Japan, pp.27-29, 1995.
- [45] S. Gendelis and A. Jakovics, "Numerical Modelling of Airflow and Temperature Distribution in a Living Room with Different Heat Exchange Conditions," *Latvian Journal of Physics and Technical Sciences*, v.47(4), pp.27-43, 2005.
- [46] S. E. Ozcan, C. Ozlem, V. Erik and D. Berckmans, "Predicting 3D spatial temperature uniformity in food storage systems from inlet temperature distribution," *Postharvest Biology and Technology*, vol. 37, pp. 186-194, 6 April 2005.
- [47] R. Beckwith, D. Teibel, and P. Bowen, "Unwired wine: sensor networks in vineyards," *Sensors, 2004. Proceedings of IEEE*, vol.2, pp.561,564, 24-27 Oct. 2004.
- [48] Y.A.L. Borgne, M. Moussaid and G. Bontempi, "Simulation architecture for data processing algorithms in wireless sensor networks," *20th International Conference on Advanced Information Networking and Applications*, v.2, pp.5, 18-20 April 2006.
- [49] H. Zhang, Jose M. F. Moura, and B. Krogh, "Estimation in Sensor Networks: A Graph Approach," *Information Processing in Sensor Networks*, pp. 203-209, 2005.
- [50] F. Hugo Uchida and J. G. Pieters, "Modelling greenhouse temperature using system identification by means of neural networks," *Neurocomputing*, v.56, pp. 423-428, 2004.
- [51] S. Qingshan, L. Ying and D. Garath, "Wireless intelligent sensor networks for refrigerated vehicle," *In IEEE 6th Symposium on Emerging Technologies Mobile and Wireless Communication*, v.2, pp. 525-528, 2004.

- [52] L. A. Zadeh, "Fuzzy Sets," *Information and Control*. 8(3), pp.338-353, 1997.
- [53] P. P. Bonissone, "Soft Computing: The Convergence of Emerging Reasoning Technologies," *Soft Computing*. Springer-Verlag, v.1(1), 1997.
- [54] B. Cheng and D. Titterington, "Neural Networks: a Review from a Statistical Perspective," *Statistical Science*. vol. 9, pp.2-54, 1994.
- [55] S. V. Kartalopoulos, "Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications," *IEEE Press*, 1996.
- [56] M. Kanevsky, R. Arutyunyan, L. Bolshov, V. Demyanov, and M. Maignan, "Artificial neural networks and spatial estimations of Chernobyl fallouts," *Proceedings IAMG Annual conference*, Osaka, Japan, pp.27-29, 1995.
- [57] I. Seginer, "Some artificial neural network applications to greenhouse environmental control," *Computer and Electronics in Agriculture*, v.18 (2-3), pp.167-186, 1997.
- [58] P.M Ferreira, E.A. Faria and A.E. Ruano, "Neural network models in greenhouse air temperature prediction," *Neurocomputing*, v.43 (1-4), pp. 51-75, 2002.
- [59] Y. Huang, "Advances in artificial neural networks-Methodological development and applications," *Algorithms 2009*, v.2, pp.973-1007, 2009.
- [60] Y. Radhika and M. Shashi, "Atmospheric temperature prediction using support vector machines," *International Journal of Computer Theory and Engineering*, v.1(1), pp. 55-58, 2009.
- [61] Z. Jun, L. Chunbo, G. Jianhua, W. Daojun and S. Tao, "A distributed computing service for neural networks and its application to flood peak forecasting," *Proceedings ICONIP 2006*, Part II, pp. 890-896, 2006.

- [62] U. Lotrič and A. Dobnikar, "Parallel implementations of feed-forward neural network using MPI and C# on .NET platform," *Adaptive and Natural Computing Algorithms*, Part VI, pp.534-537, 2005.
- [63] M. Hamidreza and M. Alireza, "XDANNG: XML based distributed artificial neural network with globus toolkit," *International Journal of Computer Science and Information Security*, v.2(1), pp.39-41, 2009.
- [64] L. Cristaldi, M. Faifer, F. Grande and R. Ottoboni, "An Improved M2M Platform for Multi-Sensors Agent Application," *Sensors for Industry Conference*, pp.79-83, Feb 2005.
- [65] L.B. Ruiz, F.A. Silva, T.R.M. Braga, J.M.S. Nogueira and A.F. Loureiro, "On Impact of Management in Wireless Sensors Networks," *Network Operations and Management Symposium*, v.1, pp.657–670, 19-23 April 2004.
- [66] K. H. Hiramatsu, T. Yamada and T. A. Okadome, "Finding Small Changes using Sensor Networks," *Smart Object Systems (in conjunction with Ubicomp2005)*, pp. 37-44, 2005.
- [67] J. Heidemann, W. Ye, J. Wills, A. Syed and Y. Li, "Research challenges and applications for underwater sensor networking," *IEEE Wireless Communications and Networking Conference*, pp.228-235, April 2006.
- [68] F.L. Lewis, "Wireless sensor networks," *Smart Environments: Technologies, Protocols, and Applications*, pp.1-17, 2004.
- [69] B. Son, Y. Her and J. Kim, "A design and implementation of forest-fires surveillance system based on wireless sensor networks for South Korea mountains," *IJCSNS International Journal of Computer Science and Network Security*, v.6, pp.124-130, 2006.
- [70] G. Vellidis, M. Tucker, C. Perry, C. Wen and C. Bednarz, "A real-time wireless smart sensor array for scheduling irrigation," *Computers and Electronics in Agriculture*, v.61, pp. 44-50, 2008.

- [71] G. Liu and Y. Ying, "Application of Bluetooth technology in greenhouse environment, monitor and control," *Journal of Zhejiang University (Agriculture & Life Science Edition)*, v.29, pp. 329-334, 2003.
- [72] L. Gonda and C.E. Cugnasca, "A proposal of greenhouse control using wireless sensor networks," *In computers in Agricultural and Natural Resources, 4th world Congress Conference*, pp. 18-21, 2006.
- [73] M. Umer, L. Kulik, and E. Tanin, "Optimizing query processing using selectivity-awareness in Wireless sensor networks," *Computers, Environment and Urban Systems* 33, pp.79-89, 2009.
- [74] H. Wu, Q. Luo and W. Xue, "Distributed cross-layer scheduling for in network sensor query processing," *IEEE International Conference on Pervasive Computing and Communications(PerCom)*, pp.179-189, 2006.
- [75] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE CS and IEEE ComSoc*, pp.46-55, 2004.
- [76] K. Xing, X. Cheng, J. Li and M. Song, "Location-centric storage and query in wireless sensor networks," *Wireless Networks*, pp.955-967, 2010.
- [77] Chi-yin Chow, Mohamed F. Mokbel and Tian He, "Tinycasper: a privacy preserving aggregate location monitoring system in wireless sensor," *International Conference on Management of Data – SIGMOD*, pp.1307-1310, 2008.
- [78] S. Tian, S. M. Shatz, Y. Yu and J. Li, "Query sensor networks using ad hoc mobile devices: A two-layer networking approach," *AdHoc Networks*, v.7, pp.1014-1034, 2009.
- [79] K Wang-il H, Jeongsik In and NhoKyung Park, "A design and implementation of wireless sensor gateway for efficient querying and managing through world wide web," *IEEE Transactions on Consumer Electronics*, v.49(4), pp.1090-1097, 2003.

- [80] S. Chatterjea and P. Havinga, "A Dynamic Data Aggregation Scheme for Wireless Sensor Networks," *14th Workshop on Circuits, Systems and Signal Processing (ProRISC)*, 26-27 November 2003.
- [81] W. Hejun and Q. Luo, "Adaptive holistic scheduling for query processing in sensor networks," *Journal of Parallel and Distributed Computing*, v. 70(6), pp. 657-670, 2010.
- [82] X. Zhang, X. Yu and X. Chen, "Inter-query data aggregation in wireless sensor networks," *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 930-933, 2005.
- [83] R.I. da Silva and V. Del Duca Almeida, "Spatial query processing in wireless sensor network for disaster management," *2nd IFIP Wireless Days (WD)*, pp.1-5, 2009.
- [84] S. Spanache, T. Escobet, et al. "Sensor Placement Optimization Using Genetic Algorithms," *Proceedings DX-2004, 15th International Workshop on Principles of Diagnosis*, France. June 2004.
- [85] H. Gonzalez-Banos and J.-C. Latombe, "A randomized art-gallery algorithm for sensor placement," *In Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG)*, pp. 232-240, 2002.
- [86] A. Birchall, "A microcomputer algorithm for solving compartmental models involving radionuclide transformations," *Health Physics* 50 (3), pp. 389-397, 1986.
- [87] A. Birchall and A.C. James. "A microcomputer algorithm for solving first-order compartmental models involving recycling," *Health Physics* 56 (6), pp. 857-868, 1998.
- [88] F. Gelbard, J.E. Brockmann, K.K. Murata and W.E. Hart, "An algorithm for locating sensors in a large multi-room building," *Tech. Rep. SAND2000-0851, Sandia National Laboratories*, 2000.

- [89] Y.O. Deog and C.N. Hee, "Determination of the minimal number and optimal sensor location in a nuclear system with fixed incore detectors," *Nuclear Engineering and Design*, Volume 152(1-3), pp. 197-212, Nov 1994.
- [90] S. Dhillon, K. Chakrabarty, and S. S. Iyengar, "Sensor placement for grid coverage under imprecise detections," *Proceedings of the 5th ISIF International Conference on Information Fusion*, pp. 1581-1587, July 2002.
- [91] K. Flouri, B. Beferull-Lozano and P. Tsakalides, "Optimal gossip algorithm for distributed consensus SVM training in wireless sensor networks," *16th International Conference on Digital Signal Processing*, pp.1-6, 2009.
- [92] A. Pozdo, L. Foresti and M. Kanevski, "Data-driven topo-climatic mapping with machine learning methods," *Natural Hazards: Journal of the International Society for the Prevention and Mitigation of Natural Hazards*, v.50, pp.497-518, 2009.
- [93] A. Kulakov, D. Davcev and G. Trajkovski, "Application of wavelet Neural-Networks in Wireless Sensor Networks," *Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp.262-267, 2005.
- [94] V. Lakshmi Narasimhan, A. Alex, Arvind and Ken Bever, "Greenhouse asset management using wireless sensor actor networks", *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pp. 9-14, 2007.
- [95] L. Bottou, "On-line learning in neural networks," *International Conference on On-line learning and stochastic approximations*, Cambridge University Press, pp. 9-42, 1998.
- [96] A. Sharma, T. Banerjee and D. P. Agarwal, "Exploiting Spatial Correlation in a three dimensional Wireless Sensor Network," *Mobile Adhoc and Sensor Systems*, pp. 1-6, 2007.

- [97] L.B. Yann-Ael, M. Moussaid and G. Bontempi, "Simulation Architecture for Data Processing Algorithms in Wireless Sensor Networks," *In Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA '06)*, 2006.
- [98] M. Kolahdouzan and C. Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *Proceedings of the 30th VLDB Conference*, Toronto, Canada, pp. 840-851, 2004.
- [99] B. Zhang and S.N. Srihari, "A Fast Algorithm for Finding k-Nearest Neighbors with Non-metric Dissimilarity," *Proceedings of the Eighth International workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, pp.13-18, 2002.
- [100] A. I. Manjula, L.T. Watson and M.W. Berry, "SHEPPACK: A Fortran 95 package for Interpolation using the Modified Shepard Algorithm," *ACM SE'06*, Florida, USA, pp. 476-481, 2006.
- [101] H. Ledoux, "Computing the 3D Voronoi Diagram Robustly: An Easy Explanation," *4th International Symposium on Voronoi Diagrams in Science and Engineering*, pp.117-129, 9-11 July 2007.
- [102] W.C.M. Van Beers and J.P.C. Kleijnen, "Kriging interpolation in simulation: a survey," *Simulation Conference, Proceedings of the 2004*, v.1, pp.121, 5-8 Dec 2004.
- [103] Z. Song and R. Nick, "K-Nearest neighbor search for moving query point," *7th International Symposium on Advances in spatial and temporal databases*, pp. 79-96, July 2001.
- [104] Flexsim simulation environment <http://www.flexsim.com>
- [105] Adaptive systems development environment <http://www.peltarion.com>



- [106] Alyuda neural network software <http://www.alyuda.com>
- [107] M. Artur, “Three-Dimensional wave polynomials”, *Mathematical problems in Engineering*, pp.583-598, 2005.
- [108] K.N. Jainendra, “An analytical technique for 3-dimensional interpolation”, *Journal of BIT Numerical mathematics*, Vol 24(1), pp.119-122, 1984.
- [109] L.B. Chandrajit, “Multi-dimensional Hermit interpolation and approximation for modelling and visualization”, *ICCG Proceedings of the CSI international conference on Computer graphics*, pp.335-348, 1993.
- [110] IBM’s Mote Runner Software <http://www.zurich.ibm.com/moterunner/>
- [111] A report submitted to Auckland Meat Processing (AMP) plant, Auckland – Carne Technologies <http://www.carnetech.co.nz>

# A. Appendices

The implementation of the nWSN architecture involves the software user interface and hardware that needs to be customized to mimic the real-time environment. We have used RZUSBSTICK and AVRRAVEN from Atmel to test the developed concepts. The effort towards building a three dimensional virtual reality environment to run the experiments has been successfully implemented within the simulation modeling. This appendices section gives all about the interfaces and coding.

## A.1 Hardware Environment:

The hardware environment from Atmel Corporation allows the programmers to use object-oriented languages such as C# and Java to develop portable WSN applications that can be dynamically distributed, loaded and updated even after the deployment. The developed code is embedded into the infrastructural node and portable nodes. The QBnWSN uses the time synchronization for the synchronized data input to the cluster head. The code for these nodes is given here.

### IBMs Mote Runner Implementation:

*Batch file to compile the assembly (start.bat)*

```
mrgac --del Inode-6.0
mrgac --del Pnode-6.0

mrc --assembly=Inode-6.0 Inode.java
mrc --assembly=Pnode-6.0 Pnode.java

mrgac --copy Inode-6.0.sba
mrgac --copy Pnode-6.0.sba

mrsh.exe
```

*Simulation initialization code (run.mrsh):*

```
saguaro-start
lip-enu
```

```
lip-create -p 02-00-00-00-A8-A8-D2-12
sleep 3000
l0 mote-reset
sleep 3000
l0 wlip-setup
sleep 3000
```

```
lip-create -p 02-00-00-00-2F-5B-50-04
sleep 3000
l1 mote-reset
sleep 3000
l1 moma-delete Inode-6.0
l1 moma-load Inode-6.0
sleep 3000
wlip-appeal
socket-bind -s collect.js collect
socket-send collect 1 00
```

### *A.1.1 Infrastructural node*

```
package coord;
import com.ibm.saguaro.system.*;
public class Inode {
    private static final int GATEWAY = 0x5678;
    private static final int PANID = 0x1234;
    private static byte[] data = new byte[LIP.PAYLOAD + 10];

    static {
        Radio.acquire();
        Radio.setShortAddr(GATEWAY);
        Radio.setPanId(PANID, false);
        Assembly.setDataHandler(new DataHandler(null) {

            @Override
            public int invoke(int info, byte[] data, int len) {
                return onSerialData(info, data, len);
            }
        });
        Radio.setRxDone(new RadioDone(null) {
            @Override

            public void invoke(int info) {
                onRxDone(info);
            }
        });
        Radio.setRxHandler((byte)0, new RadioRxPdu(null) {
            @Override

            public void invoke(byte[] data, int len, long time, int quality) {
                onWirelessData(data, len, time, quality);
            }
        });
    }
}
```

```

    }
    });
    Radio.enableRx(Time.currentTicks() +
    Time.toTickSpan(Time.SECONDS, 20));
    }

    private static void onWirelessData(byte[] pdu, int len, long time, int
    quality) {
        Util.copyData(pdu, 9, data, LIP.PAYLOAD, 10);
        LIP.send(data, 0, LIP.PAYLOAD + 10);
    }

    private static int onSerialData(int info, byte[] buf, int len){
        Util.copyData(buf, 0, data, 0, LIP.PAYLOAD);
        return len;
    }

    private static void onRxDone(int info) {
        Radio.enableRx(Time.currentTicks() +
        Time.toTickSpan(Time.SECONDS, 20));
    }
}

```

### *A.1.2 Portable node*

```

package snode;
import com.ibm.saguaro.system.*;
public class Pnode {
    private static final int PANID = 0x1234;
    private static final int GATEWAY = 0x5678;
    private static byte[] header = new byte[11];
    private static byte[] data = new byte[10];
    private static Timer timer = new Timer();
    private static long INTERVAL = Time.toTickSpan(Time.SECONDS,
    10);
    private static boolean started = false;
    private static byte[] extendedAddress = new byte[8];
    private static int shortAddress;
    private static int xval;
    private static int yval;
    private static int zval;
    private static int temp;

    static {
        Assembly.setDataHandler(new DataHandler(null) {

            public int invoke(int info, byte[] data, int len) {
                return onWLIPData(info, data, len);
            }
        });
    }
}

```

```

Assembly.setSystemInfoCallback(new SystemInfo(null) {

    public int invoke(int type, int info){
        return onSystemEvent(type, info);
    }
});

private static int onWLIPData(int info, byte[] data, int len) {
    WLIP.shutdown();
    return 0;
}

private static int onSystemEvent(int type, int info) {
    // system notification that WLIP has been shutdown
    if (type == WLIP.SYSEV_WLIP_DOWN)
        start();
    return 0;
}

static private void start() {
    Radio.acquire(); // before we can use the radio we need to acquire
                    // it
    Radio.getExtAddr(extendedAddress, 0);
    shortAddress = Util.get16le(extendedAddress, 0);
    Radio.setShortAddr(shortAddress);
    Radio.setPanId(PANID, false);
    header[0] = Radio.FCF_DATA | Radio.FCF_ACKRQ;
    header[1] = Radio.FCA_SRC_SADDR |
    Radio.FCA_DST_SADDR;
    header[2] = 1; // sequence number
    // destination
    Util.set16le(header, 3, PANID);
    Util.set16le(header, 5, GATEWAY);
    // source
    Util.set16le(header, 7, PANID);
    Util.set16le(header, 9, shortAddress);
    timer.setCallback(new TimerEvent(null) {
        public void invoke(byte param, long time) {
            onTimeout(param, time);
        }
    });
    // delay acquiring the radio so that the loading would not fail
    timer.setAlarmTime(Time.currentTicks() + INTERVAL);
}

static void onTimeout(byte param, long time) {
    try {
        xval = 500;
        yval = 500;
        zval = 10;
        Util.set16be(data, 0, xval);
    }
}

```

```

        Util.set16be(data, 2, yval);
        Util.set16be(data, 4, zval);
        SimpleDevices.read(SimpleDevices.MOTE_TEMP, 0, 0, data, 6,
        2);
    }
    catch (MoteException e) {
        LED.setState((byte) 2, (byte) 1);
    }
    Radio.transmit(Radio.TXMODE_CSMA, header, 11, data, 0, 8, null);
    timer.setAlarmTime(time + INTERVAL);
}
}

```

### *A.1.3 Java script file for web-based data monitoring*

```

User.collect = {
onData: function(blob) {

    var ret = "";
    if (blob.data.length == 0){
        ret = "Connected.\n";
    }
    else if (blob.data.length == 10){
        var tmp = Formatter.unpack("2uL2u2u2u", blob.data);
        var src = tmp[0];
        var xloc = tmp[1];
        var yloc = tmp[2];
        var zloc = tmp[3];
        var T = tmp[4];
        var temp= T; /* 0.09-45;
        var file;

if(src == 33089) //Node 1
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode1.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
if(src == 17941) //Node 2
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode2.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
if(src == 31147) //Node 3
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode3.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
if(src == 8902) //Node 4
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode4.txt", "w+");
    }
}
}

```

```

        IO.File.fwrite(file, "" + temp);
    }
    if(src == 57811) //Node 5
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode5.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
    if(src == 21966) //Node 6
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode6.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
    if(src == 31293) //Node 7
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode7.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
    if(src == 11374) //Node 8
    {
        file = IO.File.fopen("D:/1/SeNSe/fuswid/Pnode8.txt", "w+");
        IO.File.fwrite(file, "" + temp);
    }
    IO.File.fclose(file);
    var currentTime = new Date();
    var hours = currentTime.getHours();
    var minutes = currentTime.getMinutes();
    if (minutes < 10){
        minutes = "0" + minutes;
    }
    //var tm = sprintf(hours + ":" + minutes + " \n");
    ret = sprintf("Time= %d:%d, SensorID = %X, X = %d, Y = %d, Z = %d,
    Temperature =
    %.2f C\n", hours, minutes, src, xloc, yloc, zloc, temp);
    }
    else {
        var tmp = Formatter.binToHex(blob.data);
        //var tmp = blob.data.length;
        ret = sprintf("Data = %s\n", tmp);
        //ret = "ERROR: unexpexted length \n";
    }
    return ret;
},

send: function(dstport, dstmote, argv) {
    return "";
},
broadcast: function(dstmote, dstport, argv) {
    return "";
},
onClose: function(status) { }
};

```

#### A.1.4 Time synchronization – QBnWSN framework: initialization

```
namespace com.ibm.moterunner.nwsn {
internal sealed class HDEFS {
    internal HDEFS () {} // Default CTOR
    internal const byte APPLICATION_OFF = 0x00000002;
    internal const byte CHANNEL = 0x00000003;
    internal const byte CMD_OFF = 0x00000001;
    internal const byte CMD_UPDT = 0x00000002;
    internal const byte CMD_WLIP = 0x00000001;
    internal const uint GATEWAY = 0xD1D1;
    internal const byte MAX_DEVICES = 0x00000008;
    internal const byte MOTEID_OFF = 0x00000007;
    internal const uint PANID = 0x4ACE;
    internal const byte PORT = 0x0000000F;
    internal const byte RATE = 0x00000005;
    internal const byte RATE_OFF = 0x00000003;
    internal const byte STATUS_OFF = 0x00000000;
    internal const byte TEMP_CURR_OFF = 0x00000010;
    internal const byte TEMP_HIGH_OFF = 0x00000006;
    internal const byte TEMP_LOW_OFF = 0x00000004;
    internal const byte WDATA_LEN = 0x00000026;
    internal const byte WPAYLOAD_OFF = 0x00000009;
}
}
```

#### A.1.5 Time synchronization – QBnWSN framework: infrastructural node

```
namespace com.ibm.moterunner.nwsn {
using com.ibm.saguaro.system;
#if DEBUG
using com.ibm.saguaro.logger;
#endif
public class nWSN {
    //internal static uint lip;
    // temporary data to be sent to motes
    internal static byte[] pendingData;
    internal static uint pending;
    // list of all known mote ids
    internal static uint[] knownMotes;
    internal static uint knownNum = 0;
    // buffer for sending radio messages
    //internal static byte[] radioMessage;
    internal static byte[] lipBuffer;
    internal static int handleLIP (int info, byte[] data, uint len){
    // remember senders address
```



```

Util.copyData(data, 0, lipBuffer, 0, LIP.PAYLOAD);
// simply attach message
if (len <= LIP.PAYLOAD + 1)
return -1;
// proper message
// add pending data to forward the data to the specified mote
addPendingData(data, LIP.PAYLOAD, len - LIP.PAYLOAD);
return -1;
}

internal static void onRxPdu (byte[] pdu, uint len, long time, uint
quality) {
// we just received some data from a mote
uint moteAddr = Util.get16le(pdu, HDEFS.MOTEID_OFF);
#ifdef DEBUG
Logger.appendString(csr.s2b("rx - moteAddr:"));
Logger.appendHexInt(moteAddr);
Logger.flush(Mote.ERROR);
#endif
// check whether the mote has some pending data
// and forward data from mote to the webapp
checkAndForwardPendingData(moteAddr);
// forward data from packet to webapp
Util.copyData(pdu, HDEFS.WPAYLOAD_OFF-2, lipBuffer,
LIP.PAYLOAD, len+2-HDEFS.
WPAYLOAD_OFF);
LIP.send(lipBuffer, 0, LIP.PAYLOAD+HDEFS.WDATA_LEN);
}

static nWSN () {
/*uint lip =*/ LIP.open(HDEFS.PORT, handleLIP);
// initialize data
pendingData = new byte[HDEFS.MAX_DEVICES *
HDEFS.WDATA_LEN];
knownMotes = new uint[HDEFS.MAX_DEVICES];
lipBuffer = new byte[LIP.PAYLOAD +
HDEFS.WDATA_LEN];
// no pending data
pending = 0x00;
Radio.acquire();
Radio.setChannel(HDEFS.CHANNEL);
// set addresses for radio filter
Radio.setShortAddr(HDEFS.GATEWAY);
Radio.setPanId(HDEFS.PANID, false);
// enable receiver
Radio.setRxDone(onRxDone);
Radio.setRxHandler(/*backlog*/0,onRxPdu);
Radio.enableRx(Time.currentTicks() + 0xEEEEEE);
}

internal static uint getMoteIndex (uint moteAddr) {
uint index = 0;

```

```

        for ( ; index < HDEFS.MAX_DEVICES ; index++)
        if (knownMotes[index] == moteAddr)
        return index;
        return index;
    }

    internal static void checkAndForwardPendingData (uint moteAddr) {
        uint motIdx = getMoteIndex(moteAddr);
        if (motIdx >= HDEFS.MAX_DEVICES) {
            // we do not know this mote
            knownNum++;
            if (knownNum == HDEFS.MAX_DEVICES)
                knownNum = 0;
            knownMotes[knownNum]=moteAddr;
            motIdx = knownNum;
        }
        #if DEBUG
        Logger.appendString(csr.s2b("motIdx: "));
        Logger.appendInt((int)motIdx);
        Logger.flush(Mote.ERROR);
        Logger.appendString(csr.s2b("pending: "));
        Logger.appendHexInt(pending);
        Logger.flush(Mote.ERROR);
        #endif
        if ( (pending & (1<<(byte)motIdx)) != 0){
            pending &= ~((uint)(1<<(byte)motIdx)); // clear pending bit
            uint off = motIdx * HDEFS.WDATA_LEN;
            LED.setState(0,1); // yellow LED used for marking transmissions
                                from gateway
                                to packet
            Radio.transmit(Radio.TXMODE_CSMA, pendingData, off,
                HDEFS.WDATA_LEN, onTxDone
            );
        }
    }

    internal static void onTxDone(byte[] pdu, uint len, int status, long
txend) {
        LED.setState(0,0);
    }

    internal static void addPendingData (byte[] data, uint offset, uint len){
        uint moteAddr = Util.get16le(data, offset);
        #if DEBUG
        Logger.appendString(csr.s2b("addPendingData:"));
        Logger.appendHex(data, 0, len + LIP.PAYLOAD);
        Logger.flush(Mote.ERROR);
        #endif
        uint motIdx = getMoteIndex(moteAddr);
        #if DEBUG
        Logger.appendString(csr.s2b("pend - moteAddr:"));
        Logger.appendHexInt(moteAddr);

```

```

        Logger.flush(Mote.ERROR);
    #endif
    if (moteIdx >= HDEFS.MAX_DEVICES) {
        LED.setState(2,1); // red LED unknown mote
        // we do not know this mote
        return;
    }

    #if DEBUG
    Logger.appendString(csr.s2b("... we know it:"));
    Logger.flush(Mote.ERROR);
    #endif
    uint off = moteIdx * HDEFS.WDATA_LEN;
    Util.copyData(data, offset+2, pendingData, off +
        HDEFS.WPAYLOAD_OFF, len-2);
    // fix header
    pendingData[off+0/*FCF*/] = Radio.FCF_DATA |
        Radio.FCF_ACKRQ | Radio.FCF_NSPIID;
    pendingData[off+1/*FCA*/] = Radio.FCA_SRC_SADDR |
        Radio.FCA_DST_SADDR;
    pendingData[off+2/*SEQNO*/] = 0xDD;
    Util.set16le(pendingData, off+3, HDEFS.PANID);
    Util.set16le(pendingData, off+5, moteAddr);
    Util.set16le(pendingData, off+7, HDEFS.GATEWAY);
    // mark that data is pending for this mote
    pending |= (uint)(1<<(byte)moteIdx);
}

internal static void onRxDone (uint info){
    Radio.enableRx(Time.currentTicks() + 0xEEEEEE);
}
}
}

```

*A.1.6 Time synchronization – QBnWSN framework:  
portable node*

```

namespace com.ibm.moterunner.nwsn {
using com.ibm.saguaro.system;
#if DEBUG
using com.ibm.saguaro.logger;
#endif
public class Packet {

    [Immutable]
    internal static readonly byte[] persistentData = {1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1,
    };
}
}

```

```

1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1
}; // 32 bytes
// data used to send to gateway
internal static byte[] data;
internal static byte status;
internal static byte rate_sec = HDEFS.RATE;
internal static long rate_ticks;
internal static byte application = 0;
// temperature thresholds
internal static uint temp_low = 4;
internal static uint temp_high = 30;
// Temperature Average
internal static uint count;
internal static uint temp;
internal static uint temp_qry;
internal static uint temp_min=0;
internal static uint temp_max=0;
internal static uint temp_avg;
internal static uint timebefore = 0;
internal static uint trcounter = 0;
internal static byte radiostatus;
//internal static uint nwsnTemp;
internal static uint[] tempdata = new uint[12];
internal static Timer timer;
// Timer at sensor node
internal static Timer mytimer;
internal static long rate_myticks;
internal static uint numSecs = 0;
internal static uint numMins = 0;
internal static uint numHours = 0;
internal static uint xval = 3;
internal static uint yval = 7;
internal static uint zval = 2;
internal static byte seqno;
internal static uint shortAddr;

static Packet () {
    status = 0;
    // turn off all leds
    LED.setState(0,0);
    LED.setState(1,0);
    LED.setState(2,0);

    // check whether we have some persisten data
    if (persistentData[HDEFS.CMD_OFF] == HDEFS.CMD_UPDT){
        #if DEBUG
        Logger.appendString(csr.s2b("we have persistent"));
        Logger.flush(Mote.ERROR);
        #endif
        // we have persistent data

```

```

    application = persistentData[HDEFS.APPLICATION_OFF];
    rate_sec = persistentData[HDEFS.RATE_OFF];
    // thresholds
    temp_low = Util.get16be(persistentData,
        HDEFS.TEMP_LOW_OFF);
    temp_high = Util.get16be(persistentData,
        HDEFS.TEMP_HIGH_OFF);
}
rate_ticks = Time.toTickSpan(Time.SECONDS, rate_sec);
Assembly.setSystemInfoCallback(onSystemEvent);
// use a timer to start the application after ~ 2s
timer = new Timer();
timer.setAlarm(initialStart, Time.currentTicks() + 2000000);
rate_myticks = Time.toTickSpan(Time.SECONDS, 1);
// Use a timer to count the TIME in an ARRAY
mytimer = new Timer();
mytimer.setAlarm(myTimeRun, rate_myticks);
}

internal static void initialStart (byte param, long time) {
    WLIP.shutdown();
}

internal static int onSystemEvent (int type, int info) {
    if (type == WLIP.SYSEV_WLIP_DOWN){
        LED.setState(0,0); // end of maintenance yellow off
        start(); // we can now start the application
    }
    return 0;
}

internal static void start () {
    Radio.acquire();
    Radio.setChannel(HDEFS.CHANNEL);
    Radio.setRxDone(onRxDone);
    Radio.setRxHandler(/*backlog*/0,onRxPdu);
    data = new byte[HDEFS.WDATA_LEN];
    Radio.getExtAddr(data, 0); // use data for temporary read of
    extended address
    shortAddr = Util.get16le(data, 0);
    // set addresses for radio filter
    Radio.setShortAddr(shortAddr);
    Radio.setPanId(HDEFS.PANID, false);
    // set up radio message frame
    // 1 1 | 1 | 2 | 2 | 2 | ... | # bytes
    //+---+---+---+---+---+---+---+---+---+---+
    +-
    //| FCF | FCA | SEQNO | DSTPAN | DSTADDR | SRCADDR |
    payload | field name
    //+---+---+---+---+---+---+---+---+---+
    +-

```

```

data[0/*FCF*/] = Radio.FCF_DATA | Radio.FCF_ACKRQ |
Radio.FCF_NSPID;
data[1/*FCA*/] = Radio.FCA_SRC_SADDR |
Radio.FCA_DST_SADDR;
data[2/*SEQNO*/] = seqno;
Util.set16le(data, 3, HDEFS.PANID);
Util.set16le(data, 5, HDEFS.GATEWAY);
Util.set16le(data, 7, shortAddr);
rate_ticks = Time.toTickSpan(Time.SECONDS, rate_sec);
// start a timer based on the persistent values
timer.setAlarm(sense, Time.currentTicks() + rate_ticks);
mytimer.setAlarm(myTimeRun, 1);
tempdata[0] = 6000;
}

```

```

internal static void onRxPdu (byte[] pdu, uint len, long time, uint
quality) {

```

```

    #if DEBUG
    Logger.appendString(csr.s2b("we got message"));
    Logger.flush(Mote.ERROR);
    #endif
    uint off = HDEFS.WPAYLOAD_OFF;
    // what command did we receive
    byte cmd = pdu[off + HDEFS.CMD_OFF];

    if (cmd == HDEFS.CMD_WLIP){
        // we need to switch to management mode
        // -----
        // we will be notified with a sysev_wlip_down event
        LED.setState(0,1); // maintenance = yellow on
        timer.cancelAlarm();
        Radio.release();
        WLIP.activate(onSystemEvent, false /*keep WLIP up
even if no gateway found
the first time*/);
        return;
    }

```

```

    if (cmd == HDEFS.CMD_UPDT){
        // we need to update our persistent parameters
        Util.updatePersistentData(pdu, off, persistentData, 0,
HDEFS.WDATA_LEN);
        // and current application parameters
        application = pdu[off+HDEFS.APPLICATION_OFF];
        // rate
        rate_sec = pdu[off+HDEFS.RATE_OFF];
        rate_ticks = Time.toTickSpan(Time.SECONDS, rate_sec);
        timer.cancelAlarm();
        timer.setAlarmTime(Time.currentTicks() + rate_ticks);
        // thresholds
        temp_low = Util.get16be(pdu, off+HDEFS.TEMP_LOW_OFF);
    }

```

```

        temp_high = Util.get16be(pdu,
        off+HDEFS.TEMP_HIGH_OFF);
        #if DEBUG
        Logger.appendString(csr.s2b("temp low = "));
        Logger.appendInt((int)temp_low);
        Logger.appendString(csr.s2b("temp high = "));
        Logger.appendInt((int)temp_high);
        Logger.appendInt((int)temp_avg);
        Logger.flush(Mote.ERROR);
        #endif
        return;
    }
}

internal static void onRxDone (uint info){
    // nothing to do
}

internal static void sense (byte param, long time)
{
    LED.setState(1,(byte)(LED.getState(1)^1)); // toggle green LED
    uint off = HDEFS.WPAYLOAD_OFF;

    try { // Sample sensors directly into current data message
        // temperature
        SimpleDevices.read(SimpleDevices.MOTE_TEMP, 0, 0,
        data, off+HDEFS.TEMP_CURR_OFF, 2);
        temp = Util.get16be(data, off+HDEFS.TEMP_CURR_OFF);

        if(temp_min == 0 || temp_max == 0)
        {
            temp_min = temp;
            temp_max = temp;
            temp_avg = temp;
        }

        if(temp <= temp_min)
            temp_min = temp;

        if(temp >= temp_max)
            temp_max = temp;
        // check the status for the application

        switch (application)
        {
            case 0:
                temp_qry = temp_min;
                radiostatus = 0;
                if (temp >= temp_low && temp <= temp_high)
                    status = 0;
                else status = 1;
                break;

```

```

        case 1:
            temp_gry = temp_max;
            radiostatus = 0;
            status = 0;
            break;
        case 2:
            temp_gry = temp_avg;
            radiostatus = 1;
            status = 0;
            break;
    }

    catch (MoteException) {
        LED.setState(2,1); // signal failed sensing with red LED
    }

    # Radio message payload PARAMS
    # 1 | 1 | 1 | 1 | 1 | ... | # bytes
    #+-----+-----+-----+-----+=====
    #| status | command | application | rate | thresholds | field name
    #+-----+-----+-----+-----+=====
    # Radio message payload (continued) THRESHOLDS
    # 2 | 2 | 2 | 2 | 2 | 2
    | # bytes
    #+-----+-----+-----+-----+-----+-----+
    =====
    #| temp low | temp high |
    field name
    #+-----+-----+-----+
    # Radio message payload (continued) CURRENT DATA
    # 2 | 2 | 2 | # bytes
    #+-----+-----+-----+
    #| temp curr | field name
    #+-----+-----+-----+
    // we already have the status
    data[off] = status; off++; //status
    data[off] = HDEFS.CMD_UPDT; off++; // cmd
    data[off] = application; off++; // application
    data[off] = rate_sec; off++; // rate
    // also send thresholds
    Util.set16be(data, off, tempdata[timebefore]); off += 2;
    Util.set16be(data, off, temp_high); off += 2;
    Util.set16be(data, off, numHours); off += 2;
    Util.set16be(data, off, numMins); off += 2;
    count = count+1;
    trcounter = trcounter+1;
    Util.set16be(data, off, numSecs); off += 2;
    Util.set16be(data, off, temp_gry); off += 2;
    uint sid = shortAddr & 7;

    switch (sid) {
    case 2:
        xval = 10;

```



```

        yval = 15;
        zval = 5;
        Util.set16be(data, off+5, xval);
        Util.set16be(data, off+7, yval);
        Util.set16be(data, off+9, zval);
    break;
    case 3:
        xval = 5;
        yval = 20;
        zval = 5;
        Util.set16be(data, off+5, xval);
        Util.set16be(data, off+7, yval);
        Util.set16be(data, off+9, zval);
    break;
    case 4:
        xval = 20;
        yval = 10;
        zval = 5;
        Util.set16be(data, off+5, xval);
        Util.set16be(data, off+7, yval);
        Util.set16be(data, off+9, zval);
    break;
    case 5:
        xval = 5;
        yval = 15;
        zval = 5;
        Util.set16be(data, off+5, xval);
        Util.set16be(data, off+7, yval);
        Util.set16be(data, off+9, zval);
    break;
}
// transmit with CSMA to avoid collisions with other packets
Radio.transmit(Radio.TXMODE_CSMA, data, 0,
HDEFS.WDATA_LEN, onTxDone);
if(trcounter<=10 || radiostatus == 1)
{
    RadioTransmit();
}
data[2] = ++seqno; // always increment sequence number
// set up a new timer for sensing data
timer.setAlarmTime(Time.currentTicks() + rate_ticks);
}

internal static void RadioTransmit(){
// transmit with CSMA to avoid collisions with other packets
Radio.transmit(Radio.TXMODE_CSMA, data, 0,
HDEFS.WDATA_LEN, onTxDone);
}

internal static void myTimeRun (byte param, long time)
{
    try {

```

```

        uint temp_numSecs =
        (uint)Time.currentTime(Time.SECONDS);
        numSecs = temp_numSecs%60;
        numMins = temp_numSecs/60;
        numHours = numMins/60;
        if(numMins%5==0)
        {
            myTempRotate ();
        }
    }
    catch (MoteException) {
        LED.setState(2,1);
    }
    mytimer.setAlarmTime(Time.currentTicks() + rate_ticks);
}

internal static void myTempRotate ()
{
    try {
        for(uint x=1; x>0; x--)
        {
            tempdata[x] = tempdata[x-1];
        }
        tempdata[0] = temp;
    }
    catch (MoteException) {
        LED.setState(2,1);
    }
}

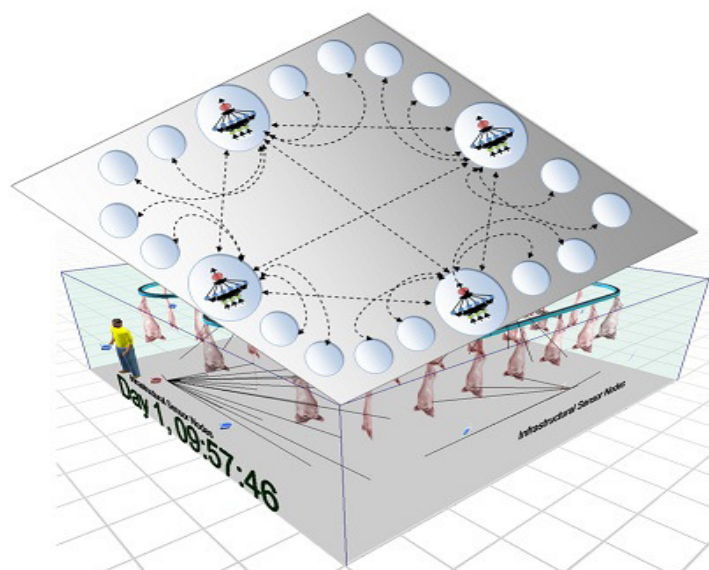
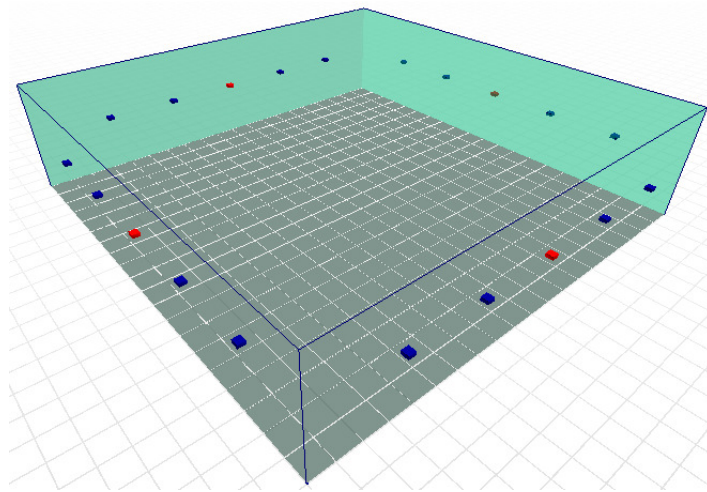
internal static void onTxDone(byte[] pdu, uint len, int status, long
txend) {
    // enable receiver for a very short time
    Radio.enableRx(Time.currentTicks() + 100000);
}
}
}

```

## A.2 Simulation Environment:

### *Three Dimensional Virtual Reality Environments:*

A three dimensional virtual reality space is designed in Flexsim to build the thermal map within that. The red and blue colored nodes are infrastructural nodes having one red node at each side acting as a cluster head. Apart from these nodes situated on the inner surface of the space, the portable nodes can be deployed anywhere in the space by using a designed GUI that controls the model. The designed space volume is  $20 \times 20 \times 5 \text{ m}^3$  and this can be altered by the interface to conduct various scenarios.



### A.2.1 Nodes deployment in simulation environment

//Temperature Limits

**double**

Temp\_min=getnodenum(node("MAIN:/project/model/Tools/globaldata/Tmin"));

**double**

Temp\_max=getnodenum(node("MAIN:/project/model/Tools/globaldata/Tmax"));

//Spatial Coordinates

**double** X\_loc=getnodenum(node("MAIN:/project/model/Tools/globaldata/X"));

**double** Y\_loc=getnodenum(node("MAIN:/project/model/Tools/globaldata/Y"));

**double** Z\_loc=getnodenum(node("MAIN:/project/model/Tools/globaldata/Z"));

//Initialization

**int** knm=getnodenum(node("MAIN:/project/model/Tools/globaldata/kcount"));

**int**

Nodes=getnodenum(node("MAIN:/project/model/Tools/globaldata/numnodes"));

**double**

Threshold=getnodenum(node("MAIN:/project/model/Tools/globaldata/Threshval"));

**treenode** SLocTable = reftable("LocTable");

**string** StrNodes=numtostring(Nodes,0,0);

**int** LocTableCols=gettablecols(SLocTable);

**int** ColRepeat;

**int** ColCount;

**int** ColNext;

**double** XLoc;

**double** YLoc;

**double** ZLoc;

**int** ColRepeatFlag=0;

**int** nCount = round(Nodes/8) + 1;

**int** nFFRows=0;

//Generate Inf. Nodes and locate in its position and set Temperature label

**for**(**int** InfN=1;InfN<=Nodes;InfN++)

{

XLoc=gettablenum("NodeTable",InfN,1);

YLoc=gettablenum("NodeTable",InfN,2);

ZLoc=gettablenum("NodeTable",InfN,3);

**int**

InfNrows=getnodenum(node("MAIN:/project/model/Tools/globaldata/numnodes"))

;

dropuserlibraryobject(node("MAIN:/project/userlibrary/WSN/SNode"));

setname(last(model()),concat("SNode",numtostring(InfN,0,0)));

setloc(last(model()),XLoc,YLoc,ZLoc);

setlabelnum(last(model()),"Temperature",temp\_at(XLoc,YLoc,ZLoc));

```

//Identify and set color to Fully functional node
if(Nodes/4 == 0 || fmod((Nodes/4),2) == 1)
{
    if(InfN == nCount)
    {
        setcolor(last(model()),255,0,0);
        nCount = nCount + Nodes/4;

        //Update FFNodes table Data
        nFFRows = nFFRows+1;
        settablesize("FFNodes",nFFRows,5);
        settablestr("FFNodes",nFFRows,1,getname(last(model())));
        settablenum("FFNodes",nFFRows,2,xloc(last(model())));
        settablenum("FFNodes",nFFRows,3,yloc(last(model())));
        settablenum("FFNodes",nFFRows,4,zloc(last(model())));

        settablenum("FFNodes",nFFRows,5,getlabelnum(last(model()),"Temperature"));
    }
}

//Generate Temp values in Grid Table (NNgrid)
settablesize("NNgrid",1,4);
int rowcount=1;
for(int xval=0;xval<=X_loc;xval++)
{
    for(int yval=0;yval<=Y_loc;yval++)
    {
        for(int zval=0;zval<=Z_loc;zval++)
        {
            settablesize("NNgrid",rowcount,4);
            settablenum("NNgrid",rowcount,1,xval);
            settablenum("NNgrid",rowcount,2,yval);
            settablenum("NNgrid",rowcount,3,zval);
            double Tempval=temp_at(gettablenum("NNgrid",rowcount,1),

gettablenum("NNgrid",rowcount,2),

gettablenum("NNgrid",rowcount,3));
            settablenum("NNgrid",rowcount,4,Tempval);
            rowcount++;
        }
    }
}

for(int x=1;x<=Nodes;x++)
{
    XLoc=uniform(0,X_loc,1);
    YLoc=uniform(0,Y_loc,2);
    ZLoc=uniform(0,Z_loc,3);
    dropuserlibraryobject(node("MAIN:/project/userlibrary/WSN/SNode"));
}

```

```

setname(last(model()),concat("SNode",numtostring(x,0,0)));
setloc(last(model()),XLoc,YLoc,ZLoc);
setlabelnum(last(model()),"Temperature",NNtempXYZ(XLoc,YLoc,ZLoc));

//Create Table and set the node data
treenode SNodesTable = reftable("NodesTable");
settablesize(SNodesTable,x,5);
settableheader(SNodesTable, 1, x, concat("SNode",numtostring(x,0,0)));
settablenum(SNodesTable,x,1,XLoc);
settablenum(SNodesTable,x,2,YLoc);
settablenum(SNodesTable,x,3,ZLoc);
settablenum(SNodesTable,x,4,NNtempXYZ(XLoc,YLoc,ZLoc));//Temperature
Val

}
//Generate Temp values in Grid Table (NNgrid)
settablesize("NNgrid",1,4);
int rowcount=1;
for(int xval=0;xval<=X_loc;xval++)
{
    for(int yval=0;yval<=Y_loc;yval++)
    {
        for(int zval=0;zval<=Z_loc;zval++)
        {
            settablesize("NNgrid",rowcount,4);
            settablenum("NNgrid",rowcount,1,xval);
            settablenum("NNgrid",rowcount,2,yval);
            settablenum("NNgrid",rowcount,3,zval);
            double
Tempval=NNtempXYZ(gettablenum("NNgrid",rowcount,1),

            gettablenum("NNgrid",rowcount,2),

            gettablenum("NNgrid",rowcount,3));
            settablenum("NNgrid",rowcount,4,Tempval);
            rowcount++;
        }
    }
}
//Generate Nodes and locate in its position and set Temperature label
for(int x=1;x<=Nodes;x++)
{
    //For data/location consistency maintain another location table
    //to keep the location constant whenever choosing the same number of nodes
    int colflag;
    if(ColRepeatFlag==1)
    colflag=1;
    else
    colflag=0;

    //Set Temperature value
    double TempVal=uniform(Temp_min,Temp_max,1);

```

```

//Check if the number of nodes already exists so that can't repeat another set of
coordinates for the same no. of nodes
for(int cols=1;cols<=LocTableCols;cols++)
{
    string ColName=gettableheader(SLocTable,2,cols);
    if(comparetext(StrNodes,ColName)==1)
    {
        ColRepeat=1;
        ColCount=cols;
        break;
    }
    else
    {
        ColRepeat=0;
    }
}

//Check the no. of nodes are repeating no need to verify the next column to start
if(ColRepeat==0)
{
    for(int colsnext=1;colsnext<=LocTableCols;colsnext++)
    {
        string ColNextName=gettableheader(SLocTable,2,colsnext);
        if(comparetext("Col",stringcopy(ColNextName,1,3))==1)
        {
            ColNext=colsnext;
            break;
        }
    }
}

//Repeat the same coordinate set for the given node numbers ELSE generate new
set of coordinates for given no.of nodes
if(ColRepeat==1&&colflag==0)
{
    XLoc=gettablenum(SLocTable,x,ColCount);
    YLoc=gettablenum(SLocTable,x,ColCount+1);
    ZLoc=gettablenum(SLocTable,x,ColCount+2);
    ColNext=ColCount;
}
else
{
    ColRepeatFlag=1;
    XLoc=uniform(0,X_loc,1);
    YLoc=uniform(0,Y_loc,2);
    ZLoc=uniform(0,Z_loc,3);
}

//Fill the coordinates in location table to keep track of the data for next time use.
settableheader(SLocTable,2,ColNext,numtostring(Nodes,0,0));

```

```

settablenum(SLocTable,x,ColNext,XLoc);
settableheader(SLocTable,2,ColNext+1," ");
settablenum(SLocTable,x,ColNext+1,YLoc);
settableheader(SLocTable,2,ColNext+2," ");
settablenum(SLocTable,x,ColNext+2,ZLoc);

dropuserlibraryobject(node("MAIN:/project/userlibrary/WSN/SNode"));
setname(last(model()),concat("SNode",numtostring(x,0,0)));
setloc(last(model()),XLoc,YLoc,ZLoc);
setlabelnum(last(model()),"Temperature",TempVal);

//Create Table and set the node data
treenode SNodesTable = reftable("NodesTable");

settablesize(SNodesTable,x,5);
settableheader(SNodesTable, 1, x, concat("SNode",numtostring(x,0,0)));
settablenum(SNodesTable,x,1,XLoc);
settablenum(SNodesTable,x,2,YLoc);
settablenum(SNodesTable,x,3,ZLoc);
settablenum(SNodesTable,x,4,TempVal);//Temperature Val
}

//Calculate k-Nearest Nodes to each node and set their distances into labels
for(int y=1;y<=Nodes;y++)
{
    treenode
    SenNodeCur=node(concat("MAIN:/project/model/", "SNode",numtostring(y,0,0)));
    treenode Stable=reftable("SortTable");
    for(int z=1;z<=Nodes;z++)
    {
        treenode
        SenNodeTo=node(concat("MAIN:/project/model/", "SNode",numtostring(z,0,0)));
        settablesize(Stable,z,2);
        double dist=sqrt(sqr(xloc(SenNodeCur)-
xloc(SenNodeTo))+sqr(yloc(SenNodeCur)-
yloc(SenNodeTo))+sqr(zloc(SenNodeCur)-zloc(SenNodeTo)));
        settablenum(Stable,z,1,dist);
        settablestr(Stable,z,2,getname(SenNodeTo));
    }
    sorttable(Stable,1);
    treenode knode=node(">labels/knn",SenNodeCur);

    for(int k=1;k<=knn;k++)
    {
        setnodename(rank(knode,k),gettablestr(Stable,k+1,2));
        setnodenum(rank(knode,k),gettablenum(Stable,k+1,1));
    }
}

//Set OptTable Size and Fill Datatype
treenode OpNodesTable = reftable("OptTable");

```



```

for(int Op=1;Op<=Nodes+1;Op++)
{
    treenode
OpNodeCur=node(concat("MAIN:/project/model/", "SNode", numtostring(Op,0,0)))
;

    settablesize(OpNodesTable,Nodes,Nodes,DATATYPE_NUMBER);
    settableheader(OpNodesTable, 1, Op, concat("SNode",numtostring(Op,0,0)));
    settableheader(OpNodesTable, 2, Op, concat("SNode",numtostring(Op,0,0)));
    if(Op==Nodes+1)
    {

        settablesize(OpNodesTable,Nodes+1,Nodes+1,DATATYPE_NUMBER);
        settableheader(OpNodesTable, 1, Op, "WNodes");
        settableheader(OpNodesTable, 2, Op, "INodes");

    }
}
//Find the first 2 nearest nodes in each node and set "1" in opttable
for(int F2kn=1;F2kn<=Nodes;F2kn++)
{
    treenode
k2SenNodeCur=node(concat("MAIN:/project/model/", "SNode", numtostring(F2kn,
0,0)));
    for(int k2Temp=1;k2Temp<=2;k2Temp++)
    {
        string
Op2KnodeTo=getname(rank(node(">labels/knn",k2SenNodeCur),k2Temp));
        for(int Rowt=1;Rowt<=Nodes;Rowt++)
        {

            if(comparetext(gettableheader("OptTable",1,Rowt),Op2KnodeTo)==1)
            {
                settablenum("OptTable",Rowt,F2kn,1);
                break;
            }
        }
    }
}

//Evaluate all nodes and compute its Temperature if that node is absent
for(int j=1;j<=Nodes;j++)
{
    treenode
CSenNode=node(concat("MAIN:/project/model/", "SNode", numtostring(j,0,0)));
    double PredTemp=TempXYZ(CSenNode);
    setlabelnum(CSenNode,"PTemp",PredTemp);
}

//Refresh the window to update all the labels
forobjecttreeunder(node("VIEW:/1"))
{
    repaintview(a);
}

```

```

}
repaintall();

```

### A.2.2 Nodes reset to initialize code

```

//Destroy the nodes created from deployment
int index;
treenode objtree=node("MAIN:/project/model");
int Totnodes=content(objtree);
int
Numnodes=getnenum(node("MAIN:/project/model/Tools/globaldata/numnodes"
));
for(int Ntimes=1;Ntimes<=Numnodes;Ntimes++)
{
    for(index = 1; index <= Totnodes; index++)
    {
        string nodename=stringcopy(getname(rank(objtree,index)),1,5);
        if(comparetext("SNode", nodename))
        {
            destroyobject(rank(objtree,index));
        }
        Totnodes=content(objtree);
        //pd(index);pr();
    }
}
//Delete table rows
treenode SNodesTable = reftable("NodesTable");
settablesize(SNodesTable,1,5);
clearglobaltable(SNodesTable);

settablesize("NNRandData",1,4);

//Set Min nodes to Zero
setnenum(node("MAIN:/project/model/Tools/globaldata/Optnum"),0);
//Set Hotspot temp table (NodeTable)
int
Infnodes=getnenum(node("MAIN:/project/model/Tools/globaldata/numnodes"));
//Spatial Coordinates
double X_loc=getnenum(node("MAIN:/project/model/Tools/globaldata/X"));
double Y_loc=getnenum(node("MAIN:/project/model/Tools/globaldata/Y"));
double Z_loc=getnenum(node("MAIN:/project/model/Tools/globaldata/Z"));
settablesize("NodeTable",Infnodes,5);
int Infcase=1;
int casein=1;
int nodddiv=Infnodes/4+1;
for(int Infrow=1;Infrow<=Infnodes;Infrow++)
{
    settableheader("NodeTable", 1, Infrow,
concat("InfNode",numtostring(Infrow,0,0)));

```

```

switch(Infcase)
{
    case 1:
    {
        settablenum("NodeTable",Infrow,1,(casein/noddiv)*X_loc);
        settablenum("NodeTable",Infrow,2,0*Y_loc+0.1);
        settablenum("NodeTable",Infrow,3,(1/2)*Z_loc);

        settablenum("NodeTable",Infrow,4,temp_at(gettablenum("NodeTable",Infrow,1
),

        gettablenum("NodeTable",Infrow,2),

        gettablenum("NodeTable",Infrow,3)));
        settablenum("NodeTable",Infrow,5,1);
        casein++;
        break;
    }
    case 2:
    {
        settablenum("NodeTable",Infrow,1,(casein/noddiv)*X_loc);
        settablenum("NodeTable",Infrow,2,1*Y_loc);
        settablenum("NodeTable",Infrow,3,(1/2)*Z_loc);

        settablenum("NodeTable",Infrow,4,temp_at(gettablenum("NodeTable",Infrow,1
),

        gettablenum("NodeTable",Infrow,2),

        gettablenum("NodeTable",Infrow,3)));
        settablenum("NodeTable",Infrow,5,1);
        casein++;
        break;
    }
    case 3:
    {
        settablenum("NodeTable",Infrow,1,0*X_loc+0.1);
        settablenum("NodeTable",Infrow,2,(casein/noddiv)*Y_loc);
        settablenum("NodeTable",Infrow,3,(1/2)*Z_loc);

        settablenum("NodeTable",Infrow,4,temp_at(gettablenum("NodeTable",Infrow,1
),

        gettablenum("NodeTable",Infrow,2),

        gettablenum("NodeTable",Infrow,3)));
        settablenum("NodeTable",Infrow,5,1);
        casein++;
        break;
    }
    case 4:
    {

```

```

        settablenum("NodeTable",Infrow,1,1*X_loc);
        settablenum("NodeTable",Infrow,2,(casein/noddiv)*Y_loc);
        settablenum("NodeTable",Infrow,3,(1/2)*Z_loc);

    settablenum("NodeTable",Infrow,4,temp_at(gettablenum("NodeTable",Infrow,1
),

        gettablenum("NodeTable",Infrow,2),

        gettablenum("NodeTable",Infrow,3)));
        settablenum("NodeTable",Infrow,5,1);
        casein++;
        break;
    }
    default:
    {
        break;
    }
}
if(fmod(Infrow,Infnodes/4)==0)
{
    Infcase++;
    casein=1;
}
}

settablesize("NNgrid",1,4);
treenode Optr=node("MAIN:/project/model/Operator");
setloc(Optr,0,0,0);

//Reset FFNodes Table//
settablesize("FFNodes",1,5);

//Fixed Temp location setting
treenode FixedT = reftable("FixedTemp");

settablenum(FixedT,1,1,0);
settablenum(FixedT,1,2,0);
settablenum(FixedT,1,1,Z_loc/2);
settablenum(FixedT,2,1,X_loc/2);
settablenum(FixedT,2,2,0);
settablenum(FixedT,2,3,0);
settablenum(FixedT,3,1,X_loc/2);
settablenum(FixedT,3,2,0);
settablenum(FixedT,3,3,Z_loc);
settablenum(FixedT,4,1,X_loc);
settablenum(FixedT,4,2,0);
settablenum(FixedT,4,3,Z_loc/2);
settablenum(FixedT,5,1,X_loc);
settablenum(FixedT,5,2,Y_loc/2);
settablenum(FixedT,5,3,0);

```

```

settablenum(FixedT,6,1,X_loc);
settablenum(FixedT,6,2,Y_loc/2);
settablenum(FixedT,6,3,Z_loc);
settablenum(FixedT,7,1,X_loc);
settablenum(FixedT,7,2,Y_loc);
settablenum(FixedT,7,3,Z_loc/2);
settablenum(FixedT,8,1,X_loc/2);
settablenum(FixedT,8,2,Y_loc);
settablenum(FixedT,8,3,0);
settablenum(FixedT,9,1,X_loc/2);
settablenum(FixedT,9,2,Y_loc);
settablenum(FixedT,9,3,Z_loc);
settablenum(FixedT,10,1,0);
settablenum(FixedT,10,2,Y_loc);
settablenum(FixedT,10,3,Z_loc/2);
settablenum(FixedT,11,1,0);
settablenum(FixedT,11,2,Y_loc/2);
settablenum(FixedT,11,3,0);
settablenum(FixedT,12,1,0);
settablenum(FixedT,12,2,Y_loc/2);
settablenum(FixedT,12,3,Z_loc);

```

```

forobjecttreeunder(node("VIEW:/1"))
{
repaintview(a);
}
repaintall();

```

### *A.2.3 Grid generation code in simulation*

```

//Generating Grid Table coords based on grid size and number of points required
double gridsize;
double locx;
double locy;
double locz;
int gridpoints=5;
treenode objectview=node("VIEW:/active/ortho",views());
if(objectexists(objectview))
{
    gridsize=getnodenum(gridx(objectview));
    pf(gridsize);pr();
}
else
{
    gridsize=1;
}
treenode firstobject=node("MAIN:/project/model/Node1",model());
if(objectexists(firstobject))
{
    locx=xloc(firstobject);

```

```

        locy=yloc(firstobject);
        locz=zloc(firstobject);
    }
    else
    {
        locx=0;locy=0;locz=0;
    }
    int cols=gridpoints;
    int rows=gridpoints;
    //Coordinate phase change matrix
    int numbox=gridpoints-1;
    intarray phasechange=makearray(numbox);
    for(int fcount=1;fcount<=numbox;fcount++)
    {
        phasechange[fcount]=numbox*fcount-1;
    }
    //Fill total grid matrix
    int totgridpoints=cols*rows;
    //Create array to store temperature data
    doublearray tempdata = makearray((cols+1)*(rows+1));

    int pointcount=1;
    for(int rowcount=1;rowcount<=rows;rowcount++)
    {
        for(int colcount=1;colcount<=cols;colcount++)
        {
            settablenum("GridTable",pointcount,1,locx);
            settablenum("GridTable",pointcount,2,locy);
            settablenum("GridTable",pointcount,3,locz);

            settablenum("GridTable",pointcount,4,temp_at(gettablenum("GridTable",pointc
ount,1),

            gettablenum("GridTable",pointcount,2),

            gettablenum("GridTable",pointcount,3)

            ));

            locx=locx+gridsize;
            pointcount++;
        }
        locx=0;
        locy=locy+gridsize;
    }
}

```

#### A.2.4 Internal functions code

```

double x=parval(1);
double y=parval(2);
double z=parval(3);

```

```

double dist;
double sum_Invdist;
double prod_DT;
double temp_Point;
int numtabrows=gettablerows("FixedTemp");
doublearray sq_dist_array=makearray(numtabrows);
doublearray inv_dist_array=makearray(numtabrows);

for(int j=1; j<=numtabrows;j++)
{
    if(gettablenum("FixedTemp",j,5)==1)
    {
        dist=sqrt(sqr(gettablenum("FixedTemp",j,1)-
x)+sqr(gettablenum("FixedTemp",j,2)-y)+sqr(gettablenum("FixedTemp",j,3)-z));
        sq_dist_array[j]=sqr(dist);
        if(dist==0)
        {
            inv_dist_array[j]=10000000;
        }
        else
        {
            inv_dist_array[j]=1/dist;
        }
        sum_Invdist=sum_Invdist+inv_dist_array[j];
        Gsum_Invdist=sum_Invdist;
    }
    else
    {
        dist=0;
        sq_dist_array[j]=sqr(dist);

        inv_dist_array[j]=0;
        sum_Invdist=sum_Invdist+inv_dist_array[j];
        Gsum_Invdist=sum_Invdist;
    }
}

//pf(Gsum_Invdist);pr();
doublearray weight_array=makearray(numtabrows);
for(int k=1;k<=numtabrows;k++)
{
    if(gettablenum("FixedTemp",k,5)==1)
    {
        weight_array[k]=inv_dist_array[k]/Gsum_Invdist;
    }
    else
    {
        weight_array[k]=inv_dist_array[k]/Gsum_Invdist;
    }
}

```

```

for(int l=1;l<=numtabrows;l++)
{
    if(gettablenum("FixedTemp",l,5)==1)
    {
        prod_DT=prod_DT+weight_array[l]*gettablenum("FixedTemp",l,4);
    }
    else
    {
        prod_DT=prod_DT+weight_array[l]*gettablenum("FixedTemp",l,4);
    }
}
return prod_DT;

```

#### *A.2.5 Nodes minimization approach*

```

treenode Snode=parnode(1);
int KNcount=parval(2);

double xi=xloc(Snode);
double yi=yloc(Snode);
double zi=zloc(Snode);

double Sdist;
double inv_dist;
double sum_Invdist;
double weight_node;
double TempVal;

treenode knode=node(">labels/knn",Snode);
int numlabelrows=content(knode);
int Rlabelrows;
for(int Vn=1;Vn<=numlabelrows;Vn++)
{
    if(getnodenum(rank(knode,Vn))>0)
    {
        Rlabelrows=Rlabelrows+1;
    }
    else
    {
        break;
    }
}

//Calculate denominator(sum of inverse distances) of weight function
for(int ji=1; ji<=KNcount;ji++)
{
    Sdist=getnodenum(rank(knode,ji));
    inv_dist=1/Sdist;
    sum_Invdist=sum_Invdist+inv_dist;
}

```



```

}

//Calculate weights for each node
doublearray weight_array=makearray(KNcount);
for(int ki=1;ki<=KNcount;ki++)
{
    Sdist=getnodenum(rank(knode,ki));
    inv_dist=1/Sdist;
    weight_array[ki]=inv_dist/sum_Invdist;
}

//Calculate estimated temperature at X,Y,Z
for(int li=1;li<=KNcount;li++)
{
    treenode Tnode=node(concat("/",getname(rank(knode,li))),model());
    double NodeTempVal=getlabelnum(Tnode,"Temperature");
    TempVal=TempVal+weight_array[li]*NodeTempVal;
}
return TempVal;

```

#### *A.2.6 Portable nodes object's update*

```

treenode current = ownerobject(c);
treenode view = parnode(1);

/**Update Node data*/
//Set the node Info...Temperature and Spatial data...
int nodes = 3;
string SNodeName=getname(current);
treenode SNodeTableRow=
node(concat("MAIN:/project/model/Tools/GlobalTables/NodesTable>variables/dat
a/",SNodeName));

//Set label status for Running/Redundant
setnodenum(rank(SNodeTableRow,5),getlabelnum(current,"status"));

treenode displaynodes = objectinfo(current);
while(nodes>content(displaynodes))
{
    nodeinsertinto(displaynodes);
}

while(nodes<content(displaynodes))
{
    destroyobject(last(displaynodes));
}

if(getlabelnum(current,"status")==1)
{
    setname(rank(displaynodes,1),concat("Status: ","Running"));
}

```

```

        setcolor(current,0,0,255);
    }
    else
    {
        setname(rank(displaynodes,1),concat("Status: ","Stopped"));
        setcolor(current,255,0,0);
    }

    setname(rank(displaynodes,2),concat("Temp:
",numtostring(getlabelnum(current,"Temperature"),0,2)));
    setname(rank(displaynodes,3),concat("X:",numtostring(xloc(current),0,2)," ",

        "Y:",numtostring(yloc(current),0,2)," ",

        "Z:",numtostring(zloc(current),0,2)
    ));

    double x_grid=0;
    double y_grid=0.5;
    double gridcount=1/y_grid;
    drawtomodelscale(current);

    glBegin(GL_LINES);
    double xs=x_grid;
    double ys=y_grid;
    double ks=0;

    for(int k=0;k<=4;k++)
    {
        for(int j=1;j<=4;j++)
        {
            glColor3d(color_at(gettablenum("GridTable",1,4),1),color_at(gettablenum("GridTable",2,4),2),color_at(gettablenum("GridTable",3,4),3));

            glVertex3d(xs,0,-ks);
            glVertex3d(ys,0,-ks);
            glVertex3d(ys,0,-ks-y_grid);
            glVertex3d(xs,0,-ks-y_grid);
            xs=xs+y_grid;
            ys=ys+y_grid;
        }
        xs=x_grid;
        ys=y_grid;
        ks=k*ys;
    }
    glEnd();

```

### A.3 GUI for Nodes Minimization

Selected Node : Start/Stop Range : Disable

**Model Setup**

k-Neighbours No : 10 Room Temperature Limits :  
Tmin : 20  
No. of Nodes : 20 Locate Tmax : 25  
☒ Evenly Distribute on Grid Size 1 Threshold : 0.5

Space Coordinates : X : 20 Y : 20 Z : 5

Deploy Nodes Reset Opt

Apply Close

### A.4 GUI for Setup Scenarios

**Model Setup**

No. of Infrastructural Nodes : 12

Space Coordinates : X : 20 Y : 20 Z : 5.00

No. of Training Points: 80 Train GenData

Deploy Nodes Reset

Apply Close