UNIVERSITY OF CINCINNATI

Date: _____

I, _____,
hereby submit this work as part of the requirements for the degree of:

in:

It is entitled:

This work and its defense approved by:

Chair: _____

Self Organized Inference of Spatial Structure in Randomly Deployed Wireless Sensor Networks

A thesis submitted to the
Division of Research and Advanced Studies
of
The University of Cincinnati
in the partial fulfillment
of the requirements of the degree of
Master of Science
in the Department of Electrical and Computer Engineering
and Computer Science
of
The College of Engineering

November 2006
by
Neena George
B.Tech. (Electrical and Electronics Engineering)
College of Engineering, Trivandrum, Kerala
University of Kerala, India 2002

Thesis Advisor and Committee Chair: **Dr. Ali Minai**

Abstract

Randomly deployed wireless sensor networks are becoming increasingly viable for applications such as environmental monitoring, battlefield awareness, tracking and smart environments. Such networks can comprise anywhere from a few hundred to thousands of sensor nodes, and these sizes are likely to grow with advancing technology, making scalability a primary concern. Each node in these sensor networks is a small unit with limited resources and localized sensing and communication. Thus, all global tasks must be accomplished through self-organized distributed algorithms, which also lead to improved scalability, robustness and flexibility. In this paper, we examine the use of distributed algorithms to infer the spatial structure of an extended environment monitored by a self organizing sensor network. Based on its sensing, the network segments the environment into regions with distinct characteristics, thereby inferring a cognitive map of the environment. This, in turn, can be used to answer global queries about the environment efficiently and accurately. The main challenge to the network arises from the necessarily irregular spatial sampling and the need for totally distributed computation. We consider distributed machine learning techniques for segmentation. We also present a heuristic for segmenting within boundaries to obtain distinct segments and study the variation of segmentation quality with reconstruction at different node densities and in environments of varying complexity. The eventual goal of the work presented in this paper is to obtain

intelligent networks capable of autonomous reconfiguration based on their observations. The inference of spatial structure in monitored environments is clearly an essential first step for such self-reconfigurability.

Acknowledgements

As I reach the end of my stay at the University of Cincinnati, I would like to take this opportunity to thank everyone who has made this sojourn a memorable one.

First, I would like to thank my thesis advisor, Dr. Ali A. Minai for the excellent guidance and support provided to me during the course of my thesis work. He has been an amazingly patient advisor and has always encouraged me to push my limits and I can say that working with him has been a truly rewarding experience. I am glad to have had my first insights into research under his guidance, which has sparked a keen interest in research in me. Thank you Dr. Minai, for being an exceptional teacher and advisor.

Next, I would like to thank Dr. Raj Bhatnagar and Dr. Carla Purdy for agreeing to be on my thesis committee and for having taught me at various stages of my graduate studies. I would also like to thank my employers at the Office of College Computing, Jack and Dave for being very flexible with my schedule, thus helping me to balance both work and thesis work. A big thanks to the whole gang, Vaishali, Srinu, Kiran, Balaji and Manjari for all the fun times there. I am glad to have met a very special and truly nice person there, Vaishu, who has been a great friend and sounding board through the past one year. I am going to miss our coffee outings on thursdays, and our long, apparently crazy, but insightful discussions on anything and everything, be it academics, work, books, religion, or our philosophies on life. Thanks Vaishu, for being such a nice person.

I would like to thank my labmates, Sarjoun, DC, Abhinay and Payal for all the little ways in which they have helped me at various times. Sarjoun has been especially encouraging. Thanks a bunch Sarjoun!

It has hard not to make a mention of the friends I made at UC, who have all helped me out at one time or the other. I shall not attempt to name them all, lest I miss out on someone.

But most of all, I would like to thank my family for being very supportive and seeing me through my ups and downs. To Appa and Amma, thank you for all your love and support. To my sisters, Meena and Seená, thank you for always being there for me. I still look up to you guys. Thanks to my chach, Sam, for being a great brother, for all the motivation, and for always making me feel like a special little sister. A very special thanks to my niece, Nikita, who never fails to make me laugh.

Last, but most importantly, thanks to God for everything.

Contents

Abstract	i
Acknowledgements	iii
1 Introduction	1
1.1 Information Processing In Sensor Networks	4
1.1.1 Centralized Processing:	4
1.1.2 In-Network Processing:	5
1.2 Segmentation In Sensor Networks	6
1.2.1 Types of Segmentation	7
1.2.2 The Utility of Segmentation	10
1.3 Organization of the Thesis	12
2 Background and Related Work	13
2.1 Segmentation in Image Processing	13
2.1.1 Edge Detection Methods	14
2.1.2 Region Growing Methods	15
2.2 Image Processing and Sensor Networks	17
2.3 Information Processing in Sensor Networks	18

3	Environment Models and Algorithms	21
3.1	Environment Models	21
3.2	Sensor Model Used	24
3.3	Segmentation Process and Algorithms	25
3.3.1	Initial Segmentation	25
3.3.2	Map Formation	29
3.3.3	Inference for Un-Sensed cells	32
3.3.4	Conflict Resolution and Reassignment	34
3.3.5	Segment Refinement	36
3.3.6	Reorganizing the Node-Maps: Ensuring Load-Balancing or Load-Sharing between Nodes	39
4	Results	41
4.1	Performance Measure	41
4.2	Result Analysis	42
4.2.1	Track Environment	42
4.2.2	Demonstration of accuracy of Segment Inference: The KNN and IDW Algorithms	45
4.2.3	Segment Refinement	46
4.3	Effect of Environment Complexity and System Parameters	47
4.3.1	The Spiral Environment	48
4.3.2	The Patch Environment Type	66
4.4	Discussion	80
5	Conclusions and Future Work	85

List of Tables

List of Figures

3.1	The Track Environment	22
3.2	The Spiral Type One Environment	23
3.3	The Spiral Type Two Environment	24
3.4	The Spiral Type Three environment	25
3.5	The Patch Type One Environment	26
3.6	The Patch Type Two Environment	27
3.7	The Patch Type Three Environment	28
4.1	The Track Environment	43
4.2	Node classification after running initial segmentation algorithm	44
4.3	Map of environment obtained after running the initial segmentation algorithm	45
4.4	Map of an individual node	46
4.5	Map of multiple nodes	47
4.6	Environment map obtained after running the KNN algorithm	48
4.7	Environment map obtained after running the IDW algorithm	49
4.8	Environment map obtained after running the Segment Refinement algorithm	50
4.9	The Spiral Type One Environment	51
4.10	Accuracy plot obtained for Spiral Type One , with k=3	52

4.11	Accuracy plot obtained for Spiral Type One, with $k=5$	53
4.12	Accuracy plot obtained for Spiral Type One , with $k=7$	53
4.13	Result of Segmentation for Spiral Type One at node density 0.8, $k=3$. . .	54
4.14	Result of segmentation for Spiral Type One at node density 0.6, $k=3$. . .	54
4.15	Result of segmentation for Spiral Type One at node density 0.4, $k=3$. . .	55
4.16	Result of segmentation for Spiral Type One at node density 0.2, $k=3$. . .	55
4.17	The Spiral Type Two Environment	56
4.18	Accuracy plot obtained for Spiral Type Two, with $k=3$	56
4.19	Accuracy plot obtained for Spiral Type Two , with $k=5$	57
4.20	Accuracy plot obtained for Spiral Type Two, with $k=7$	57
4.21	Result of segmentation for Spiral Type Two, at node density=0.8, $k=3$. .	58
4.22	Result of segmentation for Spiral Type Two, at node density=0.6, $k=3$. .	58
4.23	Result of segmentation for Spiral Type Two, at node density=0.4, $k=3$. .	59
4.24	Result of segmentation for Spiral Type Two, node density=0.2, $k=3$. An example of good segmentation at low densities.	59
4.25	Result of segmentation for Spiral Type Two, at node density=0.2, $k=3$. An example of bad segmentation at low densities.	60
4.26	The Spiral Type Three environment	60
4.27	Accuracy plot obtained for Spiral Type Three Environment, $k=3$	61
4.28	Accuracy plot obtained for Spiral Type Three Environment, $k=5$	62
4.29	Accuracy plot obtained for Spiral Type Three Environment, $k=7$	63
4.30	Result of segmentation for Spiral Type Three, node density=0.8, $k=3$. . .	64
4.31	Result of segmentation for Spiral Type Three, node density=0.6, $k=3$. . .	65
4.32	Decrease of performance with increase in environment complexity, for each value of node density, using the KNN algorithm for cell inference, at $k=3$.	66

4.33	Decrease of performance with increase in environment complexity, for each value of node density, using the IDW algorithm for cell inference, at $k=3$	67
4.34	Effect of k on segmentation accuracy	68
4.35	The Patch Type One Environment	69
4.36	Accuracy plot obtained for Patch Type One, with $k=3$	69
4.37	Accuracy plot for Patch Type One, with $k=5$	70
4.38	Accuracy plot obtained for Patch Type One, with $k=7$	70
4.39	Result of segmentation for Patch Type One, at node density=0.8, $k=3$	71
4.40	Result of segmentation for Patch Type One, at node density=0.6, $k=3$	71
4.41	Result of segmentation for Patch Type One, at node density=0.4, $k=3$	72
4.42	Result of segmentation for Patch Type One, at node density=0.2, $k=3$	72
4.43	Patch Type Two	73
4.44	Accuracy plot obtained for Patch Type Two, with $k=3$	73
4.45	Accuracy plot obtained for Patch Type Two, with $k=5$	74
4.46	Accuracy plot obtained for Patch Type Two, with $k=7$	74
4.47	Result of segmentation, Patch Type Two, at node density 0.8, $k=5$	75
4.48	Result of segmentation for Patch Type Two, at node density 0.6, $k=5$	76
4.49	Result of segmentation for Patch Type Two, at node density 0.4, $k=5$	76
4.50	Result of segmentation for Patch Type Two, at node density 0.2, $k=5$	77
4.51	Result of segmentation for Patch Type Two, at node density 0.2, $k=5$	77
4.52	Patch Type Three	78
4.53	Accuracy plot obtained for Patch Type Three with $k=3$	79
4.54	Accuracy plot obtained for Patch Type Three with $k=5$	80
4.55	Accuracy plot obtained for Patch Type Three with $k=7$	81
4.56	Result of segmentation for Patch Type Three at node density 0.8, $k=3$	81

4.57	Result of segmentation for Patch Type Three at node density 0.6, $k=3$	82
4.58	Result of segmentation for Patch Type Three at node density 0.4, $k=3$	82
4.59	Result of segmentation for Patch Type Three at node density 0.2, $k=3$	83
4.60	Variation of segmentation accuracy with complexity at $k=3$ for the patch environments	83
4.61	Variation of segmentation accuracy with complexity at $k=5$ for the patch environments	84

Chapter 1

Introduction

Wireless Sensor Networks (WSNs) are networked groups of spatially distributed nodes, which are used to gather and process large amounts of data over extended environments. Each sensor node is a small electronic device, usually equipped with sensors, a processor and a transceiver on a miniature chip giving it a limited amount of power, sensing, communication and computation capabilities [27]. This enables a node to act as a self sustaining, independent unit, that is aware of its immediate environment and capable of interacting with other nodes over a short range. The nodes observe the variables or phenomena of interest at very close ranges [2], while networking these nodes helps in harnessing the collective power of the nodes over a larger area to perform higher level information processing tasks [12], [1], [3]. e.g., inference of global attributes for observed phenomena.

A further extension of network functionality, which enhances its usefulness, occurs when the nodes not only infer environment characteristics, but also make appropriate responses based on these inferences. For example, an individual heat sensor would be able to sense the temperature of its location, while a number of such sensors would be

able to collectively decide whether they lie within the region of a fire or not. If detected to be within such a region, they would respond by turning on the closest fire extinguisher.

This capability of a sensor network to interact with its physical environment [9], make inferences based on the data gathered and respond to it is characteristic of intelligent sensor networks, enabling the use of such networks in areas as diverse as environment monitoring [25], habitat monitoring, border surveillance, contaminant tracking, and health monitoring. Nodes are typically deployed in large numbers, not only due to the requirement of pervasive computing in their areas of application [11], but also because of the technological advances which have made the cost effective manufacturing of these nodes possible. The sheer number of these deployed nodes has given rise to concerns such as the scalability of processing algorithms and robustness of the network. These systems are very similar to the large scale, distributed networks like the Internet, but are subject to geographical and computational constraints in the design of network algorithms and protocols [34]. Some of the constraints on sensor nodes are

1. **Energy Constraints:** Sensor nodes typically do not have a renewable source of energy, but are powered by a battery. This fixed amount of energy stored at each node has to cover different needs such as inter node communication, on-board signal processing and other sensor operations. This constraint has a significant impact on the design and functionality of a sensor nodes from the volume of data it can transfer over a certain distance to the amount of information that it can process. Sensor nodes might often be deployed in remote areas where re-servicing a node might be difficult. Also, recharging a large number of sensor nodes is time consuming and expensive. Power conservation and power management are serious issues that have to be kept in mind in the design of sensor network algorithms and protocols.

2. **Communication Constraints:** The process of communication is quite energy intensive and is also hampered by the limited bandwidth available to the nodes and frequent packet drops which reduces the quality of wireless service available.
3. **Computational Constraints:** The small size of the sensor node enables it to have only limited memory and computational power, which not only restricts the algorithms that can be executed by the nodes but also the size of the results that can be stored within the individual nodes.
4. **Fault Tolerance:** Sensor nodes are prone to failure and may stop functioning due to the non-availability of power, physical damage or environmental issues. The network should be able to perform in spite of node failures. Fault tolerance can be defined as the ability of the network to sustain functionality in spite of node failure. Network self-organization is one way to get around node-failure issues. Self-organization is the process by which the nodes locally organize themselves to achieve a global optimization criterion. Nodes can periodically reconfigure themselves to maintain overall high connectivity in the network.
5. **Scalability:** Sensor node deployment in an environment may range from a few hundred to thousands or even millions. Sensors might even be additionally deployed in a region depending on the necessity of doing so. New or existing schemes should be able to work with very large numbers of nodes and should be able to scale well with increasing node densities. Clustering is one way to deal with scalability issues. In this type of node management, a number of neighboring nodes form a group and elect a node, called the cluster-head, to manage them.

Design of network algorithms, protocols and mode of information processing should take into account these constraints for maximal utilization of system capabilities [7].

1.1 Information Processing In Sensor Networks

Information processing can be defined broadly as the conversion of latent information into manifest information [10]. Taking the earlier example of heat sensors, the existence of a fire is the implicit or latent information present in the network. But each sensor is aware only of the temperature it senses within its immediate area. The information about the fire becomes explicit or manifest to the network only after the data gathered from all the sensors is processed. Broadly, there are two types of information processing done in sensor networks.

1.1.1 Centralized Processing:

Centralized processing is the traditional method of information processing, which involves transmission of the data gathered by the individual sensor nodes back to a central base station through some special nodes called gateway nodes. In this type of processing, sensor nodes are seen as mere dumb data gathering devices and interpretive functions such as pattern recognition, segmentation, inference, edge-detection, etc are done by back-end computers by running powerful centralized algorithms. Centralized processing is disadvantageous due to the following reasons.

- Sensor nodes are power-constrained and band-width constrained. Transmission of this high resolution data over large distances to the base stations is energy intensive which is detrimental to the life span of the network. Moreover, the quality of data transmitted is affected by the constraints on communication.
- Centralized methods are non-robust and non-scalable.
- Sensor nodes gather massive amounts of data out of which only a fraction might be

relevant to the queries to be processed. This again presents a needless communication overhead.

- Response to a phenomenon might be significantly delayed due to a time lag between the transmission of data to the base station, processing and transmission of query response the back to network.

1.1.2 In-Network Processing:

In-network processing refers to a method of decentralized, distributed type of processing, which involves moving most of the computing back into the network itself. Sensor nodes are no longer mere data gathering devices, but are used as computing devices in a distributed computing system, improving the functionality of the network and making the network an autonomous system, capable of providing high level information instead of just raw data. Nodes collaborate with their neighbors, data is exchanged and phenomena of interest inferred. Here communication and processing are done as distributedly and as locally as possible, thus considerably reducing communication costs. Decentralization of information processing is advantageous in that it makes the network highly scalable and robust to node failure to a certain degree. The other advantages offered by in-network computation are faster detection of phenomena and possible in-network actuation. In-network actuation refers to the process in which the network discovers the phenomenon of interest and triggers the response on its own. e.g., detecting the phenomenon of a fire and turning on the fire extinguishers. This gives the network a faster response time. Distributed processing enables the integration of information gathered from multiple sensors into a larger network. Measurements from nearby sensors are likely to be highly correlated and so the effective dimensionality of the data is significantly lower than the total number

of sensor measurements. Data reduction can be done by a process of data aggregation. Though in a perfect system, data is reduced as much as possible and as early as possible, providing only the high end result, sufficient care should be taken so as to not lose information that is pertinent to the system, and the final result delivered should reflect accurately the state of the system. e.g., in a network covering a chemical spill, the final information presented by the system should be a reflection of the characteristics of the spill, to include information such as the presence, extent and location of the spill. There are various approaches for in-network processing used in the areas of sensor networks, which include the Cougar approach [35], Directed diffusion [14], Tiny DB [18] and TAG [21].

The localized processing paradigms used in in-network computations reduce communication cost, making the network energy efficient. The network acts as a complex system, in which localized interactions between neighboring nodes helps to achieve a global objective through self-organization.

1.2 Segmentation In Sensor Networks

Segmentation can be defined as the identification of regions or spatially defined clusters which have similar properties. These similar properties could refer to similarity of attributes sensed, such as moisture, texture, composition, etc., or similarity in events occurring in these regions, e.g., regions with high/low traffic or regions with higher/lower event density. The goal of segmentation in a self-organizing network is to use the information obtained from local sampling to identify these regions, thereby inferring the spatial semantic structure the environment. Most environments monitored by sensor networks

have spatial structure (e.g., tracks or regions with different terrain), that decisively affects the observations made by the network. If the network can infer this spatial structure from its initial observations, it can predict the events occurring in the environment much more intelligently than a naive, uniform network. For example, network performance in tracking applications is much better if the network can infer the layout of tracks (if any), or distinguish between hazardous and navigable regions. Tracking is one application area in which segmentation is of use in sensor networks. Other applications would include contaminant detection, habitat monitoring, disaster response, etc. In each of these application areas, the goal is to infer the regions of interest by collectively analyzing the data acquired by the sensor nodes. Since every node surveys only a small part of the environment, such inference of the environment structure at a global level requires intelligent collaboration between the nodes.

1.2.1 Types of Segmentation

The different types of segmentation of interest in sensor networks are

1. **Phenomenological Segmentation:** Segmentation which is based on phenomena in the sensed environment. Each segment inferred is homogeneous in some phenomenon that it senses. i.e., the nodes identify themselves as one segment based on some phenomenological equivalence. This equivalence is defined in terms of the sensed attributes and is further divided into
 - *Static Equivalence:* Nodes group themselves in terms of a set of static attributes which can be either numerical, (i.e., nodes sense sufficiently similar attributes like temperature/pressure/moisture), or logical(i.e., temperature is less than

T). Statically equivalent segments can be extracted by comparing node readings on one or more attributes which is user specified, leading to a type of hierarchical segmentation scheme. For example, in the case of a habitat monitoring application, shady areas could be identified by comparing node readings from temperature or light sensors. These sensors would exhibit significantly lower readings of temperature or light intensities, than sensor nodes in bright, sunny regions. These regions could also be compared on further attributes, e.g., moisture content, to further subdivide shady areas into shady, but moist regions and shady, but dry regions, giving rise to a hierarchical segmentation scheme on shady areas.

- *Dynamic Equivalence:* Nodes segment the environment in terms of a set of time-varying attributes. The equivalence measured is not in terms of the attribute values directly measured by the sensors, but in the variation of these attributes. For example, an area that has varying light and shade may be regarded as a single region distinct from one that is always in light.
- *Hybrid Equivalence:* All points in the segment are equivalent in terms of a set of static and dynamic attributes such that both types are required to define the equivalence. Segments defined by hybrid equivalences are more likely to be found in nature, because dynamic phenomena strongly correlate with static attributes e.g., heavy vehicles drive only on dry areas. The attributes used for the segmentation process are predefined, but the segments themselves, or the equivalences that form the basis of the segments would be inferred through distributed computation. Thus sensors measure the relevant attributes, but the significant segments are determined within the sensor network through self-organization. As such, the segmentation process can be seen as the network

inferring the a priori unknown phenomenological structure of the monitored environment.

In all these cases, the goal is to infer the structure and regularities of the environment by mining the sensed data. Segmentation can also serve the purpose of determining what environmental states occur together, allowing for the determination of a reasonable set of states (discretization), that can be used instead of the full product space of all the variables.

2. Network Segmentation: This type of segmentation is done based on the local attributes which are pertinent to the network. All nodes within a segment are equivalent in terms of some network characteristic. Network segmentation can be further subdivided as

- *Structural Segmentation:* The equivalence of nodes is based on some structural attribute such as in-degree, out-degree, density of nodes in the neighborhood, e.t.c.
- *Functional Segmentation:* Nodes in a segment are equivalent on some functional characteristic such as available battery power, wait time, traffic pattern, e.t.c.

Network segmentation can be used to identify parts of the network with different resources or loads, information about which can be used to control network activity to achieve better performance and longer lifetime.

1.2.2 The Utility of Segmentation

Differentiating the environment into significant segments offers several advantages including:

1. *Ability to infer the phenomenological structure of the environment:* At its most basic level, phenomenological segmentation provides a meaningful map of the environment. By distinguishing distinct regions, segmentation reveals how the environment is organized at a scale that cannot be perceived by a single sensor. These regions represent phenomenologically meaningful areas that can be used to describe the environment at a level higher than raw sensor data. For example, after segmentation, an ecosystem can be described in terms of forested regions, grasslands, marshy patches e.t.c., by the network itself, instead of an array of light, temperature and light intensity values. These descriptions can be hierarchical, where segments which are at a small resolution are merged to form higher-level segments. Segmentation can also be multi-modal, where segments by single attributes are combined logically to obtain multi-attribute segments. Segment maps can also be used directly to determine and represent the connectivity structure of the environment.
2. *Identifying significant phenomena and features in unknown environments:* Sensor nodes are very often deployed in unknown territories, disaster regions, hostile territories and regions where things are virtually unknown. Segmentation would be extremely useful in identifying the salient features and discovering the regularities of the environment and hence understanding these regions, which would be useful in many application domains, especially exploratory ones.
3. *Identifying/predicting phenomenological patterns of behavior by agents in the environment:* One of the important applications of segmentation would be to correlate

the behavior of different agents in the system to the characteristics of the region. This is very commonly found in real systems with vehicles moving on smooth tracks, animals congregating in shady areas etc . By inferring the correct correlations between agent behavior and segments characteristics, the future behavior of agents can also be predicted. This can be done by using some method of associative learning or Bayesian inference.

4. *Providing a cognitive map to users:* The macro-scale map of the environment provided by segmentation can be used as a cognitive map by users of the system, or by the system itself. For example, it can be used to find regions with certain attributes, to find feasible or efficient paths between locations (e.g., through a minefield), or to plan activities (e.g., exploration in the most interesting areas, search for targets in the likeliest locations, traversals that meet certain conditions, etc.)
5. *Network Applications:* Another major class of applications for segmentation is in configuration and control of the network itself. Here, phenomenological and network segmentation can be used alone or in combination. In the most general case, segmentation can be used to tailor the dynamics of the network to the structure of the environment and the networks own resources. A specific application of this class is customizing the network configuration to the environment. The map of the environment produced via phenomenological segmentation can be used to “fit” the structure of the network to the environment. For example, local packet-forwarding or routing policies could be set such that nodes in phenomenologically active segments are used sparingly for communication while nodes in inactive segments take up the slack - essentially creating routing corridors in otherwise uninteresting regions. Similarly, the activity of sensor nodes in segments with different sensing

needs (e.g., different spatial resolution requirements), could be scheduled differently to ensure that a sufficient, but not excessive density of sensors is available everywhere. This could be supplemented with network segmentation, which would, for example, allow for different routing and activity policies in regions with sparse and dense node distribution. Such customization of the sensor network's structure and control can a) Enhance the sensing efficiency of the system and b) Maximize its useful lifetime by the intelligent use of resources.

1.3 Organization of the Thesis

This thesis is organized as follows: Chapter 2 provides an overview of segmentation and edge-detection, including approaches from image analysis, neural networks and sensor networks; Chapter 3 comprises a detailed description of the methods developed by us for segmentation in sensor networks; the simulation procedures and results are presented and discussed in Chapter 4; finally, Chapter 5 summarizes the research and identifies some future directions.

Chapter 2

Background and Related Work

The process of region detection in sensor networks is fundamentally similar to that of region detection in image processing, and image processing techniques for segmentation can be extended to the field of sensor networks. Both segmentation in sensor networks and in images aim to extract significant information from the massive amounts of spatially distributed data, to get much less, but more relevant information [29]. However, important differences between the two areas exist, making the direct application of these techniques to sensor networks difficult. A brief description of the methods used in the framework of image processing is described below, for a better understanding of the similarities and differences between both application areas, and the modifications required.

2.1 Segmentation in Image Processing

An image is made up of individual pixels, with varying intensities, which can be defined as the smallest, complete, elements of an image. Region detection or image segmentation is the process of dividing an image into a set of non-overlapping regions that completely cover the image [26]. Basic region detection schemes in image processing are based on

two principles.

1. Detection by defining the pixels that constitutes the region: *Region growing methods*.
2. Detection by defining the pixels that bound the region: *Edge detection methods*.

These two schemes are based on the fact that a region is characterized by pixels which are similar in intensity and differ from its neighboring regions. Using $H(R)$ to denote homogeneity for the region R in image P , the basic properties defining a region can be summed up as

$$H(R_i) = TRUE$$

i.e., every region is homogenous within itself

$$H(R_i \cup R_j) = FALSE, \forall i \neq j$$

Where R_i and R_j are adjacent regions

$$\cup R_i = P$$

i.e., the union of all the individual regions gives the complete image.

2.1.1 Edge Detection Methods

Edge detection methods are based on dissimilarity between adjacent pixels. Pixels with different attribute values belong to different regions and define a boundary of separation. The process of finding the boundary is called edge detection. An edge can be defined to be a significant, local change in image intensity [29]. Different types of edge detection operators are used for the purpose. These work by computing the gradients at different points in the image. Areas with significant changes in image intensity are characterized by large gradients, denoting an edge. Regions are then defined by tracking edges to identify closed boundaries. Edge detection operators are of two types

- First derivative intensity operators
- Second derivative intensity operators .

Using the first derivative intensity operators, the gradient of the image intensity is computed, by convolving the data with a mask. The two popularly used masks are the Prewitt Filter and the Sobel Filter. These use intensity values in the 3×3 region for each pixel to compute the gradients in the horizontal and vertical directions, G_x and G_y . From this the values of gradient, $G = (G_x^2 + G_y^2)^{(1/2)}$ are computed and orientation of the intensity found [29] [26]. While using the second derivative, the rate of change of intensity is calculated. Once the gradients are computed, a pre-specified threshold is applied and any value above the threshold is said to denote an edge.

2.1.2 Region Growing Methods

While edge detection methods rely on the differences between neighboring pixels, region growing methods are based on the spatial proximity of similar pixels. In this method, similar, adjacent pixels are merged together to form a region. Region growing algorithms are based on the assumption that every pixel belongs to a region and since regions are homogenous attribute-wise, similar neighboring pixels are most likely to belong to the same region. A pertinent issue to be addressed here is to decide what region properties should be used as the basis of comparison between pixels to form regions. This usually depends on the context in which region detection is done[29][26].

A significant amount of work has been done in the field of segmentation of images using variants of the above techniques. A review of these methods, which include thresholding, edge based operations and motion based operations is given by Kothari [17]. Image segmentation methods generally depend on a number of parameters. Koepfler, Lopez and

Morel [16] explain how the introduction of an energy function into the “region merging algorithm” helps to minimize the effect of these parameters, leading to a fast, multichannel algorithm.

Another set of methods used in image segmentation finds its basis in biological visual systems and uses information based on gestalt principles for segmentation. A simple neural oscillator method is proposed by Buhmann and von der Malsburg [4], which uses a neural oscillator network to segment images into figure and background. Connectivity between two neurons indicates that those oscillatory units are more likely to belong to the same segment. Global inhibition is used to prevent phase locking. The same principle is used in Pulse Coupled Neural Networks(PCNNs) [33] [32] [19] [28] and LEGION [20].

Sukourikhine, Prasad and Schlei [33] show how PCNN’s are used for segmentation in heterogenous materials and their use for edge extraction using autowaves is demonstrated in [32]. The basic principles behind PCNN’s are described in Kuntimad and Ranganath [19], and its use in an object detection system is presented [28]. LEGION has been used in the context of range image segmentation. A feature vector is associated with each oscillator which estimates the surface normal and curvature at its corresponding pixel location. Segmentation results as the emergent behavior of the network [20]. An extension to this framework, using a weight adaptation scheme to remove noise and preserve significant discontinuities in an image is presented by Chen, Wang and Liu [5]. A novel approach to image segmentation, treating it as a graph partitioning problem, is used by Shi and Malik [31]. This is extended into a dynamic image processing framework by defining motion profile vectors at each pixel [30]. An iterative method for motion segmentation using level sets is given by Mansouri and Konrad [22]. Another approach to motion-based segmentation is presented by Zhang and Minai [36], which uses block matching in conjunction with PCNN for the purpose.

2.2 Image Processing and Sensor Networks

The processes of segmentation in sensor networks and images are inherently similar, with a number of parallels between the two. Both regions in images and regions in environments with deployed sensor networks are characterized by similarities in attribute values. While in images, these attributes are usually limited to intensity values, in sensor networks the attribute values sensed by the nodes vary over a wide spectrum depending on its application. Most of the localized algorithms used in sensor networks for region detection are based on image processing techniques described above, with sensor nodes being considered as equivalent to pixels. However, several significant differences exist between the two.

1. Pixels are regularly spaced with uniform density, while sensor nodes are randomly deployed leading to irregular sampling of the environment.
2. The neighborhood of a pixel is deterministic, with its eight surrounding pixels forming its neighborhood. The notion of neighborhood in sensor networks is highly variable, with the neighbors determined typically by the transmission radius of the nodes.
3. The intensity of every pixel in an image is known, while a sensor field may have regions not covered by sensors. Intensity values in these locations must be inferred.

These differences make the application of image processing techniques to sensor networks with equivalent accuracy particularly challenging. Some of the work done so far in this context, and the modifications used to adapt these techniques to sensor networks are discussed below.

2.3 Information Processing in Sensor Networks

In the paper “Applications of Localized Image Processing Techniques in Wireless Sensor Networks” [8], the authors look into the application of classical edge-detection schemes as applied to sensor networks, focusing on the application of first order differentials. The Prewitt filter is a common approximation used, which is non-trivial to implement, given the irregular placement of pixels. Each node implements a modification of the Prewitt filter by itself, applying the value of the node to sectors of the filter within which other nodes do not fall. Every node then computes the gradient at its location. If this is greater than a threshold, t , it deems itself to be present on an edge. Every node is assumed to know its location, and that of its neighbors, which are the nodes which form within a radio range, R . On applying the Prewitt filter to compute the gradient at a particular node, random deployment may lead to nodes not having neighbors that fall into a particular sector of the filter, which is taken care of by assigning the value of the computing node to that particular sector. There may also be a case where multiple nodes may be present in a sector, which is taken into account by using the mean value of all nodes present in that sector. Edge detection using this scheme is found to perform well at sufficiently high densities. Chintalapudi and Govindan [6] look into three different schemes of edge/boundary detection using a statistical approach and a pattern recognition based approach is used in addition to an image processing scheme. In the statistical scheme, event predicate values are gathered from neighboring sensors and if these form a bimodal distribution, an edge is deemed to be present. Location information is not necessary. The image processing approach used again applies the Prewitt filter, but uses a weighted scheme to compensate for the irregularities caused by random node deployment. In the approach based on pattern recognition, the sensor tries to classify the data into two data

sets, using a linear classifier. The sensor tries to find an edge, $L(a, b, c) = ax + by + c = 0$, such that all the sensors within a probing neighborhood with similar values within a probe neighborhood lie on one side of the line. If the line passes within a distance r of the sensor, the sensor is determined to be an edge sensor. This scheme requires the knowledge of location. Of the three approaches, the linear classifier is found to perform the best. Nowak and Mitra [23] tackle the problem of boundary estimation in sensor networks, defining a boundary as being the delineation between two homogeneous regions. Boundary detection is done through a method of complexity regularization using a hierarchical, tree based estimation method while trying achieve an optimal energy-accuracy trade-off. Boundaries are described as Lipschitz functions, covering a broad range of parametric and non-parametric boundaries.

While boundary detection is of importance in the context of tracking the extent of phenomena, a method is needed to determine the semantically significant regions in the environment. This would not only involve extracting information from the locations at which sensors are present, but an estimation of the environment at locations not covered by sensors. Though some work has been done which deals with estimation of homogenous fields using interpolation methods [23], scenarios involving abrupt spatial changes are likelier to occur in a variety of applications and distributed, energy efficient methods are needed to address the problem of estimating these inhomogeneous fields. As segments are comprised of regions with similar attributes, there is a natural analogy between finding segments and finding clusters. We use a method of region growing proposed by Panaganadan and Sukhatme for region tracking [24]. The basic idea behind this algorithm, as used in image processing, is that if two adjacent pixels are similar in intensity, they are merged into a single region. While they use it purely in the context of tracking a specific region and in a single attribute case, we investigate the use of the algorithm in finding

all regions in environments of different complexities, with multi-attribute specifications. We examine the use of non-hierarchical, distributed methods to estimate inhomogeneous fields and use them in conjunction with a region growing algorithm to get a comprehensive global estimate of the segments. We also use two machine learning algorithms, namely the inverse distance weighted (IDW) algorithm and the K-nearest neighbors (KNN) algorithm for data imputation [13], making intelligent guesses about un-sensed regions. We evaluate the performance of these methods with different node densities.

Chapter 3

Environment Models and Algorithms

A comprehensive study of the phenomenon of segmentation requires the testing of various algorithms over a range of environment types. In this thesis, we generate a sufficiently wide variety of environments in order to characterize the performance of the segmentation algorithms as a function of environment complexity and different parameter values.

In this chapter, we look at the different environment models used, the model of the network, and the algorithms implemented. We also state the different assumptions made in the generation and use of these models and algorithms.

3.1 Environment Models

Each environment, E , is modelled as an $n \times n$ cellular region, with N_a observable attributes distributed over it. The distribution of attributes over the environment is highly inhomogeneous, with locally homogenous extended structures called segments corresponding to patches, regions, tracks etc. Regions or segments can be defined as connected subsets of the environment which have similar values for all or some attributes.

Every cell location (x, y) , is characterized by attributes, $a_i(x, y)$, where $i = 1, 2, \dots, N_a$.

There are $N_S(E)j$ segments in the environment, drawn from a fixed number of segment types. Attribute a_i is distributed over segment S_j , in a range $[a_{ij}^{min}, a_{ij}^{max}]$, with mean μ_{ij} and variance σ_{ij} . Each segment type has a characteristic mean and variance for each attribute, and the means for different segment types are well separated on at least one attribute. Each segment also has several global attributes such as area, perimeter, etc.

Specifically, we model three environment types:

- **Track Environment:** The 50×50 track environment, E_T , has two different piecewise linear tracks, S_{T1} and S_{T2} of different widths and is characterized by two attributes, a_i and a_j . The tracks are similar to each other on attribute a_i , but differ on both a_i and a_j from the background segment, S_B . Mean μ_{iB} is well separated from μ_{iT1} and μ_{iT2} , while μ_{jB} , μ_{jT1} and μ_{jT2} , are well separated from each other.

Figure 3.1: The Track Environment

- **Spiral Environments:** The spiral type 54×54 environment, E_S , consists of two segment types, S_{R1} and S_{R2} , spiraled together. The segments differ from each

other on both attributes, a_i and a_j . S_{R1} , and S_{R2} are modeled with well separated means, μ_{iR1} and μ_{iR2} for attribute a_i and μ_{jR1} and μ_{jR2} for attribute a_j . This is a very complex segmentation problem, and is used primarily as a benchmark because its complexity can be controlled systematically. It has been widely used in the classification literature [15]. The spiral width is varied systematically to change complexity. The spiral environments used in this thesis are shown in 3.2, 3.3, and 3.4

Figure 3.2: The Spiral Type One Environment

- **Patch Environments:** The 45×45 patch environments, E_P , have two different types of patch segments, S_{P1} and S_{P2} , distributed over the background, S_B , and are characterized by two attributes, a_i and a_j . The segments differ from each other on attribute a_i , and from the rest of the environment on both attributes, a_i and a_j . Mean μ_{iB} , is well separated from μ_{iP1} and μ_{iP2} , while μ_{jB} , μ_{jP1} and μ_{jP2} , are well separated from each other. 45×45 matrices are used to model the patch

Figure 3.3: The Spiral Type Two Environment

environments. The number and size of the patches is varied to control the complexity of this environment type. The Patch environments used are depicted in 3.5, 3.6, and 3.7

3.2 Sensor Model Used

A total of N_ν nodes, are distributed randomly over the environment. Each node, i , is equipped with some sensing, computation and communication capabilities. It also has some limited memory associated with it. Each node has a unique identifier, I_i , and is capable of sensing all attributes of interest. Every node is assumed to sense the attributes within the cell in which it is situated. Nodes also have a transmission radius, t_r , within which they can broadcast and receive messages. Nodes that fall within this

Figure 3.4: The Spiral Type Three environment

radius around a specific node are called its 1-hop neighbors, or just neighbors. Each node, i , senses the attributes within its cell, and stores the data in a measurement vector v_i . It also transmits to its neighbors a message packet with its identifier and measurement vector. Communication is assumed to be omni-directional and instantaneous over a 1-hop neighborhood. Localized communication is used to achieve the dual objectives of highly scalable algorithms and power conservation. Every node is also assumed to know its location, either through GPS or through localization algorithms.

3.3 Segmentation Process and Algorithms

3.3.1 Initial Segmentation

As mentioned earlier, segmentation refers to the identification of regions or clusters which have similar properties, based on the raw data acquired by the nodes from sensing their

Figure 3.5: The Patch Type One Environment

local environment. Every node is initially aware only of the attributes within its cell. Nodes communicate with each other and undergo a process of collaborative computation based on the data gathered by neighboring nodes to segment themselves into different regions. A segmentation scheme based on the detection of similarity in sensed attributes for neighboring nodes is used here. Using $H(S)$ to denote homogeneity for segment S , the basic property defining region S_r is

$$H(S_r) = TRUE \quad (3.1)$$

i.e. every region is homogenous within itself. Also

$$H(S_r \cup S_t) = FALSE, \forall r \neq t \quad (3.2)$$

Figure 3.6: The Patch Type Two Environment

where S_r and S_t are adjacent regions. Homogeneity can be defined in terms of all attributes or a subset of attributes. Nodes compare their readings on the specified subset of attributes and segment themselves based on these attributes.

Algorithm Description

Each node, i , senses the attributes within its cell, and stores the data in a node measurement vector v_i . It also transmits to its neighbors a message packet with its identifier, I_i , and node measurement vector, v_i . Every node checks the similarity of its cell measurement vector, with those received from each of its neighbors. A match is said to be found on one attribute, a_r , if two nodes i and j sense values which fall within a certain percentage of each other, on that attribute. If a match is found, the node marks this against a neighbor-match table, T_{ij} , which it maintains, and the process continues over

Figure 3.7: The Patch Type Three Environment

several sensing cycles. Initially the match for attribute r on node i with node j , T_{ij}^r is set to 0. Once the number of times the node finds a match with a neighbor (as a fraction of the number of trials), exceeds a pre-specified threshold, it deems its neighbor as similar to itself and hence lying in the same region. i.e, similarity between node i and its neighbor j , $\chi_{ij} = 1$. We implement the requirement of multiple matches even in static or slowly-varying environments to ensure robust segmentation in the presence of measurement noise, although we have not explicitly used a noise model. The node then sets its new node identifier or node label, τ_i , to the minimum identifier among all its similar neighbors that it knows of. Every node continues to compare itself to its neighboring nodes until no further similarities are found over a sufficiently long interval. At this point, all the nodes in a region with similar readings have converged to the same identifier value. The initial segmentation algorithm that the nodes run to identify which segment they belong to, is described below.

Pseudo-code for Initial Segmentation

```

For every node  $i$ ,      Transmit cell measurement vector  $v_i$  and node identifier,
 $I_i$ 

Repeat until no further change in new node identifier,  $\tau_i$ 

For every node  $i$ ,

    Compare attribute  $a_{ri}$  and  $a_{rj}$ 

    If  $| (a_{ri} - a_{rj}) \div | a_{ri} | \leq 0.05$ 

         $T_{ij}^r = T_{ij}^r + 1$ 

        If  $T_{ij}^r \geq \text{threshold}$ 

             $\chi_{ij}^r = 1$ .

    New node Identifier,  $\tau_i = \min(I_i, I_j, I_k \dots)$  of all  $k \in [\chi_{ij}, \chi_{ik}, \dots]$ .

```

After running the segmentation algorithm, the nodes have segmented themselves into different groups. The accuracy of segmentation, i.e, the number of correctly identified segments by the nodes as compared to the actual number and type of segments present in the environment, is dependent on various factors such as the density of deployment and distribution of the nodes, complexity of the environment, transmission radius etc.

3.3.2 Map Formation

Even at high densities, sensor nodes do not sample every part of the environment. Though a cellular model of the environment is used here, the random distribution of nodes together with the low density of distribution leaves many un-sensed cells. Complete segmentation requires assignment of segment identifiers to un-sensed cells through inference, such that every part of the environment is accounted for by the nodes through an intelligent estimate of their surroundings. To minimize the computational requirements of every individual

node, the nodes begin by forming a subjective estimate of their immediate environment, based on mutual collaboration between neighboring nodes. i.e, each node forms an estimated map of its surrounding region. This is done in such a way as to ensure that the entire environment is covered as far as is possible. The resulting map at each node is called a *node-map*.

Algorithm Description

Every node estimates the limits of its node-map, based on its location, and the location of its neighboring nodes. The limits refer to how far above, below, right or left of the node its node-map extends. The upward limit, y_{upper} is the distance above the node location to which the map extends, y_{lower} is the downward limit of the map. x_{right} refers to the limit to the right, and x_{left} is the limit to the left. Every node finds its limits of expansion as follows. A node i segregates its neighbors Λ_i into four types and stores this information in different sets, Λ_i^N , Λ_i^S , Λ_i^E and Λ_i^W . All neighbors of node i at locations with their y coordinates greater than the y coordinate of the node and their x coordinate displaced not more than one unit to right or left of the node are classified as Λ_i^N . All nodes which have their y coordinates lower than the y coordinate of the node and their x coordinate displaced not more than one cell above or below the node are classified as Λ_i^S . Nodes which have their x coordinates higher than the x coordinate of the node and their y coordinate displaced not more than one cell to right or left of the node are classified as Λ_i^E . Every node which has its x coordinate lower than the node coordinate and their y coordinate displaced not more than one cell to right or left of the node are classified as Λ_i^W . The y coordinate of the node corresponding to the minimum y coordinate of all nodes in the set Λ_i^N is the upper limit, y_i^{upper} , of the node-map, while the lower limit, y_i^{lower} , is taken as the maximum y coordinate of all y coordinates in Λ_i^S . The limits to the

left, x_i^{left} and right, x_i^{right} , correspond to the maximum of all x coordinates in Λ_i^W , and the minimum of all x coordinates in Λ_i^E , respectively. If the set of neighbors, $\Lambda_i = \Phi$, i.e., the node i has no neighbors, the nodes take the limits of expansion as the transmission radius t_r . If there are no neighbors in one direction, the limit of expansion in that direction is taken as the smaller of its distance to the environment boundary or the transmission radius. Every node forms its node-map within these limits of expansion.

Pseudo-code for Map Estimation

```

For every node  $i$ ,
Find  $\Lambda_i \leftarrow$  all neighbors of the node  $i$ 
If  $\Lambda_i \neq \Phi$ ,
     $\Lambda_i^N \leftarrow$  all  $j \in \Lambda_i$ , where  $y_j > y_i$ ,  $|x_j - x_i| \leq 1$ 
    If  $\Lambda_i^N \neq \emptyset$ 
        Upper-limit,  $y_i^{upper} = \min y_j$  of all  $y_j \in [\Lambda_i^N]$ 
    else
        Upper-limit,  $y_i^{upper} = \min[y_i + t_r, envsize - y_i]$ 
     $\Lambda_i^S \leftarrow$  all  $j \in \Lambda_i$ , where  $y_j < y_i$ ,  $|x_j - x_i| \leq 1$ 
    If  $\Lambda_i^S \neq \emptyset$ 
        Lower-limit,  $y_i^{lower} = \max y_j$  of all  $y_j \in [\Lambda_i^S]$ 
    else
        Lower-limit,  $y_i^{lower} = y_i - \min[y_i - t_r, 1]$ 
     $\Lambda_i^E \leftarrow$  all  $j \in \Lambda_i$ , where  $x_j > x_i$ ,  $|y_j - y_i| \leq 1$ 
    If  $\Lambda_i^E \neq \emptyset$ 
        Limit-right,  $x_i^{right} = \min x_j$  of all  $x_j \in x_E$ 
    else

```

```

    Limit-right,  $x_i^{right} = \min[x_i + t_r, envsize - x_i]$ 
 $\Lambda_i^W \leftarrow$  all  $j \in \Lambda_i$ , where  $x_j < x_i$ ,  $|y_j - y_i| \leq 1$ 
    If  $\Lambda_i^W \neq \emptyset$ 
        Limit-left,  $x_i^{left} = \max x_j$  of all  $x_j \in [\Lambda_i^W]$ 
    else
        Limit-left,  $x_i^{left} = \min[x_i - t_r, 1]$ 
    If  $\Lambda_i = \Phi$ ,
        Limits of expansion

```

$$y_i^{upper} = \min[y_i + t_r, envsize]$$

$$y_i^{lower} = y_i - \max[y_i - t_r, 0]$$

$$x_i^{right} = \min[x_i + t_r, envsize]$$

$$x_i^{left} = \max[x_i - t_r, 0]$$

At the end of running the map expansion algorithm, each node has a blank map of its immediate environment in its memory. Next, they try to fill in the maps by inferring the attributes of these cells through a process of intelligent collaboration with neighboring nodes.

3.3.3 Inference for Un-Sensed cells

Once the nodes have formed their individual node-maps, they try to form an estimate the values of the attributes of each un-sensed cell on their node-map, based on the values of the attributes of the nearest known nodes. Since regions with similar attributes have the same label, the nodes assign a label to each cell on its node-map, which corresponds to the inferred attributes and hence the segment to which it belongs. This is done by using two variants of a classification algorithm, the *K-Nearest Neighbors(KNN)* and *Inverse Distance Weighted (IDW)* algorithms. The working two algorithms are described below.

The KNN Algorithm

The KNN algorithm assigns a class label to a cell under consideration, by considering the class label of the k closest nodes for the cell and choosing the most commonly occurring label. The distance measure used here is the Euclidean distance between the neighboring nodes and the cell under consideration. In this case, every node i assigns to each unsensed (and therefore unlabeled) cell $p^i(x, y)$ on its node-map, a label, $CellLabel_p^i$, by finding Λ_i^p , the set of k nearest neighboring nodes to the cell, and assigning the plurality label to the cell.

Pseudo-code for the KNN Algorithm

For every node i ,

Find k nearest neighboring nodes, Λ_i^p , of $p^i(x, y)$

$CellLabel_p^i \leftarrow$ plurality label of k nearest neighbors of Λ_i^p

The IDW Algorithm

The Inverse Distance Weighted Algorithm is a variant of the *K-Nearest Neighbors* algorithm. As with the *KNN*, it finds the nearest k neighbors for each cell, $p^i(x, y)$, on its node map. It then assigns each of the k neighbors a weight proportional to the inverse of their distance from the cell. The weights for nodes with the same label are summed to give a weight for that label, and the cell is assigned the label with the largest total weight.

Pseudo-code for the IDW Algorithm

For every node i ,

- Find k nearest neighboring nodes, Λ_i^p , of $p^i(x, y)$ on $nodemap_i$
- Weight groups of nodes with similar labels inversely as their distance to the cell
- Sum the weights of each similarly labelled set of nodes.
- $CellLabel_p^i \leftarrow$ label of the group with the greatest weight

3.3.4 Conflict Resolution and Reassignment

Since spatially close nodes have overlapping node maps, some cells can be given conflicting labels by different nodes. This occurs when the k nearest neighbors for a cell as seen by two nodes are significantly different from each other. The cell for which conflict occurs is called a “point-ofconflict” for the two nodes. Conflict does not occur frequently and is more likely to happen in cases of low density of distribution or small transmission radius. But to ensure consistency in the labelling of cells in the case of such an occurrence, we implement a procedure for resolving such conflicts that might occur. The process of conflict resolution is called only once by each node. Conflicts are resolved by a process of localized, inter-node communication, with exchange of the labelled node-maps. Node i checks the map received from its neighbors, λ_i , for the existence of shared cells. i.e., cells on the node-map of i which also occur on the node-map of its neighboring node, j . The node then checks for conflict on these shared points. On detection of a conflict, it calls a function *resolve – conflict* to decide on a consistent label for the point of conflict. Function *resolve – conflict* takes as its input arguments, the node-maps of its surrounding neighbors. From these maps, the labels of the surrounding eight cells of

the point-of-conflict are found out and the label determined as the majority label among the eight surrounding cells. If the labels of any of the eight cells in consideration are in disagreement, the cell label given by the most number of neighboring nodes is taken as the label of the point. Every node undergoes this process of conflict resolution only once. An implicit assumption is made that a node that detects conflict first, informs its neighbors of its decision to resolve the conflict that occurs, so that two nodes do not try to resolve at the same time. It is deemed desirable that ultimately each nodes node-map includes cells labelled with its own segment identifier. This is achieved by reassignment of cells in each nodes node-map to neighboring nodes with appropriate labels. Each node compares the labels of each cell in its node-map with its own label. Any cell that is labelled differently from itself is deleted from its node-map after ensuring that it is included in the node-map of at least one neighboring node that is labelled similarly to that of the cell. A node first checks for the existence of that cell within the node-map of a neighboring node of the same label as that of the cell. If this is not found, it keeps the cell in its own node-map after relabelling the cell to its own label. This last step is a potentially important mechanism for removing noise in cell label estimates since it embodies a smoothness assumption for segments.

Pseudo-code for conflict resolution and reassignment of cells

```

For every node  $i$ ,
  For every cell  $p^i \in node - map_i$ 
    If  $CellLabel_p^i \neq CellLabel_p^j$  for any  $j \in \Lambda_i$ 
      Call function resolve-conflict
    If  $CellLabel_p^i \neq \tau_i$ .
      If  $\exists j \in \Lambda_i$  where  $\tau_j = CellLabel_p^i$  and  $p^i(x, y) \in nodemap_j$ 

```

```

    delete  $p^i(x, y)$  from node-map of  $i$ 
else set  $CellLabel_p^i = \tau_i$ 

```

Pseudo-code for the function resolve-conflict

```

For point-of-conflict,  $p^i(x, y)$ 
    From own map and maps of neighboring nodes,
    find CellLabels of  $p^j(x+1, y+1), p^j(x-1, y-1), p^j(x, y+1), p^j(x, y-1),$ 
         $p^j(x-1, y), p^j(x+1, y-1), p^j(x-1, y+1).$ 
     $CellLabel_p^i \leftarrow$  majority label of eight surrounding cells.
Return  $CellLabel_p^i$ .

```

Following the process of conflict resolution and reassignment, nodes form consistently labelled maps, overlapping with similarly labelled node-maps, but non-overlapping with dissimilarly labelled ones.

3.3.5 Segment Refinement

Segmentation becomes useful only when the segments formed are completely unique. i.e., two similar segments should identify themselves separately if they are separated by a dissimilar segment. With the segmentation algorithm described above, two nodes would identify themselves as part of the same segment if they lie within transmission radius. i.e., if $D(S_1, S_2) < t_r$, where S_1 and S_2 are similar segments. These nodes in two different segments, separated by a narrow width of a dissimilar segment, are very likely to communicate with one another and acquire the same label, thus wrongly assuming themselves to be parts of the same segment. To solve the problem, every node tries

to estimate its distance from its surrounding boundaries to undergo a process of re-segmentation within those boundaries, to form unique, enclosed segments. This is done as follows. Every node subdivides its neighbors into two sets, the set of *similar neighbors*, SN , which share the same label as itself, and the set of *dissimilar neighbors*, DN , which have labels dissimilar to its own (i.e., neighbors which presumably lie on the boundary of a nearby segment of different type). The node then goes through the node-maps of the nodes in set DN to find the closest dissimilar points, and hence distance to a boundary, if it exists, in the following eight subsets. These distances approximate the distance of a node to its surrounding boundaries. The eight boundary approximations used are

$$B_{ur} \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to node } i \text{ on node-map of } j \in [DN], \text{ where } y_j > y_i, x_j > x_i$$

$$B_r \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to node } i \text{ on node-map of } j \in [DN], \text{ where } y_j = y_i, x_j > x_i$$

$$B_{dr} \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j < y_i, x_j > x_i$$

$$B_d \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j < y_i, x_j = x_i$$

$$B_{dl} \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j < y_i, x_j < x_i$$

$$B_l \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j = y_i, x_j < x_i$$

$$B_{ul} \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j < y_i, x_j < x_i$$

$$B_u \leftarrow \text{Closest Cell}(x_j, y_j) \text{ to } i \text{ on node-map of } j \in [DN], \text{ where } y_j > y_i, x_j = x_i$$

Only those members of set SN_i which lie on or within these boundaries are included in the new list of similar neighbors, SN_{t_i} . The node then sets its newest node identifier, φ_i to the minimum identifier of all nodes in SN_{t_i} . The heuristic used here is that B_{ur} , B_r , B_{dr} etc., approximately identify the distance that a node lies from a boundary and

hence all the connected nodes in the same segment, within the identified boundary limits, converge to the same identifier value.

Pseudo-code for Segment Refinement

Initially, $\varphi_i = \tau_i$ For every node i ,

Transmit original node identifier, I_i , segment label, τ_i , and node-map to neighboring nodes, Λ_i

Repeat until no further change in newest label, φ_i

For every node i ,

$SN_i \leftarrow \forall j, k, \dots \in \Lambda_i, \text{ where } \tau_i = \tau_j.$

$DN_i \leftarrow \forall j, k, \dots \in \Lambda_i, \text{ where } \tau_i \neq \tau_j.$

$\forall DN_i$ traverse node maps to find

$B_{ur} \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j > y_i, x_j > x_i$

$B_r \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j = y_i, x_j > x_i$

$B_{dr} \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j < y_i, x_j > x_i$

$B_d \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j < y_i, x_j = x_i$

$B_{dl} \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j < y_i, x_j < x_i$

$B_{dl} \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j < y_i, x_j < x_i$

$B_{ul} \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j < y_i, x_j < x_i$

$B_u \leftarrow \text{Cell}(x, y)$ on node-map of $j \in [DS]$, where $y_j > y_i, x_j = x_i$

Find

$SN_{iupr} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_{ur}$

$SN_{ir} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_r$

$SN_{idr} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_{dr}$

$SN_{id} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_d$

$$SN_{ild} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_{dl}$$

$$SN_{il} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_l$$

$$SN_{iupl} \leftarrow \forall j \in SN_i, \text{ where } d(i, j) \leq B_{ul}$$

$$SN_{i1} \leftarrow \{SN_{iupr} \cup SN_{ir} \cup SN_{idr} \cup SN_{id} \cup SN_{ild} \cup SN_{il} \cup SN_{iupl}\}$$

$$\text{Newest node identifier of } \varphi_i \leftarrow \min \tau_j \forall j \in SN'_{i1}$$

After re-segmentation, the nodes reorganize themselves into smaller, distinctive segments. Nodes once more undergo the process of conflict resolution and reassignment to ensure map consistency and non-overlap with dissimilar segments.

3.3.6 Reorganizing the Node-Maps: Ensuring Load-Balancing or Load-Sharing between Nodes

Once nodes have consistent and homogenous maps, these maps are readjusted to balance the load for each node and to achieve sufficient redundancy in the overall map of the environment through overlaps between individual node maps. This makes the system robust against node failure. Every node tries to satisfy two constraints

- Reduce the size of its node-map (i.e., the area that it covers).
- Ensure that every point is shared by at least one other node.

Each node processes its node-map to ascertain which one of the following three criteria each cell on the node-map satisfies

- Not shared by any other node.
- Shared by one other similar neighbor.
- Shared by more than one similar neighbors.

Cells of each type are processed differently

- *Unshared cells*: For each unshared point, the node finds a similar neighbor with at least one point on its node-map lying adjacent to itself. It then sends a message to the node to add this point to its node-map. After a number of successive iterations, every cell is shared by at least one other node. If an adjacent neighboring node similar to itself is not found, load sharing does not take place.
- *Shared by one similar neighbor*: No further processing of the points is done by the node.
- *Shared by more than one similar neighbor*: If the point is shared by more than one neighbor, the node finds if the point lies closer to itself than to its other neighbors. If not, it deletes the point from its node-map.

At this point, the network has differentiated itself into distinctively labelled nodes, corresponding to the various segments in the environment. The accuracy of segmentation, as mentioned before, is dependent on various factors such as density of distribution, transmission radius, node-distribution, etc. A certain degree of robustness is introduced into the network by overlap between node-maps of similar, adjacent nodes. An overall map of the environment can be obtained by coalescing the node-maps of individual nodes in the network.

Chapter 4

Results

This chapter presents an analysis and discussion of the results obtained from implementing the algorithms presented in Chapter 3. We use the Track Environment to demonstrate the results of the algorithms discussed, while the spiral and patch environments are used to study the performance of the algorithms and its variation with different parameters and complexity.

4.1 Performance Measure

The performance of the segmentation algorithms implemented is validated by using a performance measure which attempts to incorporate two things.

1. The number of segments identified by the segmentation algorithm as opposed to the correct number of segments in the environment.
2. The accuracy of classification of cells in the correctly identified segments.

The accuracy of segmentation is then calculated by finding the total number of correctly classified cells in among all segments, $\sum S_i^{correct}$, where $S_i^{correct}$ is the number of correctly

classified cells in each segment S_i , in the environment as opposed to the total number of cells, N_E , in the environment.

As a result of segmentation, three cases may arise.

1. Every distinct Segment, S_i , in the environment correctly identifies itself. In this case,

$$S_i^{correct} = \text{Total number of correctly classified cells in segment } S_i$$

2. Each segment, S_i , is broken down into several segments, $S_{i1}, S_{i2}, \dots S_{in}$, e.t.c. Here, the largest segment amongst these segments, $S_{ilargest}$ is identified.

$$S_i^{correct} = \text{Total number of correctly classified cells in } S_{ilargest}.$$

3. Two or more Segments, S_i, S_j, S_n , are identified as a single segment, S_i . Here,

$$S_i^{correct} = \text{Total number of correctly classified cells in segment } S_i.$$

The performance measure, $P_{accurate}$, defined as a percentage of the correctly classified cells, is given as

$$P_{accurate} = \sum_i S_{icorrect} \setminus N_E$$

4.2 Result Analysis

4.2.1 Track Environment

As mentioned before, the track environment is used to demonstrate the performance of the algorithms, while the other two environment types are used for evaluating their accuracy under different node densities and parameter values. The track environment, E_T , shown in figure 4.1, has two different piecewise linear tracks, S_{T1} and S_{T2} of different widths and is characterized by two attributes, a_1 and a_2 . The tracks are similar to each other on

attribute a_1 , but differ on both a_1 and a_2 from the background segment, S_B . Mean $\mu_1 B$ is well separated from μ_{1T1} and μ_{1T2} , while μ_{2B} , μ_{2T1} and μ_{2T2} , are well separated from each other.

Figure 4.1: The Track Environment

Nodes that are randomly distributed over the environment, sample the attributes at their location, and segment themselves into various clusters based on this sampling. During deployment, no attempt is made to assign unique cells to nodes, so it is possible for two or more nodes to fall into the same cell. On running the initial segmentation algorithm as presented by A. Panagandan and G. Sukhatme [24], the nodes segment themselves into different groups based on their sensing as can be seen in figure 4.2.

A picture of the map of the environment obtained after the nodes initially segment themselves is shown in figure 4.3.

Two problems are seen to occur with the algorithm.

1. A significant area of the environment is left unsensed, resulting in segments with

Figure 4.2: Node classification after running initial segmentation algorithm

“holes”, i.e, cells about which no information is available. Analyzing the results obtained over 10 iteration runs, it is seen that at 0.6 node density, only an average of 44.3% of the environment is covered. This reduction in the percentage of the environment actually covered, as opposed to the percentage of node distribution over the environment is probably due to the fact that as distribution is random, and each node is assumed to sense the attributes of the cell within which it falls, a number of nodes might fall within the same cell in the environment, leading to a reduction in the total number of cells actually sensed.

2. Two separate track segments identify themselves as one single segment. This occurs due to the fact that the distance of separation between the two tracks is less than the transmission radius. This results in communication between two nodes lying in two separate, but similar segments, and results in their identifying themselves as one single segment. The other algorithms help ensure consistent and balanced

Figure 4.3: Map of environment obtained after running the initial segmentation algorithm

assignments.

To get around the problem of unsensed areas in the environment, we implement the two algorithms, the KNN and the IDW algorithm, while to get around the problem of incorrect segment identification, we propose and implement a heuristic for segment refinement.

4.2.2 Demonstration of accuracy of Segment Inference: The KNN and IDW Algorithms

Prior to running the inference algorithms, every node makes an estimate of the area around itself, called its *node – map*, for which it should infer the cell attributes. It runs the *mapexpansion* algorithm to create a map around itself. Depending on the node distribution, the nodes form maps of varying sizes. An example of the maps formed is shown in figure 4.4 and figure 4.5.

Figure 4.4: Map of an individual node

Using the *KNN* or *IDW* algorithm, every cell in the node-map of a node is assigned a label, based on the label of the nearest nodes to the cell. This helps cover the unsensed cells in the environment. A demonstration of the results obtained after running the *KNN* algorithm is shown in figure 4.6.

The *IDW* algorithm works similarly, with the label of the group of cells with the greatest weight chosen as the label of the cell, as seen in figure 4.7.

4.2.3 Segment Refinement

As seen above, the *IDW* and *KNN* algorithm when run as such, identify similar, separate segments which lie within transmission radius as one segment. This problem is overcome by running the segment refinement algorithm. The segment refinement algorithm finds the approximate distance of a node from its boundaries and reruns the segmentation algorithm within these boundaries to form unique, enclosed segments. The results obtained after

Figure 4.5: Map of multiple nodes

running the segment refinement algorithm on the track environment are shown figure 4.8.

In the next section, we take a look at the performance of the algorithms in different environments, with varying parameter values.

4.3 Effect of Environment Complexity and System Parameters

The quality of segmentation results obtained over environments of different complexities is studied by running the segmentation algorithms at different node densities for different values of k . The quality of segmentation obtained is evaluated using the performance metric described above.

Figure 4.6: Environment map obtained after running the KNN algorithm

4.3.1 The Spiral Environment

As described in Chapter 3, the spiral environment, E_S , consists of two segment types, S_{R1} and S_{R2} , spiraled together. The segments differ from each other on both attributes, a_1 and a_2 . S_{R1} , and S_{R2} are modeled with well separated means, μ_{1R1} and μ_{1R2} for attribute a_1 and μ_{2R1} and μ_{2R2} for attribute a_2 . Here we use spirals which are 9 cells wide, 6 cells wide and 3 cells wide to vary the complexity of the environment.

The quality of segmentation obtained at node densities 0.2, 0.4, 0.6 and 0.8 is plotted for varying values of k used in the segment inference algorithms. Here k is set to 3, 5 or 7.

Spiral Type One

The Spiral Type One Environment used is of the least complexity, with cells approximately 9 cells wide spiralled together. A picture of this environment is seen in figure 4.9.

Figure 4.7: Environment map obtained after running the IDW algorithm

The performance of the segmentation algorithms used are studied, under varying values of node density and using different values of k in the KNN and IDW algorithms. The accuracy plots obtained at node densities 0.2, 0.4, 0.6 and 0.8 for $k = 3$, $k = 5$, and $k = 7$.

As is seen in figures 4.10, 4.11 and 4.12, the IDW and KNN algorithms show comparable results, with the IDW algorithm showing slightly better results than the KNN algorithm. This is as expected, as nodes that lie closer to a cell are more likely to be part of the same segment that the cell belongs to. The accuracy of segmentation is seen to be quite high even in the case of nodes at 0.2 node density. This is probably due to the spiral being of sufficient width(hence less complex) in this case. Some examples of the resulting segmentation obtained at different node densities is depicted in the following figures 4.13, 4.14, 4.15, 4.16.

As seen in figures 4.13, 4.14, 4.15, and 4.16, the segmentation results obtained

Figure 4.8: Environment map obtained after running the Segment Refinement algorithm

for the Spiral Type One environment shows a high degree of comparability to the actual environment, The segmentation results show very high accuracy at node densities 0.8 and 0.6, with accuracy falling at lower node densities. The segmentation results are seen to be less accurate at the edges of the spiral, while the parts of the segment within the spiral are accurately reproduced.

Spiral Type Two

The Spiral Type Two Environment used is of the slightly more complex than the Spiral Type One Environment, with cells approximately 6 cells wide spiralled together as in figure 4.17.

The accuracy plots depicting the performance of the inference and segmentation algorithms over this environment type at $k = 3$, $k = 5$ and $k = 7$ is shown in the figures 4.18 and is shown in figures 4.19 and 4.20 respectively.

Figure 4.9: The Spiral Type One Environment

Here too, the performance in the *IDW* and *KNN* algorithms in segment inference for subsequent segment refinement, shows very minor variations, with performance increasing with increasing node density, as expected. Nodes show correct inference of their segments in majority of the cases, with incorrect inferences at lower node densities 4.25. Some of the resulting segments obtained are shown in figures 4.21, 4.22, 4.23 and 4.24.

As is seen in figures 4.21, 4.22, 4.23 and 4.24, the segmentation results are highly accurate at higher node densities, comparable to the results of the Spiral Type One environment, with incorrect segmentation results produced at lower densities. An example of a bad segmentation result produced at lower densities is shown in 4.25. Again, the incorrect segmentation is produced toward the edges of the spiral.

Figure 4.10: Accuracy plot obtained for Spiral Type One , with $k=3$

Spiral Type Three

The Spiral Type Three Environment used is of the the most complex of the spiral environments used, with spirals of approximately 3 cells in width in a spiral. This can be seen in figure 4.26.

Once more, the performance of the algorithms are studied with different parameter values. The results obtained are shown in the plots 4.27, 4.28 and 4.29.

Figure 4.11: Accuracy plot obtained for Spiral Type One, with $k=5$

Figure 4.12: Accuracy plot obtained for Spiral Type One , with $k=7$

Figure 4.13: Result of Segmentation for Spiral Type One at node density 0.8, $k=3$

Figure 4.14: Result of segmentation for Spiral Type One at node density 0.6, $k=3$

Figure 4.15: Result of segmentation for Spiral Type One at node density 0.4, $k=3$

Figure 4.16: Result of segmentation for Spiral Type One at node density 0.2, $k=3$

Figure 4.17: The Spiral Type Two Environment

Figure 4.18: Accuracy plot obtained for Spiral Type Two, with k=3

Figure 4.19: Accuracy plot obtained for Spiral Type Two , with k=5

Figure 4.20: Accuracy plot obtained for Spiral Type Two, with k=7

Figure 4.21: Result of segmentation for Spiral Type Two, at node density=0.8, $k=3$

Figure 4.22: Result of segmentation for Spiral Type Two, at node density=0.6, $k=3$

Figure 4.23: Result of segmentation for Spiral Type Two, at node density=0.4, $k=3$

Figure 4.24: Result of segmentation for Spiral Type Two, node density=0.2, $k=3$. An example of good segmentation at low densities.

Figure 4.25: Result of segmentation for Spiral Type Two, at node density=0.2, $k=3$. An example of bad segmentation at low densities.

Figure 4.26: The Spiral Type Three environment

Figure 4.27: Accuracy plot obtained for Spiral Type Three Environment, $k=3$

Figure 4.28: Accuracy plot obtained for Spiral Type Three Environment, k=5

Figure 4.29: Accuracy plot obtained for Spiral Type Three Environment, $k=7$

A few results obtained from the Spiral Type Three environment are depicted below in figure 4.30 and 4.31.

The Spiral Type Three Environment shows the least accurate results among all the segment types, with incorrect inference of segments being made even at very high densities. This happens as the segments are of very narrow widths and so inaccurate classification of cells near the edges, leads to nodes wrongly detecting boundaries during the segment refinement process, and reclassify themselves as belonging to smaller, enclosed segments. As is seen in the results obtained, the algorithms perform best under high density of node distribution, in all the three environment types. It is also seen that the performance of the algorithms decreases with increase in environment complexity for each value of density of distribution used. The algorithms do not perform well even under higher densities of node distribution in the Spiral Type Three Environment, examples of which are shown in figure 4.30 and figure 4.31. The decrease of performance with increase in environment

Figure 4.30: Result of segmentation for Spiral Type Three, node density=0.8, $k=3$

complexity is shown in the graphs in figures 4.32 and 4.33. The value of k does not seem to have any significant effect on the segmentation results as in figure 4.34.

Figure 4.31: Result of segmentation for Spiral Type Three, node density=0.6, $k=3$

Figure 4.32: Decrease of performance with increase in environment complexity, for each value of node density, using the KNN algorithm for cell inference, at $k=3$

4.3.2 The Patch Environment Type

The patch environments, E_P , have two different types of patch segments, S_{P1} and S_{P2} , distributed over the background, S_B , and is characterized by two attributes, a_1 and a_2 . The segments differ from each other on attribute a_1 , and from the rest of the environment on both attributes, a_1 and a_2 . Mean μ_{1B} , is well separated from μ_{1P1} and μ_{1P2} , while μ_{2B} , μ_{2P1} and μ_{2P2} , are well separated from each other.

Here, we use three different types of patch environments, and vary the complexity by varying the size of the patches. The distance between boundaries, i.e., between the patch and the environment boundary is kept the same in all cases.

Figure 4.33: Decrease of performance with increase in environment complexity, for each value of node density, using the IDW algorithm for cell inference, at $k=3$

Patch Type One

The Patch Type One Environment, consists of two rectangular patches separated from each other by 15 units as shown in figure 4.35.

The quality of segmentation obtained from running the segmentation algorithms on this environment under different node densities, with varying k is depicted in plots below shown in figure 4.36 and figure 4.37.

Here, the results obtained using the *KNN* and *IDW* algorithm follow each other very closely, without significant difference in performance. Some of the resulting segments obtained after segmentation, are shown in figures 4.39, 4.40, 4.41, and 4.42.

As seen in the figures, very good segmentation results are obtained at all values of node densities. The accurate results at even at low node densities is due to the low complexity of the environment type.

Figure 4.34: Effect of k on segmentation accuracy

Patch Type Two

The Patch Type Two Environment, consists of four rectangular patches separated from each other by 4 units, and 5 units from the boundary. The segmentation results obtained for varying values of k and node densities are shown in the figures variation in performance under for different node densities, for varying k is shown in the plots seen in figures 4.47, 4.45, and 4.46.

The *KNN* and *IDW* algorithms closely follow each other in this results in this environment type too, with performance increasing with increasing node density. A few of the Segmentation results obtained at the different values of node density are shown in the figures 4.47, 4.48, 4.49, 4.50, and 4.51.

Figure 4.35: The Patch Type One Environment

Figure 4.36: Accuracy plot obtained for Patch Type One, with k= 3

Figure 4.37: Accuracy plot for Patch Type One, with k= 5

Figure 4.38: Accuracy plot obtained for Patch Type One, with k= 7

Figure 4.39: Result of segmentation for Patch Type One, at node density=0.8, $k=3$

Figure 4.40: Result of segmentation for Patch Type One, at node density=0.6, $k=3$

Figure 4.41: Result of segmentation for Patch Type One, at node density=0.4, $k=3$

Figure 4.42: Result of segmentation for Patch Type One, at node density=0.2, $k=3$

Figure 4.43: Patch Type Two

Figure 4.44: Accuracy plot obtained for Patch Type Two, with k= 3

Figure 4.45: Accuracy plot obtained for Patch Type Two, with k= 5

Figure 4.46: Accuracy plot obtained for Patch Type Two, with k= 7

Figure 4.47: Result of segmentation, Patch Type Two, at node density 0.8, $k=5$

As seen in the figures of the results of segmentation, the Patch Type Two Environment shows substantially accurate results, with incorrect inferences made at lower densities. Wrong inferences are made around the edges of the segments, even leading to two segments merging with one another at lower densities due to wrong assignment of cell labels in the cells between the two segments.

Patch Type Three

The Patch Type Three environment consists of four rectangular patches separated from each other by 4 units, and 5 units from the boundary.

Figure 4.48: Result of segmentation for Patch Type Two, at node density 0.6, $k=5$

Figure 4.49: Result of segmentation for Patch Type Two, at node density 0.4, $k=5$

Figure 4.50: Result of segmentation for Patch Type Two, at node density 0.2, $k=5$

Figure 4.51: Result of segmentation for Patch Type Two, at node density 0.2, $k=5$

Figure 4.52: Patch Type Three

Figure 4.53: Accuracy plot obtained for Patch Type Three with $k=3$

Observing the graphical plots obtained from the analysis of the Patch Type Three Environment, it is seen that the *IDW* shows a larger variation in performance than the *KNN* algorithm under lower densities, without significant difference in the accuracy of inference at higher node densities.

A few of the segmentation results obtained from the algorithms are shown in the figures 4.56, 4.57, 4.58, and 4.59.

The Patch Type Three Environment shows resulting segmentation results which reflect the higher accuracy of segmentation at higher node densities, and decreasing performance of the algorithm at lower node densities as seen in the other environment types. The variation of segmentation accuracy with increase in environment complexity is shown in the figures 4.60 and 4.61.

As before, no significant effect is seen in the variation of k , on accuracy of segmentation performance.

Figure 4.54: Accuracy plot obtained for Patch Type Three with $k=5$

4.4 Discussion

An analysis of the performance of the algorithms over environments of different complexities, shows that the *IDW* and *KNN* algorithms show similar performances in the resulting segmentation, with the *IDW* algorithm performing slightly better in most cases. In a few cases, the *IDW* algorithm is found to perform much better than the *KNN* algorithm in cell label inference at lower density, in the more complex environments. Performance is seen to decline with increase in environment complexity. This variation is very evident in the spiral environments, while it is not sufficiently clear in the patch environment. This might be because the variation in complexity in the patch environments is not sufficiently large to demonstrate this effect. Variation of k does not seem to have a significant effect on the accuracy of segmentation obtained, with no specific pattern seen in the accuracy obtained at different k .

Figure 4.55: Accuracy plot obtained for Patch Type Three with k=7

Figure 4.56: Result of segmentation for Patch Type Three at node density 0.8, k=3

Figure 4.57: Result of segmentation for Patch Type Three at node density 0.6, $k=3$

Figure 4.58: Result of segmentation for Patch Type Three at node density 0.4, $k=3$

Figure 4.59: Result of segmentation for Patch Type Three at node density 0.2, k=3

Figure 4.60: Variation of segmentation accuracy with complexity at k=3 for the patch environments

Figure 4.61: Variation of segmentation accuracy with complexity at $k=5$ for the patch environments

Chapter 5

Conclusions and Future Work

In this thesis, we have presented a comprehensive study of segmentation using sensor networks with various node densities and in environments of varying complexity. The distributed segmentation algorithms and heuristics presented here are seen to have reasonably good accuracy, with better performance at high densities. However, segmentation does break down when environments reach a certain level of complexity.

We also present a heuristic for segment refinement, and indicate the necessity for such a refinement to take place. In an actual scenario, refinement is a necessary step to identify the distinct regions or areas in the environment, to completely utilize the network's segmentation capability.

Though this study indicates good performance in the most general case, further refinements are necessary for more accurate results. In some occurrences, under lower densities, some areas are seen to be left unsensed. Also as seen in the segmentation results, segments can be broken into smaller fragments. Merging these segments into one under a set of given conditions, using multi-hop communication between nodes to identify and provide information to segments about the similar segments, using a hierarchial segmentation

scheme to enable answering global queries on demand, etc. are areas of potential further study. Another area of future work is in using the heuristics presented in this work in a the framework of a continuous environment, using schemes like Voronoi tessellations or convex hull algorithms.

The comprehensive segmentation algorithm presented in this thesis can be used as the basis of intelligent sensor networks that use the inferred spatial structure of their environment to optimize scheduling, sensing, resource usage, configuration, routing and query response. Thus, the main contribution of this work is to provide an important *enabling technology* for better sensor network systems.

References

- [1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. In *IEEE Communications Magazine*, volume 40, pages 102–114, August 2002.
- [2] A. Bharathidasan and V.A.S. Ponduru. Sensor networks: An overview, available online: <http://wwwcsif.cs.ucdavis.edu/bharathi/sensor/survey.pdf>.
- [3] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world, <http://citeseer.ist.psu.edu/bonnet00querying.html>. *IEEE Pers. Commun.*, vol. 7, pp. 10–15., 2000.
- [4] J. Buhmann and C.V.D. Malsburg. Sensory segmentation by neural oscillators. *Biological Cybernetics*, vol. 67, pp. 233–246, 1992.
- [5] K. Chen, D. Wang, and X. Liu. Weight adaptation and oscillatory correlation for image segmentation. *IEEE Transactions on Neural Networks*, 11(5):1106–1023, 2000.
- [6] K.K. Chintalapudi. Localized edge detection in sensor fields.<http://citeseer.ist.psu.edu/744233.html>. 2003.
- [7] D. Culler and D. Estrin. Overview of sensor networks. *EUROMICRO Symposium on Digital System Design (dsd)*, 0, 2004.

- [8] D. Devaguptapu and B. Krishnakumar. Applications of localized image processing techniques in wireless sensor networks. In *SPIE's 17th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, volume 38, pages 187–191, 2003.
- [9] J. Elson and D. Estrin. Sensor networks: A bridge to the physical world. In *Wireless Sensor Networks*, Kluwer, 2004.
- [10] Wikipedia Entry. Information processing.
- [11] D. Estrin, D. Culler, and K. Pister. Connecting the physical world with pervasive networks. In *IEEE Pervasive Computing*, 2002.
- [12] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Fifth Annual International Conference on Mobile Computing and Networks*, 1999.
- [13] N. George, A. Minai, and S. Doboli. Self organized inference of spatial structure in randomly deployed sensor networks. In *International Conference on Complex Systems*, 2006.
- [14] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.
- [15] N. Karunanithi, R. Das, and D. Whitley. Genetic cascade learning for neural networks. In *Combinations of Genetic Algorithms and Neural Networks, 1992. COGANN-92. International Workshop*, 1992.

- [16] G. Koepfler, C. Lopez, and J. M. Morel. A multiscale algorithm for image segmentation by variational method. *SIAM Journal on Numerical Analysis*, 31(1):282–299, 1994.
- [17] R. Kothari. Image segmentation. In *Artificial Neural Systems Laboratory, University of Cincinnati, Cincinnati, OH, Preprint*.
- [18] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data-centric routing in wireless sensor networks. In *IEEE INFOCOM 2002*, 2002.
- [19] G. Kuntimad and H.S. Ranganath. Perfect image segmentation using pulse coupled neural networks. *IEEE Transactions on Neural Networks*, 10(3):591–598, 1999.
- [20] X. Liu and D. L. Wang. Range image segmentation using a relaxation oscillator network. *IEEE-Neural Networks*, 10(3):564, May 1999.
- [21] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.
- [22] A. Mansouri and J. Konrad. Motion segmentation with level sets. citeseer.ist.psu.edu/mansouri99motion.html. pages 126–130.
- [23] R. Nowak and U. Mitra. Boundary estimation in sensor networks: Theory and methods. In *2nd International Workshop on Information Processing in Sensor Networks*, 2003.
- [24] A. Panangadan and G.S. Sukhatme. Data segmentation for region detection in a sensor network.

- [25] E.M. Petriu, N.D. Georganas, D. Makrakis, and V.Z. Groza. Wireless intelligent sensor network environment monitoring, 2002.
- [26] M. Petrou and P. Bosdogianni. *Image Processing. The Fundamentals*. John Wiley And Sons, Ltd., 1999.
- [27] G.J. Pottie and W.J. Kaiser. Wireless integrated sensor networks. In *Communication of the ACM*, May 2000.
- [28] H.S. Ranganath and G. Kuntimad. Object detection using pulse coupled neural networks. *IEEE Transactions on Neural Networks*, 10(3):615–620, 1999.
- [29] J. Russ. *The Image Processing Handbook*. Ron Powers, 1999.
- [30] J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *International Conference on Computer Vision*, pages 1154–1160, 1998.
- [31] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
- [32] A.N. Skourikhine. Parallel image processing with autowaves: Segmentation and edge extraction. In *4th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida*, 2000.
- [33] A.N. Skourikhine, L. Prasad, and B.R. Schlei. Neural network for image segmentation. In *Conference Applications and Science of Neural Networks, Fuzzy Systems and Evolutionary Computation, part of 45th SPIEs International Symposium on Optical Science and Technology, San Diego, California*, 2000.
- [34] D. Timmermann. Wireless sensor systems - constraints and opportunities. *EUROMI-CRO Symposium on Digital System Design (dsd)*, 0:8, 2005.

-
- [35] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. 2002.
- [36] X. Zhang and A.A. Minai. Temporally sequenced intelligent block matching and motion segmentation using locally coupled oscillators. *IEEE Transactions on Neural Networks*, 15(5), 2004.