

Spring 2013

Unmanned Ground Vehicle navigation and coverage hole patching in Wireless Sensor Networks

Guyu Zhang
Louisiana Tech University

Follow this and additional works at: <https://digitalcommons.latech.edu/dissertations>



Part of the [Automotive Engineering Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Zhang, Guyu, "" (2013). *Dissertation*. 314.
<https://digitalcommons.latech.edu/dissertations/314>

This Dissertation is brought to you for free and open access by the Graduate School at Louisiana Tech Digital Commons. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of Louisiana Tech Digital Commons. For more information, please contact digitalcommons@latech.edu.

**UNMANNED GROUND VEHICLE NAVIGATION
AND COVERAGE HOLE PATCHING IN
WIRELESS SENSOR NETWORKS**

by

Guyu Zhang, B.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

**COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY**

January 2013

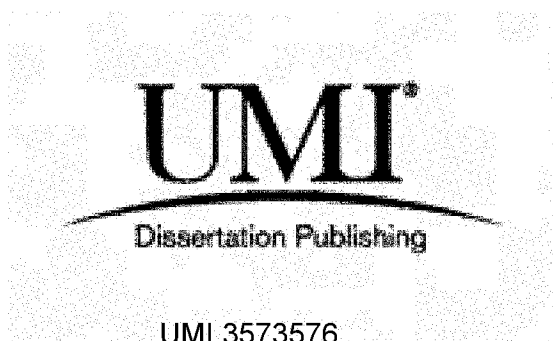
UMI Number: 3573576

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.

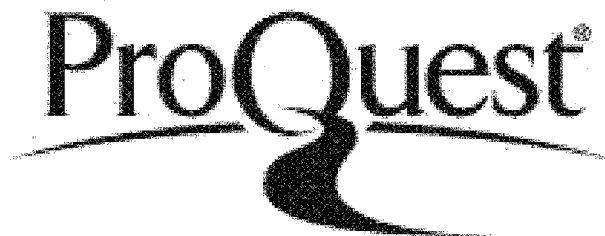


UMI 3573576

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

January 14, 2013

Date

We hereby recommend that the dissertation prepared under our supervision
by Guyu Zhang

entitled Unmanned Ground Vehicle Navigation and Coverage Hole Patching in
Wireless Sensor Networks

be accepted in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy



Supervisor of Dissertation Research



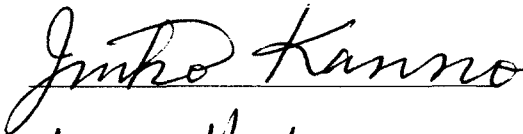
Head of Department

Department

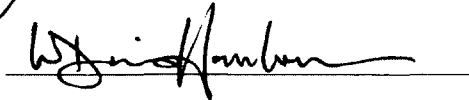
Recommendation concurred in:

 (Duncan)

Vic Vrande Elshar



Advisory Committee

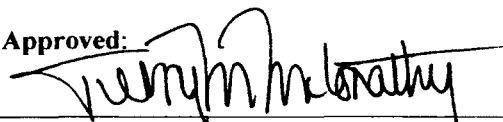


Approved:



Director of Graduate Studies

Approved:



Dean of the Graduate School



Dean of the College

ABSTRACT

This dissertation presents a study of an Unmanned Ground Vehicle (UGV) navigation and coverage hole patching in coordinate-free and localization-free Wireless Sensor Networks (WSNs). Navigation and coverage maintenance are related problems since coverage hole patching requires effective navigation in the sensor network environment. A coordinate-free and localization-free WSN that is deployed in an ad-hoc fashion and does not assume the availability of GPS information is considered. The system considered is decentralized and can be self-organized in an event-driven manner where no central controller or global map is required.

A single-UGV, single-destination navigation problem is addressed first. The UGV is equipped with a set of wireless listeners that determine the slope of a navigation potential field generated by the wireless sensor and actuator network. The navigation algorithm consists of sensor node level-number assignment that is determined based on a hop-distance from the network destination node and UGV navigation through the potential field created by triplets of actuators in the network. A multi-UGV, multi-destination navigation problem requires a path-planning and task allocation process. UGVs inform the network about their proposed destinations, and the network provides feedback if conflicts are found. Sensor nodes store, share, and communicate to UGVs in order to allocate the navigation tasks. A special case of a single-UGV,

multi-destination navigation problem that is equivalent to the well-known Traveling Salesman Problem is discussed.

The coverage hole patching process starts after a UGV reaches the hole boundary. For each hole boundary edge, a new node is added along its perpendicular bisector, and the entire hole is patched by adding nodes around the hole boundary edges.

The communication complexity and present simulation examples and experimental results are analyzed. Then, a Java-based simulation testbed that is capable of simulating both the centralized and distributed sensor and actuator network algorithms is developed. The laboratory experiment demonstrates the navigation algorithm (single-UGV, single-destination) using Cricket wireless sensors and an actuator network and Pioneer 3-DX robot.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.

Author Guyda Zhang
Date 01/21/2013

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
ACKNOWLEDGMENTS	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Motivations	2
1.2 Scope of Work	3
CHAPTER 2 RELATED WORK.....	9
2.1 Coverage Problem and Coverage Hole Patching	9
2.2 Automatic Unmanned Ground Vehicle Navigation	10
2.3 Task Allocation	13
2.4 Leader Election	14
2.5 Obstacle Avoidance.....	15
2.6 Hardware Experiment	15
CHAPTER 3 PROBLEM FORMULATION	17
CHAPTER 4 ALGORITHMS	24
4.1 Single-UGV, Single-Destination Case	26
4.1.1 Level Assignment Algorithm	26
4.1.1.1 Centralized Algorithm	27

4.1.1.2	Distributed Algorithm	28
4.1.2	UGV Control Algorithm.....	32
4.1.2.1	Centralized UGV Control Algorithm.....	39
4.1.2.2	Distributed UGV Control Algorithm.....	41
4.1.2.3	Comparisons with Other Algorithms	45
4.2	Multi-UGV, Multi-Destination Case.....	46
4.2.1	Leader Election Algorithm	46
4.2.2	Level Number Assignment Algorithm	52
4.2.3	Task Allocation Algorithm	53
4.2.4	Special Case: Traveling Salesman Problem	55
4.3	Hole Patching Algorithm	58
CHAPTER 5	WIRELESS SENSOR NETWORK SIMULATOR.....	62
5.1	Architecture Design	62
5.2	Function Development	65
5.2.1	Node Layer	65
5.2.2	Interface Layer.....	68
5.2.3	Network Layer	70
5.3	Applications	71
CHAPTER 6	HARDWARE TESTBED.....	77
6.1	Equipment Design.....	77
6.2	Experiment Setup	80
6.3	Experiment Results.....	81
CHAPTER 7	RESULTS AND DISCUSSION	83

CHAPTER 8 CONCLUSIONS AND FUTURE WORK.....	97
BIBLIOGRAPHY.....	101

LIST OF TABLES

Table 5.1:	Selected functions in the <i>Node</i> class	66
Table 5.2:	Selected functions in the <i>UGV</i> class.....	68
Table 5.3:	Selected functions in the <i>Netcard</i> class.....	69
Table 5.4:	Selected functions in the <i>SigDetector</i> class.....	70
Table 5.5:	Selected functions in the <i>Network</i> class.....	70

LIST OF FIGURES

Figure 1.1: Hole patching in a WSAN.	4
Figure 1.2: Hop distances between nodes: node u and v are with hop distance 1 in the left figure while node u and v are with hop distance 2 in the right.	6
Figure 3.1: Arrays of listeners on the UGV.	20
Figure 4.1: Logic flow of the proposed algorithms.	24
Figure 4.2: Successive processes of Algorithm 2.....	30
Figure 4.3: UGV navigation in a WSAN, where each node's ID, level number and connections are shown. At this step in the example, we have the potential field generated by the triplet of actuator nodes A , B , and C , shown as circles, which are assigned to nodes 1, 2, and 3, respectively.....	34
Figure 4.4: Illustrated cases in the proof of Lemma 4.1.....	36
Figure 4.5: An example of Equation 3.3, where nodes A , B and C are arbitrarily placed.....	38
Figure 4.6: General view of navigation-assisted nodes in the centralized control algorithm.....	39

Figure 4.7:	Successive iterations of the Level Assignment Algorithm. Active nodes, shown as squares marked with their ID, broadcast their ids for maximum distances of (a) one (b) two (c) four and (d) eight hops. Relay nodes, shown as circles marked with the maximum ID received prior to the iteration, only retransmit received messages with larger IDs. The shaded nodes represent those nodes covered by the eventual leader for that iteration. Solid arcs represent the communication graph. A dashed arc from node a to node b indicates that node a was able to transmit its id to the given node within the proper hop distance. Observe that during the final iteration of this example only two nodes remain active and the leader gets selected after the iteration completes.....	51
Figure 4.8:	A critical case for the nearest neighbor algorithm in MTSU. The distances between nodes are labeled underneath, where δ is a very small positive real value.	56
Figure 4.9:	Area coverage where nodes are connected at the length of r_c . Left: no coverage hole when $r_s = r_c/\sqrt{3}$; middle: a coverage hole exists in the middle when $r_s < r_c/\sqrt{3}$; right: no coverage hole when $r_s > r_c/\sqrt{3}$	59
Figure 4.10:	Hole patching process. Upper: an identified coverage hole; middle: new nodes are added along the edges of former coverage hole; lower: the hole is fully patched.	61
Figure 5.1:	Architecture of the simulation testbed.....	63
Figure 5.2:	Structure of the <i>run()</i> function.....	67
Figure 5.3:	Components in NodeStore.	71
Figure 5.4:	GUI Application Interface.	73
Figure 5.5:	GUI with a network deployed.	74
Figure 5.6:	GUI shows a network with multiple levels assigned.....	75
Figure 5.7:	GUI Application with results displayed.	76
Figure 6.1:	The Cricket mote.	78
Figure 6.2:	Cricket nodes hung on the ceiling.	78
Figure 6.3:	Cricket nodes on the UGV.....	79

Figure 6.4: System block diagram of the experimental setup.	80
Figure 6.5: Trajectory of the one navigation. Besides the symbols labeled in the figure, local minimum points are highlighted by large dots. The part that the large arrow points to is an error movement, which might be caused by reading errors of listeners.	82
Figure 6.6: Potential fields calculated from listener readings during runtime.	82
Figure 7.1: Illustration of a path taken by the UGV using our distributed navigation algorithm in a simple WSN.	83
Figure 7.2: Average number of messages sent by each node in the distributed level assignment algorithm.	85
Figure 7.3: Evaluation of the distributed (top) and centralized (bottom) UGV control algorithms. R_1 shows the ability to control the main navigation direction.	86
Figure 7.4: Evaluation of the UGV control algorithm: R_2 shows the control ability for the step movement.	87
Figure 7.5: Comparison of one-actuator and three-actuator navigation with regards to R_1	88
Figure 7.6: Comparison of the one-actuator and three-actuator navigation algorithms regarding the mission failure rate F	91
Figure 7.7: Number of messages sent per node in the leader election process.	91
Figure 7.8: Performance ratio of the traveled distance by the proposed algorithm and the genetic algorithm (circles), and performance ratio of the traveled distance by the simple algorithm and the genetic algorithm (triangles) in the multi-UGV, multi-destination scenario when UGVs start at the same locations.	93
Figure 7.9: Comparison of results by the proposed algorithm to the simple nearest neighbor algorithm in the single UGV and multiple destination scenario.	94
Figure 7.10: A randomly deployed network with three coverage holes.	95
Figure 7.11: The coverage holes are patched after the patching algorithm.	96
Figure 8.1: Local minimum point is occupied by an obstacle.	98

Figure 8.2: Representation of the size limit on the obstacle. The listeners ring
represents the UGV. 99

ACKNOWLEDGMENTS

I would never have been able to finish my dissertation without the guidance of my committee members, help from group members and support from my family. It is a pleasure to pay tribute to those who made this dissertation possible.

I would like to express my deepest gratitude to my advisor, Dr. Rastko R. Selmic, for his brilliant insight, broad knowledge, and patient advice throughout the research in the past few years. I would like to thank Dr. Christian A. Duncan, who helped me set up the simulation testbed and gave me countless valuable advises and criticisms in all aspects of my research. I would like to thank Dr. Jinko Kanno, who shared ideas and requirements from a mathematician's point of view.

I thank my beloved wife, Xuemei Zhang, for her love, trust and support. I also thank Xuemei for delivering me two lovely angels, Sarah Zhang and Selena Zhang.

I dedicate this dissertation to my parents, Lin Zhang and Meijiu Pan, who have dedicated their entire life for me. I also dedicate this dissertation to my grandparents, Zhixiang Zhang and Xiufang Yang. They never leave me in my heart.

Finally, I would like to offer my regards and blessings to all of those who were important to the successful realization of this dissertation, as well as expressing my apology that I could not mention personally each one.

CHAPTER 1

INTRODUCTION

Developments in MEMS-based sensor technology, low-power RF and A/D design enabled the developments of relatively inexpensive and low-cost wireless sensor systems [14, 20, 26]. Wireless Sensor Network (WSN) technology was developed in parallel with these fields. A WSN is a distributed, self-organized system consisting of sensors that are connected over wireless communication links. WSNs can be used in a variety of applications such as home automation, intelligent traffic control and cyber-physical systems [45, 62, 74]. Different applications of WSNs may have distinct Quality of Service (QoS) restrictions, including connection reliability, time-varying delay and packet loss. In applications where a fully covered sensing domain is preferred [50], such as in habitat monitoring, QoS is based on the coverage area. The WSN is built when connections of sensor and actuator nodes can be automatically established after the deployment is completed. For example, consider the air-drop deployment of a WSN in a remote and hazardous area. Certain areas may not be fully covered due to a random deployment or due to node failures. In such cases, a more refined deployment of the WSN to reach the required QoS is needed.

This chapter provides a brief overview of motivations for this research, as well as the scope of problems considered.

1.1 Motivations

The problem of automatic UGV navigation through a self-organized WSN to patch coverage holes that have been detected is considered. Automatic navigation of UGVs becomes a challenging problem when expensive and energy consuming localization modules such as GPS are not always available since it is not realistic to charge or replace batteries frequently in certain circumstances. As a result, there is no attempt to apply any localization techniques or acquire any coordinate information to assist with the navigation problems.

A general overview of a self-organized network is given in [47]. Briefly, no outside control is necessary for a self-organized network. A self-organized network with inside central controllers is considered as a centralized system while a self-organized network without a central controller can be considered as a decentralized system. For a wireless sensor network system with a central controller, these controllers maintain global information of the whole system. The global information can be obtained either by direct connections from a central controller to each individual node, or by indirect connections such as multi-hop links, where each local package needs to be transmitted in a certain number of hops to reach the central controller. Centralized approaches can suffer from central controller failure and delayed responses to local changes. In the systems where information is gained from indirect connections, high communication demands are needed, which are worsened when the size of the network increases considerably, limiting bandwidth.

Recently, researchers have introduced the sensor node as a small-form-factor embedded system which is capable to a certain extent of processing data. In this experiment (Figure 6.1), one such node, the Cricket mote [53] is used.

Because of these reasons stated above, there is motivation to investigate algorithms that are supposed to run in an event-driven, decentralized manner. In this way, no central controller will exist to provide any command or guidance. Information transfer is realized by messages that are transmitted through wireless links between nodes. Thus, since global information is too expensive to get for every node, each node should be responsible to take its own action based on the limited received messages. Compared to algorithms that proceed in a centralized manner, distributed approaches are typically fast, flexible to changes and robust to individual points' failures. However, due to the limitation of global information, distributed algorithms can produce suboptimal solutions.

1.2 Scope of Work

Buchart deals with coverage hole detecting problems in [10], where nodes on the boundaries of coverage holes can be identified. These identified nodes are called hole boundary nodes. This dissertation focuses on two problems that happen after the hole boundary nodes are identified. First, is an attempt to navigate Unmanned Ground Vehicles (UGVs) to bring in supplemental nodes to these coverage holes. Second, these supplemental nodes are used to patch coverage holes. This process is illustrated in Figure 1.1.

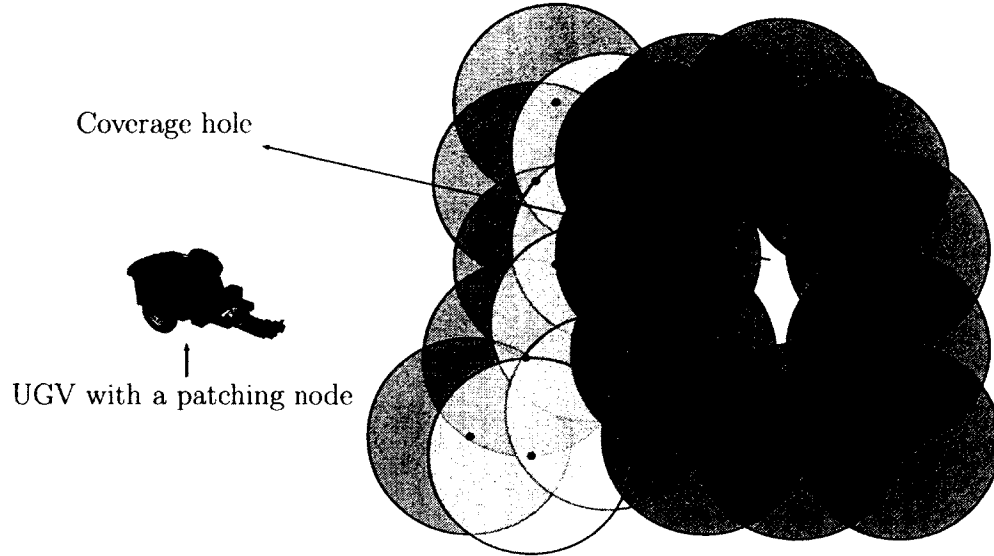


Figure 1.1: Hole patching in a WSN.

The problem of coordinate-free and localization-free UGV navigation in a sensor covered area is explored. Then, a potential-field-based navigation approach is proposed, where the potential field is formulated by actuators in the WSN. Thus, a Wireless Sensor and Actuator Network (WSAN) is used for the navigation. WSANs have become an active research area since they are considered as an enhancement to the traditional WSN. In contrast to traditional sensor networks, which only get information about the physical world (like environment and habitat monitoring, battlefield surveillance and chemical pollution detection), WSANs can make decisions and perform appropriate actions to change or adjust the environment based on gathered information. WSANs do not only monitor the environment passively, but also actively interact with the physical world. For example, in [56], authors use a WSAN to monitor and control combined sewer overflow events in city sewer systems.

Unmanned Ground Vehicles (UGVs) are autonomous mobile robotic platforms that can be deployed in remote and inaccessible environments and are often expected

to substitute for humans in many harsh environments. Placing a GPS module on each sensor node significantly reduces an already limited battery life. Predefined maps or landmarks are usually used in UGV navigation. However, this information is not always available since WSNs are usually deployed in remote and coordinate-free areas. Moreover, these kinds of off-line navigation methods that use prior data make the UGV unable to adapt readily to the dynamic changes of the environment.

By using potential fields generated by a subset of a WSN, it is demonstrated that UGV navigation can be realized by just the interaction between the UGV and static sensor and actuator nodes. The proposed navigation algorithm proceeds in two phases. In the first phase all nodes in the network are classified into different sets or different *levels* according to the distance to certain nodes. Here the distance is referred to as the hop-distance, not the physical distance. Nodes u and v have hop distance one if there is a direct connection between them, as shown in the left figure of Figure 1.2. The solid line means there is a connection between two nodes. The non-adjacent nodes are of distance two if there is a node w with which u and v can both communicate, as shown in the right figure of Figure 1.2. Phase One floods the network from the destination node until all nodes have equivalent hop-mapping assigned. In the case of a specific application of coverage hole patching, the hole boundary nodes are the starting nodes for the process. The flooding process stops when all nodes are updated with a new hop level number. Then in Phase two of the proposed algorithm, a navigation start point is set up according to the position of the UGV, and the navigation rule in the potential field is defined, which can be found in detail in Chapter 4.

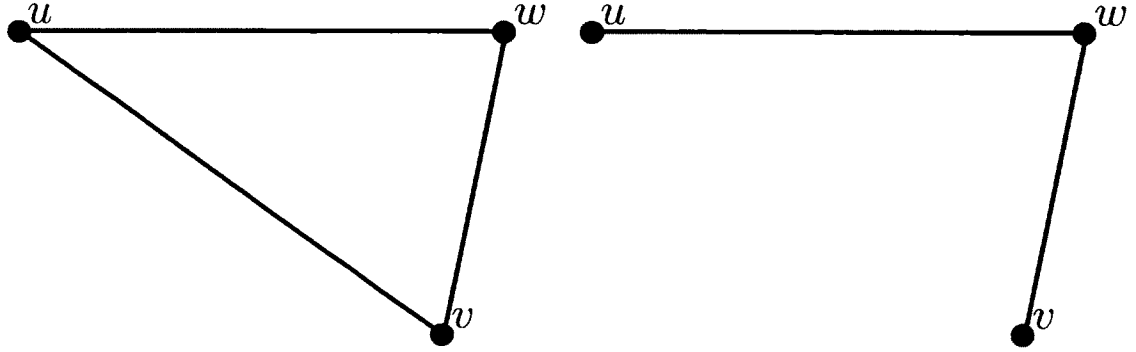


Figure 1.2: Hop distances between nodes: node u and v are with hop distance 1 in the left figure while node u and v are with hop distance 2 in the right.

The same navigation approach to determine the navigation route is used, and the situations that arise when multiple destinations and multiple UGVs are involved are also discussed. Due to the distributed manner of the algorithm, the first problem that needs to be addressed is the identification of different destinations before any path planning. This problem is trivial when a single node is the destination, where the node's ID number can be considered as the destination. However, when each destination consists of a set of nodes (like the coverage holes), a solution must be found to distinguish each destination, also referred to as a task. A leader election algorithm is proposed to be implemented at every node. For example, in the case of coverage hole detection and patching, nodes that are on the hole boundary can trigger the leader election process to identify a unique ID for that specific cluster of nodes. After every destination gets an ID number, it is proposed that a level assignment algorithm starts from the destination nodes. This multiple destination level assignment algorithm is used to calculate the hop distances, hence estimating the actual distances, between nodes and destinations, where after the assignment every node in the WSN stores the hop distance to the various destinations. Finally, the

multi-UGV, multi-destination navigation planning problem is formulated as a task allocation problem, where each UGV is considered to be an agent and each destination a task. Also considered is a special case where multiple destinations exist with only one UGV in operation. This problem can be considered as an open traveling salesman problem, where the UGV is not required to go back to the original position.

Since the hole boundary nodes are treated simply as target destinations for the purpose of UGV navigation without certain underlying assumptions that are conditions or results inherited from the previous hole detection algorithm, this problem can be generalized to any problem that relates to navigation of UGVs to any pre-determined set of target nodes within a WSN.

During the hole-patching process, in order to maximize the area covered for each new node, it is proposed to deploy one new node along the perpendicular bisector of every hole boundary edge. New nodes are added by visiting all the existing hole boundary edges. Since a global map or coordinates are not available, a hole detection algorithm is used iteratively to validate the position for the new node.

A Java-based simulation testbed is developed that is capable of simulating both distributed and centralized navigation algorithms in a WSN. A brief description of the hierarchy and functions of this testbed is provided in Chapter 5. All proposed algorithms are tested in the simulation testbed while all the outputs are plotted in MATLAB.

To begin the process of empirically evaluating the algorithm in real-life scenarios, a hardware testbed is introduced with a single UGV and a wireless sensor and actuator network deployed on the laboratory ceiling, as shown in Figure 6.2. The Cricket

platform [53] is used to build the WSAN and a Pioneer 3-DX [54] as the UGV. The experimental results of the single-UGV and single-destination algorithm are compared with an optimal UGV path between the initial position and the destination in WSAN. Future on-going work will expand this empirical testing to the multiple UGV and multiple destination problem. Also, initial results of the proposed algorithms have been presented in papers [75–78].

The remainder of this dissertation is organized as follows: some related methods are introduced in Chapter 2. Chapter 3 formulates the problems and states general assumptions. Chapter 4 presents detailed descriptions of the proposed algorithms. For certain problems, both centralized and decentralized algorithms are presented for comparison. The description of the construction of the Java-based simulation testbed is given in Chapter 5, followed by the introduction of the hardware experiment set up in Chapter 6. Chapter 7 covers numerical simulation results and discussions. Finally, conclusions and propose future work are discussed in Chapter 8.

CHAPTER 2

RELATED WORK

This chapter provides essential reviews and discussions of papers that are in specific areas of WSNs and Unmanned Ground Vehicle navigation.

2.1 Coverage Problem and Coverage Hole Patching

Each sensor node in a WSN has a limited sensing and communication range. Coverage holes are inevitable in WSNs, especially for those networks which are deployed randomly. The coverage of WSNs is classified into three types: blanket coverage, barrier coverage, and sweep coverage in [34]. Coverage problems in WSNs are generally named k -coverage problems in [2, 34], where k -covered means that every point inside the area of interest is covered by at least k sensors. Authors of [34] introduced an algorithm to analyze the coverage problem by calculating the overlapped area of sensing disks. However, this algorithm requires information of node positions. In the current case, one point is considered to be covered if it is within at least one sensor node's sensing area. The coverage problem considered is a 1-coverage problem. In [10], Buchart presented two centralized algorithms Partitioning Network and Cycle Collapsing Algorithm to detect coverage holes in a WSN, which is modeled by *maximal simplicial complex* [24, 33, 50, 65]. In this dissertation, the coverage hole detecting problem is considered in a decentralized manner in [75], where only locations of

outmost boundary nodes are assumed to know. In [44], Li *et al.* propose a distributed algorithm called *3MeSH* (Triangle Mesh Self-Healing) algorithm, which requires only minimal connectivity information. During the first step, a method to deactivate redundant nodes is provided, through which the number of connections in the graph could be decreased, especially in high density networks. Then, the hole detection algorithm only considers the connectivity information of active nodes.

When there are excessive inactive nodes in a network, Liu *et al.* [46] and Deng *et al.* [25] presented algorithms to activate these sleeping nodes to patch coverage holes. In the case a sensor network consists of mobile sensor nodes, Wu *et al.* [73] proposed an approach to move these redundant nodes to patch holes. In contrast, in current scenario, the network is assumed to be composed of static nodes, where coverage holes cannot be patched by simply activating some sleep nodes. On the other hand, a certain number of UGVs are equipped to move supplemental nodes to patch coverage holes.

2.2 Automatic Unmanned Ground Vehicle Navigation

Voronoi diagrams, visibility graphs and potential fields are well-known techniques to solve the motion-planning problems [3, 8, 28, 36, 49]. Several results have also been shown that the cooperation between mobile robots and wireless sensor networks can enhance a mobile robot's navigation capability [13, 48]. While Voronoi diagrams and visibility graphs require *a priori* knowledge of the workspace map, potential field-based methods do not have the same restriction. In [40], Koren and Borenstein discussed some drawbacks of simple potential field-based navigation, such as problems

with local minima and with oscillations in narrow passages. In [38], Khatib proposed using artificial potential fields for obstacle avoidance in robot motion planning. The method generates attractive potential fields that pull the robot while obstacles generate repulsive potential fields that push the robot away. The artificial potential field is constructed by distances between objects: in [38], Khatib calculated the real distance in coordinate systems; in [9], Borenstein and Koren used sonar sensors to measure the distance; in [43], Li *et al.* used hop distance to roughly represent distance. In [19], a potential field is calculated using temperature, humidity and altitude data, which are acquired by the sensor network. In the current case, when there is no knowledge of coordinates or location and the actuating source can be varied, it is proposed to construct a serial of potential fields by signal strength.

Batalin *et al.* [4] proposed a localization-free navigation method that proceeds in two phases. In the first phase, each node calculates transition probabilities to determine the optimal navigation direction. In the second phase, a more reliable and accurate signal strength based method is employed to drive the robot. In [41], two localization-free, single mobile node navigation algorithms were presented. Periodically, either a measured distance or a hop distance metric between the mobile node and the sensor nodes is used to move the mobile node towards the destination. In both of the approaches, only one sensor node is chosen as a beacon or benchmark to control the moving direction for each step.

Mercker *et al.* [51] discussed the physical motion of a mobile robot in a distributed landmark-free sensor network. In [70], a distributed, location-aware and Voronoi diagram related multi-mobile robots navigation approach was presented.

A credit field is built as the navigation field based on a hop distance; a series of Voronoi diagrams is calculated by the mobile robot to find the path through the network. In [17], P. Chen *et al.* proposed a localized Delaunay triangulation based, distributed guiding navigation protocol that allows for multiple paths and multiple events in the network. Fu *et al.* [29] used a wireless sensor network for indoor robot navigation, employing prior knowledge of sensor positions to localize a robot's position and orientation by acquiring the information of pre-set radio emission sensors. D. Chen *et al.* [16] proposed a set of distributed algorithms for in-network path planning where sensor nodes whose coordinates are known serve as landmarks for the navigation. The algorithms ensure that each source node has at least one safe route to the destination in a dynamic environment. The algorithms are event-based and generally perform better in dynamic networks where they incur much less communication overhead than existing, periodic, flooding-type algorithms. However, in the worst case scenario, for example, when a node that is very close to the destination fails, the performance of the algorithm degrades significantly. In [22], mobile robots were used to establish positions of all sensor nodes, which are not known *a priori*, and then the navigation path is computed and stored in the sensor network. Flying robots can also be employed to repair network connectivity [23]. Alankus *et al.* [1] proposed a set of query strategies that a mobile robot controller can use to periodically collect real-time data from the network and construct a probabilistic road-map for the navigation.

To reduce the communication expense, Buragohain *et al.* [11] introduced a concept of a skeleton graph, which is a sparse subset of the real graph. However, this

algorithm cannot work in coordinate-free situations like this case since the construction of a sub-network is based on the real distances between nodes.

In [43], the robot navigates incrementally along the optimal safest path via an artificial potential field combined with a goal location. Li *et al.* used a hop-distance metric based on the minimum number of hops as a measure of the node's distance from given targets or obstacles. Each node calculates a potential value from the hop distances and potential values from its direct neighbors. The authors proved that the computed path has an upper bounded with respect to the potential integration on the optimal sensor path. This method belongs to one-beacon based navigation since the hop-distance metric is built from node to node. O'Hara *et al.* presented a similar one-beacon based navigation algorithm in [57]. Their experimental results show that the path is 24% longer than the optimal path on average. Chapter 4 will compare the current approach (three-beacon based) to one-beacon based navigation.

2.3 Task Allocation

Batalin *et al.* [5,6] proposed distributed task allocation algorithms using a sensor network. The sensor network is divided into multiple navigation fields based on the priority of tasks that are related to distances from robots to certain tasks. However, it is possible that all the robots might get assigned to the same task if they are originally put in the same navigation field since robots do not participate in the decision making process. When mobile nodes are able to localize themselves in a predefined map, Coltin *et al.* [21] proposed two algorithms to allocate tasks in wireless sensor networks: an auction-based algorithm and a tree-based algorithm. Simulation results demonstrate

that the auction-based algorithm is more efficient regarding the traveling distance while the tree-based algorithm is more efficient regarding the communication cost. Viguria *et al.* [71] presented an auction-based distributed algorithm that can avoid infinite loops caused by a scenario that two robots share the best bids for at least three tasks. However, while this project aims to develop an algorithm without the existence of any central controller, their algorithm is not fully distributed since central robots are required to control all the bids and assign tasks. Parker [61] introduced a distributed behavior-based task allocation software architecture (ALLIANCE) that is robust and flexible. ALLIANCE mainly utilizes sensory data to allocate tasks, while broadcast communication is used to enhance robot's perceptual abilities. An underlying assumption is that each robot should either be able to sense actions from others or in the range of others' communication radius. This assumption is not considered in this method.

2.4 Leader Election

A leader election problem for ring networks is presented in rings [15, 42] and for arbitrary networks in [30, 39]. A good survey of distributed algorithms can be found in [67]. Burns [12] has proved that the lower bound of an asynchronous leader election algorithm is $\Omega(m + n \log n)$. Vasudevan *et al.* [69] proposed an asynchronous leader election algorithm (AEFA) for dynamic networks, where a source node is responsible for initializing and finalizing the algorithm. However, in the current scenario, there is no need to find another leader if a source node already exists.

2.5 Obstacle Avoidance

Obstacle avoidance will be discussed as a possible extension in Chapter 8. As described in [9], a single sonar has some inherent shortcomings such as misreading from ultrasonic noise or neighboring sensors reflection, poor directionality and specular reflection. By assuming a robot can perfectly follow the edges of an obstacle and know the corners of obstacles a prior, Papadimitriou *et al.* [60] analyzed the lower bound on the length of navigation. Borenstein *et al.* [9] developed a vector field histogram-based approach for the real-time and fast obstacle avoidance, where the navigation area is divided into many cells, and obstacles are represented by a modified certainty grid method [27]. The greater the certainty value of a cell, the higher the probability that the cell is occupied by an obstacle. This approach uses coordinates to calculate the certainty value of each cell and assumes the coordinates of the navigation goal are known *a priori*. However, these two conditions can not be satisfied when an obstacle avoidance algorithm is developed in a coordinate-free and localization-free WSN area.

2.6 Hardware Experiment

In Wang and Hu [72], a Cricket platform is used for localization by a trilateration method. A trilateration query protocol is applied in order to localize newly added sensor nodes. In [37], Kapse *et al.* introduced an indoor localization method using Cricket platforms and a Pioneer robot. In addition, Mohammad [55] used both Cricket and Pioneer 3-DX systems to fulfill the navigation task based on the trilateration theory, demonstrating that the robot follows the designated path within the error of 10 cm. With the Cricket platform, Wang and Xiao [18] developed a localization

method based on Maximum Likelihood Estimation (MLE). However, these methods require the coordinate information of each node, which is not always possible. Instead, this project proposes navigation algorithms that are applicable for coordinate-free sensor and actuator networks where precise knowledge of the location of the nodes is not required.

CHAPTER 3

PROBLEM FORMULATION

The main problem addressed is to navigate UGVs in a self-organized WSN such that the UGVs are able to reach target nodes. The problem stems from the hole coverage patching problem where the UGV is required to automatically navigate through the sensor network towards the nodes that have been identified as coverage hole boundary nodes, as shown in Figure 1.1. When there is more than one hole boundary node, the UGV is allowed to navigate to any one of the hole boundary nodes since they are equivalent in terms of hole patching tasks. Though the navigation technique can be used for other purposes requiring traversal to an identified set of nodes, for concreteness the process is explained with regards to the original application of the hole patching problem.

To model the system, certain assumptions are made on the capabilities of the WSN and UGVs. In particular, the problem is considered under the following assumptions:

1. Nodes in the network are identical with regard to both communication and actuation capabilities. Each node is capable of producing an actuating signal with an amplitude a at up to three distinct frequencies f_k for $k \in \{1, 2, 3\}$;
2. Sensor nodes are stationary after the deployment;

3. Communication between nodes is uniform and constant where two nodes can always communicate if and only if they are within communication radius r_c ;
4. The actuating model is omni-directional with actuation signal range $r_a \geq 2r_c$;
5. The UGV has a sufficient control to move in a given direction, i.e., the UGV is a point mass as in [4,43] without any kinematic dynamics;
6. The UGV can communicate with sensor nodes within distance r_c and is equipped with a set L of listener devices capable of detecting actuator signals at frequencies f_k within distance r_a ;
7. The target node(s) are identified before the start of the navigation algorithm at time t_0 , and the UGV and the target node(s) are always connected by a path in the communication graph;
8. Coverage holes or destinations are far enough from each other.

A brief overview is presented before the details of the algorithms. Both centralized and distributed algorithms share the same core concept. First, sensor nodes are classified by the hop-distance from the nearest target node. The hole boundary nodes are labeled, which are identified in the previous process, as hop-0 or level-0 nodes. Then, hop-1 (level-1) nodes, hop-2 (level-2) nodes, etc. are identified, based on communication connections between nodes. Such a classification process is called a level assignment. The level assignment process stops when all nodes are assigned a level number. The UGV does not take part in the level assignment process. Second, the UGV progressively moves towards lower level nodes until reaching any hole boundary edge or hole boundary node. This is accomplished by using an actuator

network, which could be formed by one of several actuator sources including LEDs, magnetic forces, or specific radio frequencies. The WSA_N effectively guides the UGV by generating a potential field. This process is called the UGV control. In general, the level assignment algorithm is the preliminary step for the UGV control algorithm.

At any instant, a subset of the nodes in the network can be transmitting an actuation signal at a given frequency. For time t , let S_k^t , for $k \in \{1, 2, 3\}$, be the set of nodes currently transmitting at frequency f_k and let $S^t = \cup S_k^t$ be the set of all nodes currently actuating. From assumptions, each node $i \in S_k^t$ can generate a radially symmetric potential field U_{ik} at frequency f_k . Three different frequencies, f_1 , f_2 and f_3 , were chosen each time for the active actuators to avoid interference. For each node $i \in S_k^t$, the potential field at each listener $j \in L$ on the UGV, is given by

$$U_{ijk}^t = a \cdot e_{ij}, \quad (3.1)$$

where e_{ij} is the signal strength that listener j gets from node i . Signal strength e_{ij} is inversely proportional to the path loss $(d_{ij})^m$ [63], where d_{ij} is the physical distance from node i to listener j and m is the path loss coefficient, usually $m \geq 1$. For simplicity, it is assumed that $e_{ij} = (d_{ij})^{-m}$. The signal strength e_{ij} becomes infinity when $d_{ij} = 0$. The combined potential field at listener j for frequency f_k is given by

$$U_{jk}^t = \sum_{i \in S_k^t} U_{ijk}^t = a \sum_{i \in S_k^t} (d_{ij})^{-m}, \quad (3.2)$$

where $d_{ij} \neq 0$. The computation complication can be avoided in the case of distance equals zero. For example, when the detected distance d equals zero, d can be set artificially to a every small value. Based on this real potential field, the UGV constructs an artificial potential field at each listener, which it uses to navigate the network,

given by $U_j^t = \sum_k 1/U_{ijk}^t$. In the proposed algorithm, it is ensured that at most one node is transmitting at a specific frequency at any given time. Thus, U_j^t simplifies to

$$U_j^t = \sum_k \frac{1}{U_{ijk}^t} = \frac{1}{a} \sum_{i \in S_k^t} (d_{ij})^m. \quad (3.3)$$

The listener devices are placed on the UGV such that all (but one) listeners $j \in L$ are equally spaced at angles θ_j (to indicate an angle direction to navigate the UGV) on a circle centered around the remaining listener 0. Let θ_0 to indicate the center of the circle, as shown in Figure 3.1. Listeners should be placed such that their separation is above the precision of their distance measurement capabilities. For example, for Cricket systems, the precision is 1-3 cm [53]. The number of listeners and the radius ρ can be changed to adjust the accuracy of the UGV control. Simulation results illustrate the concept of multiple listeners.

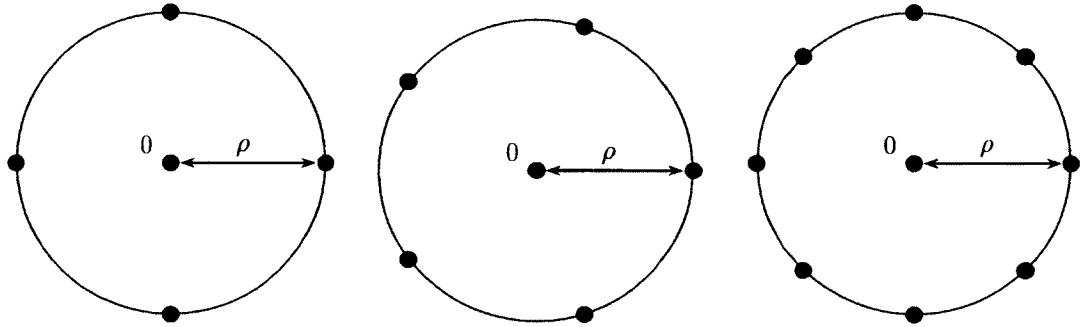


Figure 3.1: Arrays of listeners on the UGV.

At time t , the UGV determines its new relative moving direction θ^t by finding the local minimum value among the potential fields given by

$$\theta^t = \theta_j, \text{ where } U_j^t = \min_{j \in L} U_j^t. \quad (3.4)$$

If $\theta^t = \theta_0$, then the UGV is assumed to have reached a local minimum position.

The UGV moves in discrete steps with a predefined step size, requiring a trade-off between accuracy and energy consumption. In the simulation tests, a step size ρ is used. For larger step sizes, the accuracy will be lower, increasing the likelihood that the UGV oscillates around the local minimum of the potential field. However, in a sparse network with a large sensor communication radius, it may not be energy efficient to use a relatively small and fixed step size as the UGV could repeatedly be adjusting its course. Making the step size dynamic and analyzing the trade-off under real-life situations will be considered in future work.

When multiple UGVs and multiple destinations arise during the same time period, a decentralized allocation process is needed to allocate each UGV a distinct destination. Each UGV can take only one destination at a time and each task only needs one UGV to execute. Supposing M destinations and N UGVs, an Integer Linear Programming (ILP) problem can be formulated as follows:

- the set of M destinations or tasks is denoted as $\{T_1, T_2, \dots, T_M\}$
- the set of M relative weights of the tasks is denoted as $\{w_1, w_2, \dots, w_M\}$
- the set of N UGVs is denoted as $\{I_1, I_2, \dots, I_N\}$
- the nonnegative cost of UGV I_i for task T_j is C_{ij} , where $1 \leq i \leq N$ and $1 \leq j \leq M$.

As in most real-life applications, only problems when $M \geq N$ are considered. The task allocation problem is to find an optimal allocation of UGVs to accomplish all tasks. An allocation can be considered as a set of UGV-task pairs (I_i, T_j) . Now the problem can be formulated as an ILP problem – find non-negative integers α_{ij} that maximize

$$\sum_{i,j} \alpha_{ij} C_{ij} w_j, \quad (3.5)$$

subject to

$$\begin{aligned} \sum_j \alpha_{ij} &\geq 1, \quad \forall i, \\ \sum_i \alpha_{ij} &= 1, \quad \forall j. \end{aligned} \quad (3.6)$$

The Equation (3.5) is the overall system cost, while Equation (3.6) defines the constraints. Let define each individual utility as

$$C_{ij} = \frac{1}{d_{ij}}, \quad (3.7)$$

where d_{ij} is the hop distance between UGV I_i and task T_j . The definition of weight w_j varies for different applications. For the initial application, the weight w_j of task T_j can be calculated as the number of nodes involved in T_j . If a solution exists, it can be obtained using ILP.

Also considered is the special case when multiple holes exist with only one available UGV in the network. The single-UGV, multi-destination navigation problem is similar to the Traveling Salesmen Problem (TSP), where the UGV aims to intentionally visit all destinations once. Still, some destinations can be visited more than once if they are on the path aimed at other destinations. The UGV is not required to go back to the original start point.

Before the leader election process starts, it is an open problem to determine which nodes should participate and respond to it. Although in the original problem, this process can only proceed in nodes identified themselves as hole boundary nodes [75],

there is no general answer to all applications. To focus on the topic, it is assumed that nodes can identify themselves.

Detailed introduction of the proposed algorithms as well as pseudo codes are presented in Chapter 4.

CHAPTER 4

ALGORITHMS

This chapter describes algorithms for network hop-distance identification and UGV navigation control, as well as coverage hole patching. A general logic view of proposed algorithms is shown in Figure 4.1.

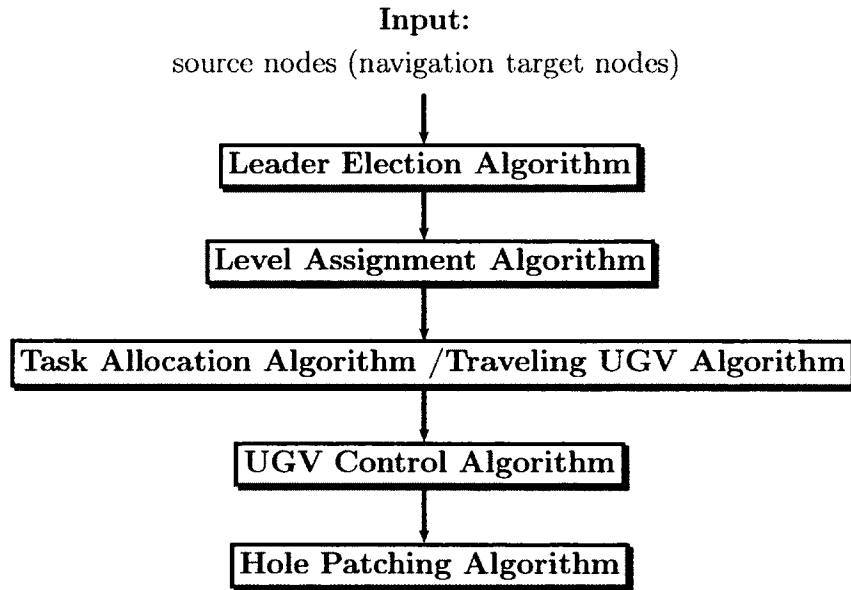


Figure 4.1: Logic flow of the proposed algorithms.

The Leader Election Algorithm is required to distinguish different targets when there are multiple targets exist. After targets are identified, the Level Assignment Algorithm will be triggered to estimate the nodes' distances to each target. Then, if there are multiple UGVs ready to be deployed, the Task Allocation Algorithm will

be applied to assign UGVs to different targets (tasks). Alternatively, the Traveling UGV Algorithm will be used to determine the traveling order to the targets. Once the UGV has a target selected/assigned, the navigation will proceed by the UGV Control Algorithm. Finally, the Hole Patching Algorithm will be performed if there are coverage holes needed to be patched in the network.

The Leader Election Algorithm and the Level Assignment Algorithm are prerequisite steps prepared for the later algorithms. The Task Allocation Algorithm and the Traveling UGV Algorithm deal with cases when multiple targets and multiple UGVs are involved. The UGV movement or UGV navigation is controlled by the UGV control Algorithm, which can be considered as the core algorithm. The Hole Patching Algorithm is related to the original application where coverage holes are aimed to be patched.

A simple single-UGV, single-destination navigation problem is first considered. Two algorithms will be included related to this problem: the Level Assignment Algorithm and the UGV Control Algorithm. A centralized algorithm is presented, followed by a corresponding distributed version. In the centralized algorithm, there exists a central controller, which connects to all the nodes and has full access to all information in the nodes. In this case, the cost of transmitting messages is not considered since the data are processed and stored locally inside the controller. In the distributed algorithm, each node is considered as an individual processing and storage unit. A main difference between centralized and distributed algorithms is the way they retrieve and process data: for the centralized algorithm, data are retrieved, processed and stored inside the central controller while for the distributed algorithm,

data processing in the individual nodes is heavily coupled with the exchange of data via radio communication, where delays can be common. Therefore, space complexity are discussed and time complexity in the centralized algorithm and communication complexity in the distributed algorithm, respectively.

When considering the multi-UGV, multi-destination navigation problem, the following algorithms are used in navigation and control: Timer-based Leader Election Algorithm, Distributed Multi-Destination Level Number Assignment Algorithm and WSN-Aided Greedy Task Allocation Algorithm. In the special case of a single-UGV, multi-destination navigation problem, the WSN-Aided Nearest Neighbor Algorithm is presented.

After the algorithms for UGVs navigation, a Hole Patching Algorithm (HPA) is presented for cases when hole boundary nodes are identified. Hole patching nodes are deployed along virtual perpendicular bisectors of each hole boundary edge. Since no additional information is available about either the coverage hole or nodes' coordinates, some redundant nodes might be added to the network, and the algorithm is not optimal in terms of number of added nodes.

4.1 Single-UGV, Single-Destination Case

4.1.1 Level Assignment Algorithm

The control algorithm requires that each node has a graph theoretic notion of its distance to the target node(s), called the *hop distance*. A *hop* is simply a communication link from one node to another. Thus, the hop distance between two nodes is equivalent to the smallest number of edges in all paths in the communication

graph between them. The *hop distance* h_{ij} is defined as the number of hops from node i to node j .

4.1.1.1 Centralized Algorithm

In the centralized level assignment algorithm, data are processed and stored inside the central controller. The controller arranges nodes with the same hop distance to targets in the same set. Let S be the set of all nodes in the network and $S(0)$ be the set of target nodes (hole boundary nodes), where $S(0) \subset S$. The level assignment process starts from the nodes in $S(0)$, defined as hop 0 nodes. The subset $S(l) \subset S$ is defined as $S(l) = \{i \mid h_{io} = l\}$, where $o \in S(0)$. Consequently, hop-1 nodes belong to $S(1)$, hop-2 nodes belong to $S(2)$, etc. Once node $i \in S$ is added to $S(l)$, it is said that node i is assigned its level number. Level numbers are assigned in ascending order until all nodes receive a level number. Algorithm 1 presents the centralized level assignment algorithm.

Algorithm 1 Centralized Level Assignment Algorithm for Single-Destination WSAN

```

1: Set  $l = 0$ 
2: for hole boundary node  $i$  do
3:    $i \rightarrow S(l)$ 
4: end for
5: while there still exist node(s) not yet leveled do
6:   for node  $j \in S(l)$  do
7:     for node  $k \in N(j)$  do
8:       if  $k$  does not belong to  $S(l)$  or another lower level set then
9:          $k \rightarrow S(l + 1)$ 
10:      end if
11:    end for
12:  end for
13:   $l = l + 1$ 
14: end while

```

Algorithm 1 is based on the Breadth First Search Algorithm (BFS), possibly with multiple source nodes. Similarly as in BFS, the time complexity of Algorithm 1 is bounded by the number of nodes in the network, n , as well as the number of communication links, m . It has a running time of $O(n + m)$ and uses $O(n)$ additional space as the level in every node in the network must be stored. There is room for improvement, particularly if the UGV is relatively close to the target node. If target nodes are initially within range of the UGV, which could be checked with a simple ping, the level assignment process can be terminated once the level of one of these nodes has been identified. As described in the next subsection, the proposed navigation algorithm always moves the UGV to lower leveled nodes. For a specific target, no higher level is required if the UGV has detected a lower level. Consequently, if T is the number of target nodes, let b be the maximum degree, number of neighbors, of any node in the network, and D be the hop distance from the UGV to one of the target nodes, the algorithm's time and space complexity can also be bounded by $O(Tb^D)$.

4.1.1.2 Distributed Algorithm

In a distributed system, every node has a limited ability to store and process data, and data are shared by direct transmissions between nodes. How to modify the centralized algorithm to perform a level assignment on a distributed system is shown. For each node, let the *level assignment* l represent the current *known* shortest hop distance from the node to any target node. Initially, l is 0 for all target nodes and infinity for all other nodes. As the algorithm proceeds, a node adjusts its level number whenever it receives a message indicating a shorter hop distance, subsequently

transmitting its revised level number to all of its neighbors. This process is similar to a distributed shortest path problem from target nodes to all other nodes, with the weight of each edge equaling one. Many distributed shortest-path finding methods already exist [31, 35, 52, 66], but they focus more on providing algorithms to handle changes in network topology. However, node or link failures are not considered when developing the Level Assignment Algorithm since, unlike these prior algorithms, it is not needed as an independent algorithm. The level assignment is processed mainly for the UGV control algorithm, presented in Section 4.1.2.

The level assignment algorithm needs to be both simple and efficient. In case of communication failures, the UGV can adjust its navigation. The proposed algorithm is simpler than the complex distributed shortest path algorithms [31, 35, 52, 66] due to the following facts: first, it avoids loops in finding the minimum weights by using unit weight for every edge; second, the messages transmitted in the algorithm are very simple, only the local level number is needed; third, the algorithm does not employ any technique to detect changes in the network during the Level Assignment Algorithm, which is helpful for the time and energy savings in WSAWs.

The original application considered here is to navigate a UGV towards a single target hole consisting of possibly many target nodes, where the entire set of target nodes is considered as a single target. Algorithm 2 presents the pseudo-code of the Level Assignment Algorithm for the single-target navigation problem. The process of Algorithm 2 is illustrated in Figure 4.2. The level number of a node only depends on the level number of its neighbors.

Algorithm 2 Distributed Level Number Assignment Algorithm for Single-Destination WSN

```

1: if node is target node then
2:    $l \leftarrow 0$ 
3:   broadcast  $l$ 
4: else
5:    $l \leftarrow \infty$ 
6: end if ▷ Initialization phase
7: while time remaining do
8:    $l_r \leftarrow$  the received level number from a neighbor
9:   if  $l > l_r + 1$  then
10:     $l \leftarrow l_r + 1$ 
11:    broadcast  $l$ 
12:   end if
13: end while ▷ Level number assignment

```

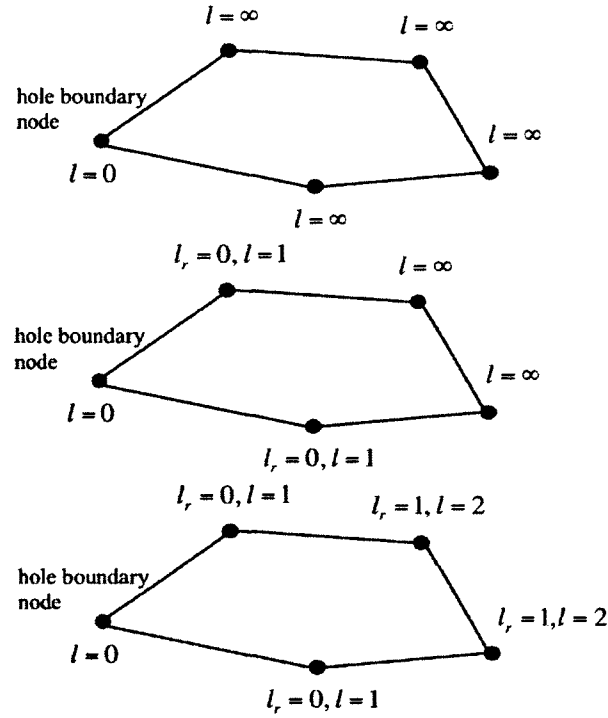


Figure 4.2: Successive processes of Algorithm 2.

After receiving direct messages from the target nodes, 1-hop neighbors self-identify as level-1 nodes. Subsequently, level-2 nodes self-identify after receiving messages from level-1 nodes, etc. Every node eventually receives the lowest level number possible under the assumption that there are no topological changes in the

communication graph during this process. What is essential and addressed in different manners in other schemes is when to terminate the process.

This study is more concerned with communication cost over performance time. Therefore, it is assumed that there is an estimated bound on the time it takes a single message to propagate from a target node to every other node in the network. In particular, it is assumed that the time it takes for a message to pass from a target node to all other nodes is $O(D)$. Clearly, $D < n$ but it could be significantly less in practice. Therefore, each node runs the level assignment process for some factor of D units of time. In practice, this algorithm can be event driven. Except when the node(s) self-triggers the algorithm, updates of level numbers can simply be a part of the regular message retrieval system whereby the nodes can update their level numbers as new information arrives. The proposed navigation algorithm could be adapted for this situation; however, in the analysis presented here the simplified model is used.

The *communication complexity* is defined as the maximum number of messages transmitted during the execution as in [35,66,68]. In this scenario, a message broadcast to multiple neighbors counts as one underlying cost. Since Algorithm 2 is event driven, messages are generated exclusively when a level number changes. Therefore, the communication complexity is asymptotically bounded by the maximum number of times that a level number changes. In [31,52,66], the authors discuss communication complexity for synchronous communication models. The distributed model looks at asynchronous communication resulting in the inevitability of redundant messages because of potential transmission delays, as can be found, for example in [7, Chapter 5].

Suppose there are n nodes in the network and k target nodes. During the initialization phase of Algorithm 2, only target nodes send a message, yielding a communication complexity of $O(1)$ for each target node and $O(k)$ for all target nodes. In the level number assignment phase of Algorithm 2, the if-condition is always false for target nodes. For every non-target node, the first new level value received must necessarily be no more than $n - k$. Since each broadcast by a node occurs exclusively when the level number decreases until reaching a minimum of at least one, each node can therefore broadcast at most $n - k$ times. Thus, the total number of messages generated is $O(k + (n - k) \cdot (n - k))$ or $O(n^2)$ where k is a constant. In practice, particularly when used for finding holes in a coverage area, the graph is expected to be sparse. Meanwhile, if there is an upper bound on the number of neighbors for each node and some additional assumptions on the communication delays are also made, the expected complexity can be improved to $O(n)$ according to the theorems presented in [68].

4.1.2 UGV Control Algorithm

The control algorithm presented here uses similar concepts as in [64]. However, a potential field is generated using a series of three actuator nodes. There are two main advantages for using multiple actuators: first, the system is more robust to node failures and more tolerant of noise. For example, in an algorithm with only one actuator node (beacon), when the connection (actuation signal, not communication signal) between the UGV and the actuator node is lost because of node failure or noise, new communications between the UGV and its neighboring nodes have to

be established to find an alternative actuator node in order to continue the UGV's navigation progress. However, when there are three actuators, the algorithm can be tuned to tolerate up to two failures of actuating connections. In such a case, with no need to stop and establish new communication connections with other nodes, the UGV can instead move directly towards the active actuator, which is equivalent to a one-actuator algorithm. Chapter 7 shows simulation results that compare the UGV control algorithm with one and three actuators. Second, using three actuator nodes offers additional flexibility in terms of the generated potential field, compared to the case with only a single actuator node. For example, in a real-time experiment the movement of the UGV can be controlled in order to avoid certain coverage areas by adjusting the amplitude of a for each node independently. The shape of the potential field can be adjusted with three adjustable actuators, allowing for more variation in the UGV path compared with the control using only one actuator. Only three actuators are used since without additional assumptions the presence of a fourth node to act as an actuator cannot be guaranteed. When using more actuators, there is also a trade-off between energy required by the additional actuators and energy saved in reducing the navigation traveled distance of the UGV. Though worthy of further exploration, the analysis of this trade-off using specific actual physical systems is reserved for future work.

To avoid the UGV getting stuck during the navigation, it is important for the network to control which three actuators are active. Suppose the three active actuators are labeled nodes A , B , and C . Let node B be a neighbor of the UGV that has the lowest level number among all the neighbors of the UGV, for example node 2 in

Figure 4.3. When there is more than one neighbor with the same lowest level number, the UGV arbitrarily picks one of them. Once node B is chosen, two of its neighbors are chosen as nodes A and C , where at least one of nodes A and C should be one level lower than node B . It is possible that in the initial step the first node chosen for node B has only one neighbor. However, as is discussed shortly, this condition is trivial. The UGV navigation can be fulfilled by sequentially switching these actuator triplets. That is, three active actuators are used to generate a potential field to drive the UGV to a specific position (a local minimum point) based on Equation 3.3. As proven in Lemma 4.1, the UGV converges to the local minimum of the potential field located inside the area within distance r_c of all three active actuator nodes. Assumption 6 ensures that the UGV can always hear signals from the three active actuators.

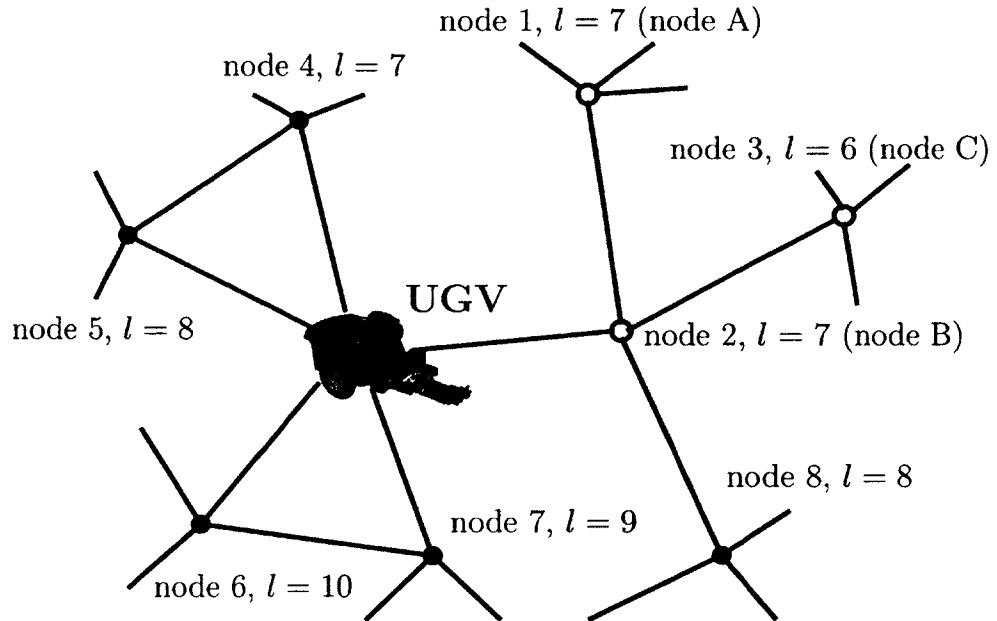


Figure 4.3: UGV navigation in a WSN, where each node's ID, level number and connections are shown. At this step in the example, we have the potential field generated by the triplet of actuator nodes A , B , and C , shown as circles, which are assigned to nodes 1, 2, and 3, respectively.

Due to the original problem statement and the application to reach hole boundary nodes, as well as for simplicity reasons, this dissertation does not allow the UGV to go back to any higher level nodes. However, it is possible and feasible to make the UGV tolerate some extent of backtracking. This is useful when the UGV cannot find any lower level nodes due, for example, to node failures or encounters certain dangerous situations. In this dissertation, the navigation is considered to have failed if the UGV cannot find any lower level node, which can only happen if there is a topological change in the network.

Lemma 4.1. *Any local minimum point p of the potential field ABC is located in the area within distance r_c of all three active actuator nodes A , B and C .*

Proof. Define d_{ij} to be the physical (real) distance between two points i and j . That is, the distance from node i to a local minimum point p can be denoted as d_{ip} . Without loss of generality, $a = 1$ can be set in Equation 3.3, making the combined potential field at point p to be $U_p = (d_{Ap})^m + (d_{Bp})^m + (d_{Cp})^m$. Assume for the sake of contradiction that the lemma is false and that p lies outside the stated area. Then at least one of the distances is larger than r_c . There are two situations, as shown in Figure 4.4. First, assume that B is furthest from p . Let q be a point infinitesimally close to p on the ray extending from B to p . That is, $q = p + \epsilon(B - p)$ for some $\epsilon > 0$. Observe that $d_{qB} < d_{pB}$. Now examine the triangle formed by B , A and p . Since the edge from B to A has length $d_{BA} \leq r_c$ and since the edge from B to p has length $d_{Bp} > r_c$, the angle at p must have value less than 90° as edge BA cannot be the longest side. But this means that q lies inside the circle centered at A of radius d_{Ap} , for sufficiently

small ϵ . Therefore, $d_{Aq} < d_{Ap}$. Similarly, $d_{Cq} < d_{Cp}$ can be shown. This implies that $U_q = (d_{Aq})^m + (d_{Bq})^m + (d_{Cq})^m < (d_{Ap})^m + (d_{Bp})^m + (d_{Cp})^m = U_p$. This contradicts the fact that p is a local minimum.

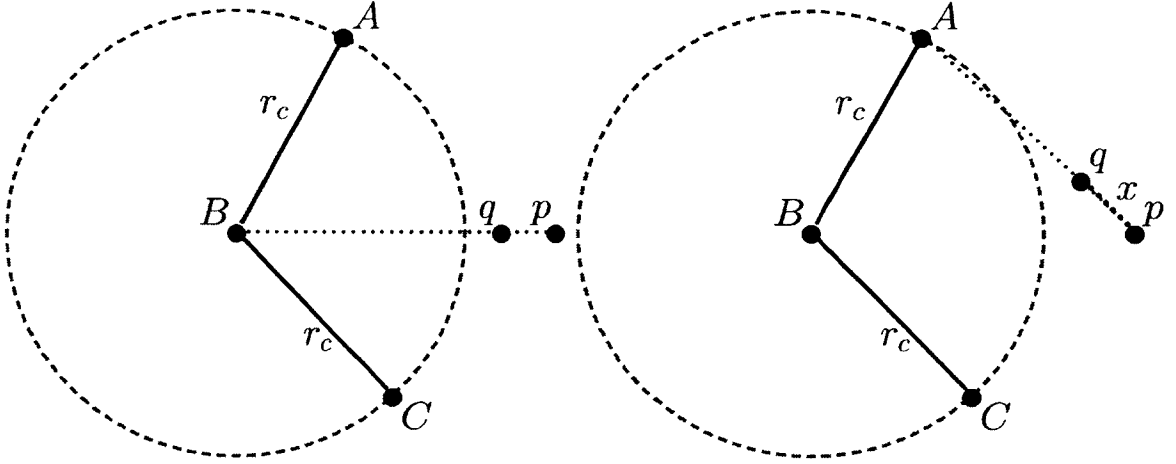


Figure 4.4: Illustrated cases in the proof of Lemma 4.1.

Now, assume that A is furthest from p , the case for C being analogous. Again let $q = p + \epsilon(A - p)$ be a point infinitesimally close to p . Using the property of the triangle formed by A , B and p as before, it can be shown that $d_{Bq} < d_{Bp}$. However, the case for node C is a bit trickier. If $d_{AC} \leq r_c$, then the triangle argument can again be used to show that $d_{Cq} < d_{Cp}$ yielding the contradiction that $U_q < U_p$. However, it is possible that $d_{AC} > r_c$. Now look at the change in the sum of the two distance terms associated with A and C in the movement from p towards A . That is, consider $\Delta = (d_{Aq})^m + (d_{Cq})^m - (d_{Ap})^m - (d_{Cp})^m$. Observe from the choice of q , that $d_{Aq} = d_{Ap} - x$ for some x infinitesimally close to 0 and that $d_{Cq} \leq d_{Cp} + x$. Thus, let $\Delta \leq f(x) = (d_{Ap} - x)^m + (d_{Cp} + x)^m - (d_{Ap})^m - (d_{Cp})^m$. Clearly, $f(0) = 0$. If the first derivative of this function is located at 0, it can be seen that $f'(x) = m((d_{Cp} + x)^{m-1} - (d_{Ap} - x)^{m-1})$.

Consequently, $f'(0) = m((d_{Cp})^{m-1} - (d_{Ap})^{m-1}) \leq 0$ since from the assumption $d_{Ap} \geq d_{Cp}$ and $m \geq 1$. This means that $f(x) \leq 0$ for x infinitesimally close to 0. So, $U_q - U_p = (d_{Aq})^m + (d_{Bq})^m + (d_{Cq})^m - (d_{Ap})^m - (d_{Bp})^m - (d_{Cp})^m = (d_{Bq})^m - (d_{Bp})^m + \Delta$. Since $\Delta \leq f(x) \leq 0$ and $d_{Bq} < d_{Bp}$, $U_q - U_p < 0$ which again contradicts the fact that p was a local minimum. \square

Lemma 4.1 proves that when the amplitudes of actuating signal strength from nodes A, B and C are equal, the UGV that can arrive at the local minimum point of the current potential field (generated by nodes A, B and C) will be able to communicate to all of the three nodes A, B and C . Based on simulation results, when the amplitudes are not equal and can be adjusted, it is found that the local minimum point is still located inside the communication range of nodes A, B and C , as illustrated in Figure 4.5, where nodes A, B and C are arbitrarily placed. As can be seen from the upper figure of Figure 4.5, there is only one minimum point in the defined area, indicating the local minimum of Equation 3.3 is also the global minimum or the UGV can at least always find the right direction until reaching the minimum point. In the lower figure of Figure 4.5, one can see that the local minimum point is always located in the intersection (darkest area shown) of the communication radii of nodes A, B and C , which also validates Lemma 4.1.

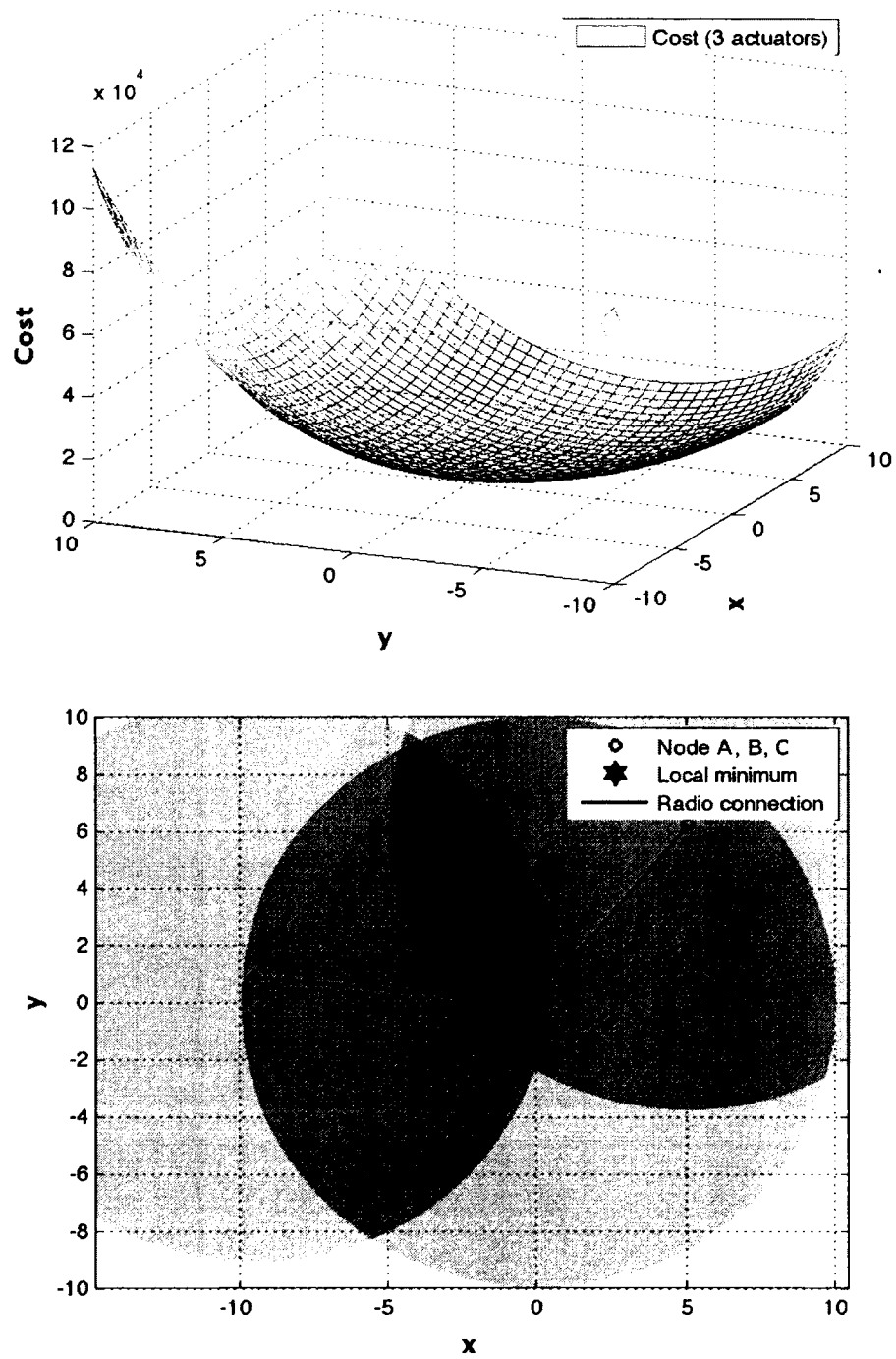


Figure 4.5: An example of Equation 3.3, where nodes A, B and C are arbitrarily placed.

4.1.2.1 Centralized UGV Control Algorithm

In the centralized algorithm, the central controller has the level numbers of all nodes. To take advantage of this property and for simplicity, the centralized control algorithm proceeds off-line since the working sequences of the actuators can be predetermined as in [64]. After Algorithm 1 is finished, the central controller constructs a subset of nodes for navigation, which is called the *navigation nodes*. Navigation nodes are classified as base nodes and assist nodes, as shown in Figure 4.6.

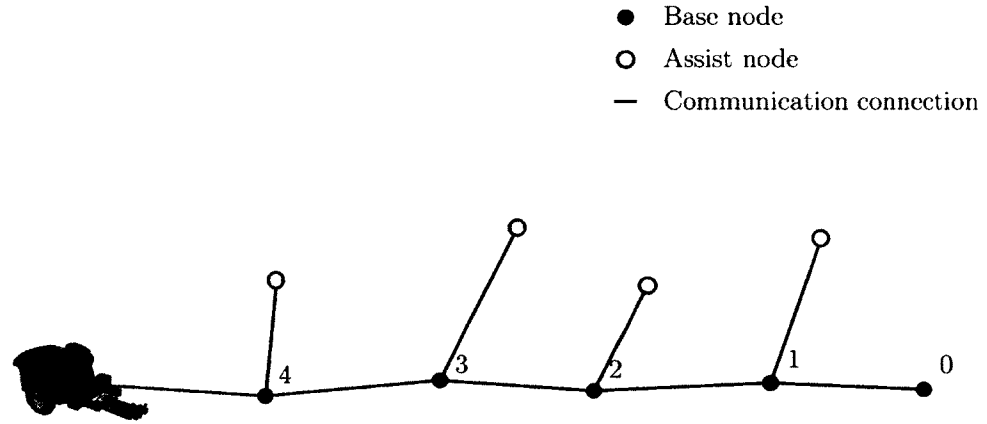


Figure 4.6: General view of navigation-assisted nodes in the centralized control algorithm.

The base nodes, which control the main navigation direction form a sequence of nodes from higher level nodes starting with a neighboring node of the UGV to a hop 0 node, guaranteeing that the UGV can navigate to the destination. Assist nodes are chosen to generate the potential fields. Algorithm 3 provides the pseudo-code for the centralized control algorithm where the initial node p is a node originally within communication range of the UGV.

The procedure for determining base nodes and assist nodes in Algorithm 3 proceeds iteratively with each iteration taking at most $O(b)$ time, where b has been defined as the maximum number of neighbors of each node (also the maximum degree of the underlying communication graph). Each iteration of the moving phase takes constant time except for the delay in waiting for the UGV to navigate to a local minimum. Since each iteration decreases a level, the process terminates in at most D iterations. Thus, the total time complexity is $O(bD)$. Since the algorithm needs to store a base node and an assist node for each successive level, the space complexity of Algorithm 3 is $O(D)$.

Algorithm 3 Centralized UGV Control Algorithm for Single-Target WSAN

```

1:                                     ▷ Code to construct base nodes
2:  $l$  is the level of the initial node  $p$ 
3:  $L = l$ 
4: put  $p$  into  $baseNode[L]$ 
5: while  $L > 0$  do
6:   search for an arbitrary lower level neighbor of  $baseNode[L]$ 
7:   put the neighbor into  $baseNode[L - 1]$ 
8:    $L = L - 1$ 
9: end while
10:                                     ▷ Code to construct assist nodes
11:  $L = l$ 
12: while  $L > 0$  do
13:   search for any neighbor of  $baseNode[L]$  that is not the node in  $baseNode[L - 1]$ 
14:   put the neighbor into  $assistNode[L]$ 
15:    $L = L - 1$ 
16: end while
17:                                     ▷ Code to move the UGV
18:  $L = l$ 
19: while  $L > 0$  do
20:   turn on actuators  $baseNode[L]$ ,  $baseNode[L - 1]$  and  $assistNode[L]$ 
21:   navigate UGV to the local minimum of the three active actuators
22:   turn off actuators  $baseNode[L]$  and  $assistNode[L]$ 
23:    $L = L - 1$ 
24: end while

```

4.1.2.2 Distributed UGV Control Algorithm

The communication in the distributed WSN uses an event-driven and message-passing framework. A single thread of each node, responsible for handling communication and processing, runs in an infinite loop that continuously checks for events on a queue and based on the type processes that event. In particular, if the event is a new message from another node, it reads the new message and processes the message in a similar manner. Algorithm 4 shows the general framework, though there are many other valid ways possible.

Algorithm 4 Event-Based System Framework

```

1: procedure MAINTHREAD
2:   ▷ Initialize some global variables
3:   leaderFlag ← false                                     ▷ True when sensor alarm triggered
4:   loop
5:     event ← GETNEXTEVENT
6:     if event.type = newMessage then
7:       PROCESSMESSAGE
8:     else if event.type = sensorEvent then
9:       PROCESSSENSOREVENT
10:    else if event.type = leaderAlarmEvent then
11:      PROCESSLEADERALARM
12:    else if event.type = activateHopCountEvent then
13:      PROCESSACTIVATEHOPCOUNT
14:    end if
15:  end loop
16: end procedure
17:
18: procedure PROCESSMESSAGE
19:   m ← RECEIVEMESSAGE
20:   if m.type = exit then
21:     EXIT                                                    ▷ Terminate the program
22:   else if m.type = leaderElection then
23:     HANDLELEADERELECTIONMESSAGE(m.id, m.l)
24:   else if m.type = leaderElected then
25:     HANDLELEADERELECTEDMESSAGE(m.id)
26:   else if m.type = levelAssignment then
27:     HANDLELEVELASSIGNMENTMESSAGE(m.id, m.l)
28:   end if
29: end procedure

```

The subsections discuss the individual events and message types. Also note that due to power constraints in WSNs, it is not practical to use a common clock. Thus, the framework is built on an asynchronous model. The assumption is made that times are relatively the same, e.g. five seconds on one system is about five seconds on the other.

A distributed, on-line navigation algorithm is proposed that proceeds in a series of steps. The distributed control algorithm is an improved version of the centralized one. In each step, there are two phases: a communication phase where the specific potential field is determined for an intermediate target area and a navigation phase where the UGV moves through the field towards the intermediate target area. The specific active potential field is determined during the communication phase. In the navigation phase, the UGV first calculates the next moving direction based on Equation 3.3 and then moves by a predefined step size in that direction, after which the UGV calculates a new direction. When the UGV reaches a local minimum of the potential field (within a margin of error), the current step is completed and the communication phase of the next step starts.

At the initial step and whenever the UGV reaches a local minimum, the algorithm switches to the communication phase, where it assigns a triplet of actuator nodes.

From Algorithm 2, it is known that node B , unless it is a target node, has at least one neighbor that is at a level lower than itself. In the UGV control algorithm, node B assigns the role of node C to one of its lower-leveled neighbors and then arbitrarily picks one other neighbor as node A . The triplet of actuator nodes A , B

and C generates the active potential field for the following navigation phase. For simplicity, in the rest of this dissertation, it is assumed communication between the UGV and the triplet of actuator nodes is done primarily through node B .

Lemma 4.1 guarantees that the UGV can always connect to a node that has a lower level than the previous node B after reaching the local minimal point. That is, the next communication phase is guaranteed to pick a new node B for the next potential field whose level number is at least as low as the previous node C . In general, when the UGV ultimately reaches a target node, there exists a series of nodes from high level nodes to the final level-0 node along the UGV's path that each act as node B during some step of the UGV control. The fact that the UGV can always communicate with node B also implies that it can detect the actuator signals from all three active actuator nodes as the distance from the UGV to the farthest of these three nodes is at most $2r_c$, and by Assumption 4 there is $2r_c \leq r_a$.

Another consequence of Lemma 4.1 is that the UGV can send a single command to nodes A , B and C to turn the actuators off when the current step is completed, which helps to conserve energy in the network nodes.

In the pseudo-code for the distributed control algorithm (Algorithm 5), the specifics of the process for choosing nodes A , B and C are not included, which has been discussed earlier. Communication details, such as waiting time to receive messages, are not discussed here. The communication complexity of the control algorithm is fairly straightforward. Let D be the hop distance from the UGV to one of the target nodes. At each phase, at most a constant number of messages is transmitted from the UGV to establish and turn on and off a triplet of actuator nodes at the current

UGV location. Each actuator node responds at most once to this call. This means that there are $O(D)$ messages transmitted by the UGV, and each node in the network transmits at most $O(1)$ messages, though in practice far fewer nodes will be involved. Thus, the communication complexity is $O(n)$.

Algorithm 5 Distributed UGV Control Algorithm for Single-Target WSAN

```

1: ▷ Code for the UGV
2: repeat
3:   broadcast a string "UGV_request"
4:   wait for response from all neighbors
5:   select node  $B$ , the neighbor with the smallest level number
6:   send message "UGV_nB_on" to  $B$ 
7:   receive node ids of  $A$  and  $C$  from  $B$ 
8:   repeat
9:     listen for actuator signals from nodes  $A$ ,  $B$  and  $C$ 
10:    calculate potential field at each listener
11:    move towards the minimum in the potential field
12:  until at a local minimum
13:  send message "UGV_off" to nodes  $A$ ,  $B$  and  $C$ 
14: until node  $B$  is a target node
15:
16: ▷ Code for all network nodes
17: turn off actuator
18: loop
19:    $m \rightarrow$  the received string from a neighbor
20:   if  $m ==$  "UGV_request" then
21:     send message with level  $l$  to the UGV
22:   else if  $m ==$  "UGV_nB_on" then
23:     broadcast the string ("NB_N" +  $l$ )
24:     wait for response from neighbors
25:     select nodes  $A$  and  $C$  based on lowest level numbers received
26:     send message "NB" to nodes  $A$  and  $C$ 
27:     send ids of  $A$  and  $C$  to the UGV
28:     turn on actuator
29:   else if  $m ==$  "UGV_off" then
30:     turn off actuator
31:   else if  $m ==$  "NB" then
32:     turn on actuator
33:   else if  $m ==$  "NB_N" +  $B.l$  then
34:     ▷  $B.l$  is the level number from sender
35:     if  $l < B.l$  then
36:       send level number  $l$  to  $B$ 
37:     end if
38:   end if
39: end loop

```

Special Case – A Node With Degree One Exists: It is necessary to examine the case where there is a degree-one node i . Based on Algorithm 2, every node except level-0 nodes identifies its own level number by adding 1 to the smallest level number of its neighbors. Being a degree-one node, if i can get any valid level number, its level number has to be one level higher than its neighbor, so there is no higher level neighbor to node i . As the UGV always travels from a higher level node to a lower level node, even if the UGV detects node i in its vicinity during its navigation, it always can pick another suitable node B with degree at least two. Thus, the UGV is only forced to assign node i as node B if the UGV is in the initial position of the navigation. There are many ways to avoid this situation. For example, the UGV can deploy one node at its initial position to act as the third node needed to construct a potential field.

Contrary to initial assumptions, if the network topology of the WSAN changes, it is possible for the UGV to be jammed somewhere in the middle of navigation because a lower level node cannot be found. Though this situation is not formally covered in this dissertation, it is possible to solve this problem once some corrective algorithms (including the algorithms presented in [75]) are triggered to start over again.

4.1.2.3 Comparisons with Other Algorithms

There are two major differences between the presented algorithms and the navigation protocol presented by Li *et al.* in [43]: first, it is proposed to use three actuators to generate a potential field while Li *et al.* proposed that each node calculates

a potential field by received hop count. Second, because of the difference in generating the potential field, in this algorithm, the UGV will move through the WSN in the space between nodes (can be found in Figure 7.1 in Chapter 7) while Li *et al.* proposed that the mobile node moves from one node to the next node until it reaches the destination.

A three-actuator algorithm can always be turned into a one-beacon algorithm by activating only one of the actuators. In a centralized case, the base nodes (without the assist nodes) can be used as beacons. In a distributed case, when node B does not send requests to nodes A and C (nodes A and C are not activated), the algorithm becomes a one-beacon algorithm. To emphasize the advantage of using more actuators, Chapter 7 compares the three-beacon and one-beacon versions of the algorithm.

4.2 Multi-UGV, Multi-Destination Case

4.2.1 Leader Election Algorithm

The first task that must be handled is the identification of each individual destination when there are many targets. Since multiple sensor nodes can detect a single event simultaneously, a destination might be actually composed of a cluster of nodes, all of which detect the same event. What needs to be accomplished is assigning a unique identifier to this destination task. This process is equivalent to assigning an identifier to the cluster of nodes. In this case, it is assumed that every node has a unique (ordinal) identifier, and thus the highest identifier among the cluster of nodes can simply be chosen. This is done by using a modification of standard leader election algorithms. That is, the cluster of nodes votes on a particular node,

the highest identifier, as the leader. The initiation and execution of this algorithm depends on a few fundamental assumptions. First, it is assumed that the cluster of nodes associated with a particular destination form a connected subgraph in the communication network; that is, they can communicate among themselves. If this is not the case, the only drawback is that a destination might be assigned with multiple identifications. Second, it is assumed that the destinations are well separated; that is, two clusters of nodes cannot communicate directly with each other. Otherwise, multiple destinations might be treated as a single destination. Third, it is assumed that once a node in a cluster detects an event, the other nodes in its cluster will detect the event at relatively the same time. Since the system is asynchronous, there will certainly be differences in timing, but a simple delay in processing the initial passing of messages can accommodate this difference. The leader election algorithm begins when a sensor event is triggered. The creation of a sensor event is application specific but could be triggered when a node senses a dangerous chemical reading or a high temperature or, as in the initial application, when a node determines that it is on the boundary of a coverage hole in the sensing area.

Because power consumption is important in WSNs, the main issue in this algorithm's performance is not particularly processing time but the communication or *message complexity*. When message complexity is counted by the number of pairwise transmissions, Burns proved [12] an $\Omega(m + n \log n)$ bound on the message complexity where m and n are the total number of links and nodes in the network, respectively.

A straight-forward approach to identifying a leader would be for every node to transmit its *id* to the cluster. This would be accomplished by having every node

transmit a message containing its *id* to its neighbors who would then relay that information on to their neighbors and so on. By remembering the messages sent to prevent retransmission, the process would terminate with every node receiving the identifiers of all nodes in their cluster. By selecting the largest *id* amongst the list, each node would agree upon their cluster's identification. However, a careful analysis of this approach will show that, if n_1 nodes in this cluster and m_1 edges in their communication subgraph, $\Theta(n_1 m_1)$ messages are transmitted. If the size of the cluster is small relative to the overall size of the sensor network, this might not be a significant problem; however, since power consumption is critical, it is preferable to have a more efficient solution that still remains relatively simple.

The timer-based leader election algorithm builds off of this approach but instead of transmitting its identification to the entire cluster, it sends the *id* only to nodes that are at most ℓ hops away, which is initially set to one. In successive passes, as long as the node has not seen a higher identifier, it doubles the distance and retransmits. Once a node has received a higher identifier, the node stops broadcasting its own identifier and simply acts as a relay. Each message transmitted contains two critical components, the potential leader's *id* and the message lifespan, ℓ_{msg} . Each node keeps track of the current best leader, the highest *id* known so far, which initially is just that node's *id*. When a new message is received, if the *id* is larger than the current best known, the receiving node updates the maximum value, and if the lifespan is larger than one, it retransmits the new identifier, with a decreased lifespan. As a further refinement, if the *id* matches the current known highest *and* the remaining lifespan is larger than previously, the message is retransmitted. Each pass lasts a certain time

T , discussed shortly. After the allotted time, triggered by an alarm event, the node retransmits if it still has the highest id seen by it so far, as shown in Algorithm 6.

Algorithm 6 Timer-based Leader Election Algorithm

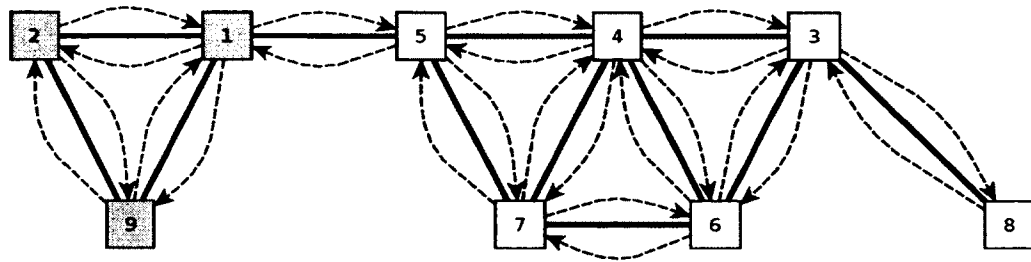
```

1: procedure PROCESSSENSOREVENT
2:   ▷ Initialize global variables
3:    $leaderFlag \leftarrow \text{true}$                                 ▷ Node can participate in leader messages
4:    $id_{\max} \leftarrow id$                                     ▷ Current max is node's id
5:    $\ell_{\max} \leftarrow \infty$                                 ▷ No relay of this value
6:    $d_{\max} \leftarrow$  estimated diameter of cluster          ▷ Application specific value
7:    $\ell_{msg} \leftarrow 1$                                     ▷ Initial message lifespan
8:    $T_w \leftarrow T_p + T_t$                                 ▷ Wait time between retransmissions
9:    $id_\ell \leftarrow -1$                                     ▷ Stores official leader, once elected
10:  ▷ Start transmitting after slight delay
11:  POSTALARMEVENT( $leaderAlarmEvent$ ,  $initialDelay$ )
12: end procedure
13:
14: procedure PROCESSLEADERALARM
15:   if  $id_{\max} = id$  then
16:     if  $\ell_{msg} \geq 2d_{\max}$  then
17:        $id_\ell \leftarrow id$                                 ▷ This node is the leader
18:       BROADCAST( $leaderElected$ ,  $id_\ell$ )
19:       POSTEVENT( $activateHopCountEvent$ )
20:     else                                                ▷ Still the potential leader
21:       BROADCAST( $leaderElection$ ,  $(id, \ell_{msg})$ )
22:       POSTALARMEVENT( $leaderAlarmEvent$ ,  $T_w$ )
23:        $\ell_{msg} \leftarrow 2\ell_{msg}$                         ▷ Increase message lifespan and wait time
24:        $T_w \leftarrow 2T_w$ 
25:     end if
26:   end if
27: end procedure
28:
29: procedure HANDLELEADERELECTIONMESSAGE( $id_r$ ,  $\ell_r$ )
30:   if  $leaderFlag = \text{false}$  then
31:     return                                              ▷ Not part of the leader algorithm, ignore message
32:   else if  $id_r > id_{\max}$  or  $(id_r = id_{\max} \text{ and } \ell_r > \ell_{\max})$  then
33:     ▷ A new larger id or longer transmission range for max id
34:      $id_{\max} \leftarrow id_r$ 
35:      $\ell_{\max} \leftarrow \ell_r$ 
36:     BROADCAST( $id_{\max}$ ,  $\ell_r - 1$ )                        ▷ Retransmit with shorter lifespan
37:   end if
38: end procedure
39:
40: procedure HANDLELEADERELECTEDMESSAGE( $id$ )
41:   if  $id_\ell = -1$  then                                ▷ First notification of a leader.
42:      $id_\ell \leftarrow id$ 
43:     BROADCAST( $leaderElected$ ,  $id_\ell$ )
44:     POSTEVENT( $activateHopCountEvent$ )
45:   end if
46: end procedure

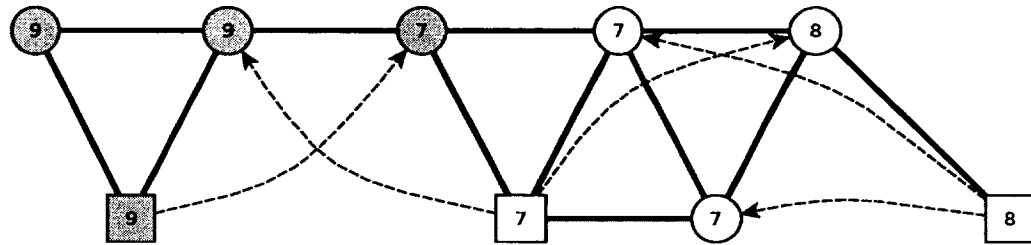
```

The process terminates once time has passed for a message length that is roughly the diameter of the subgraph. This is certainly not more than n_1 , the number of nodes in the cluster, but based on the application one could find a tighter estimate. Once the process terminates, the sole leader will broadcast a final message to the cluster to commence the next phase of the algorithm: determining hop distances in the entire network.

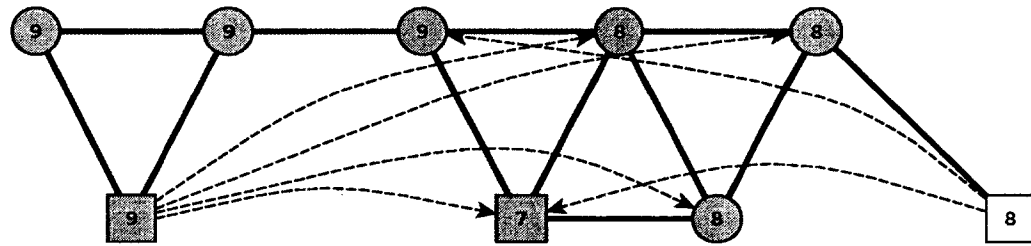
When transmitting a message of distance ℓ hops, it is important that the node waits sufficiently long for the message to propagate through the network. Though it does not cause errors in leader election, a shorter wait time can increase the chances of more messages being transmitted. This time delay can be computed using $\ell(T_p + T_t)$, where T_p is the time to process a message and T_t is the time to transmit a message. This can easily be adapted to include the time to retransmit in case of errors in communication. Algorithm 6 presents the algorithm in its entirety. Note, procedure `PROCESSSENSOREVENT` is the procedure initially triggered by a sensing event, procedure `PROCESSLEADERALARM` is the procedure triggered after every delay, which retransmits the *id* at progressively longer hop counts, assuming the *id* is still the maximum seen by that node, and procedures `HANDLELEADERELECTIONMESSAGE` and `HANDLELEADERELECTEDMESSAGE` handle the passing of leader messages throughout the cluster. Figure 4.7 illustrates one example of the algorithm. Before proceeding further, it is important to show that the algorithm does determine a leader for each cluster.



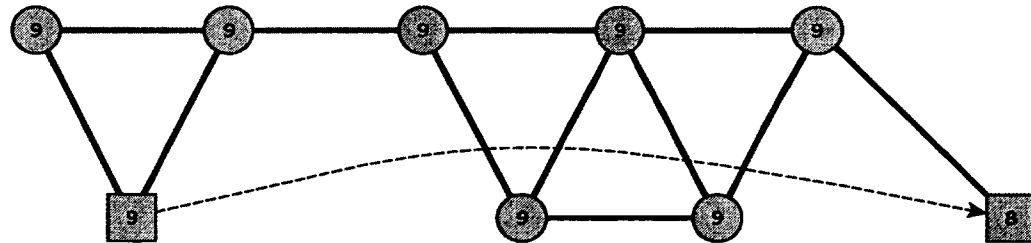
(a)



(b)



(c)



(d)

Figure 4.7: Successive iterations of the Level Assignment Algorithm. Active nodes, shown as squares marked with their ID, broadcast their ids for maximum distances of (a) one (b) two (c) four and (d) eight hops. Relay nodes, shown as circles marked with the maximum ID received prior to the iteration, only retransmit received messages with larger IDs. The shaded nodes represent those nodes covered by the eventual leader for that iteration. Solid arcs represent the communication graph. A dashed arc from node a to node b indicates that node a was able to transmit its id to the given node within the proper hop distance. Observe that during the final iteration of this example only two nodes remain active and the leader gets selected after the iteration completes.

Lemma 4.2. *When the maximum waiting time $T_w \geq d_{\max}(T_p + T_t)$, any cluster of nodes will have exactly one elected leader after following Algorithm 6.*

Proof. Since each cluster is sufficiently close to communicate among themselves but sufficiently separated to not communicate with nodes in any other clusters, each cluster can be considered as a distinct connected graph. Based on the value of d_{\max} , any message of lifespan $\ell_{msg} \geq d_{\max}$ can be received by all other nodes in the cluster. Since $T_w = \ell_{msg}(T_p + T_t)$, the message has sufficient time to reach the furthest nodes. Since each id is unique, this implies that the highest node has sufficient time to transmit its identifier to all other nodes in the cluster. Consequently, since all other nodes in the cluster will at some point have received this identifier, these nodes will no longer consider themselves the leader. \square

4.2.2 Level Number Assignment Algorithm

In the network, each node calculates and maintains its hop distance to all destination clusters, or tasks. The UGVs navigate through the network towards a destination cluster by progressing from one node to a closer node. To calculate these distances, the previous single UGV single destination Level Assignment Algorithm is modified. It is proposed that each node store a local map map_i , where the tasks' ids are set as the map's keys and the hop distances are set as the map's values. For simplicity, it is assumed that the map returns an infinite distance for ids not currently in the data structure.

Algorithm 7 describes the modified level assignment algorithm, which is triggered when the initial cluster nodes determine a winner from the leader election

and invoke the procedure `PROCESSACTIVATEHOPCOUNT`. The other nodes passively process and retransmit level numbers as they are received from other nodes. Since the system is event-based, there is no need to terminate the process, the messages progress through the network until all nodes have determined their hop distances to each task.

Algorithm 7 Distributed Multi-Destination Level Number Assignment Algorithm

```

1: procedure PROCESSACTIVATEHOPCOUNT
2:    $map_t.PUT(id_\ell, 0)$  ▷ The  $map_t$  is initially empty
3:   BROADCAST(levelAssignment, ( $id_\ell, 1$ ))
4: end procedure
5:
6: procedure HANDLELEVELASSIGNMENTMESSAGE( $id_r, \ell_r$ )
7:   if  $map_t.GET(id_r) > \ell_r$  then
8:     ▷ Discovered a distance for  $id_r$  closer than previously known
9:      $map_t.PUT(id_r, \ell_r)$ 
10:    BROADCAST(levelAssignment, ( $id_r, \ell_r + 1$ ))
11:   end if
12: end procedure

```

4.2.3 Task Allocation Algorithm

The UGVs passively wait for neighboring sensor nodes to determine hop distances to known task clusters. Once these values have been determined, the multiple UGVs must negotiate to determine which task each should tackle. The WSAN is used to store information about the tasks: each node stores an additional local hash map map_c that stores the information on the claiming status of tasks. While the tasks' *ids* are set as the map's keys, the values of the map are arrays of length two: the *ids* of the assigned UGV in the first position and the hop distance between the UGV and the task in the second position. Each node's map_c is dynamic, updating its contents based on new incoming messages. Meanwhile, the UGVs also store a copy

of map_c of the same structure constructed during the process. Initially, map_c stores the known distances to each task from the local UGV itself and is updated as new message arrive. The pseudocode is shown in Algorithm 8.

Algorithm 8 WSAN-Aided Greedy Task Allocation Algorithm

```

1: {Code for the UGVs}
2: variables:  $id_u$ ,  $T_s$ ,  $d_h$  and local map  $map_c = NULL$ 
3: if receive a level number  $h$  from task  $T_s$  then
4:   if  $map_c.get(T_s) == NULL$  then
5:      $map_c.add(T_s, \{id_u, h\})$ 
6:   end if
7: end if
8: find the task  $T_s$  with shortest distance in  $map_c$ 
9: moving toward  $T_s$  and broadcast  $(T_s, \{id_u, h\})$  to the network
10: while receive a message on task  $T_s$  do
11:   if local distance is less than received distance then
12:     update  $map_c$  with the received information
13:   end if
14: end while
15: if receive a rejection message then
16:   find an alternate task  $T_s$  not claimed by other UGVs in  $map_c$ 
17:   move toward  $T_s$  and broadcast  $(T_s, \{id_u, h\})$  to the network
18: end if
19:
20: {Code for the nodes}
21: variables:  $id_r$  and local map  $map_t = NULL$ 
22: while receive a claim message regarding  $T_s$  do
23:   if  $map_t.get(T_s) == NULL$  then
24:     add the received information to  $map_c$ 
25:     broadcast this received information to neighbors
26:   else if stored distance is larger than received distance then
27:     update  $map_c$  with the received information
28:     broadcast this received information to neighbors
29:   else if stored distance equals received distance then
30:     forward tied info to lower level neighbors regarding  $T_s$ 
31:   end if
32: end while
33: if receive a tied information then
34:   if is the leader then
35:     pick one UGV and send a rejection message backward
36:   else
37:     forward the tie information
38:   end if
39: end if

```

The algorithm starts when a UGV receives a message that includes the tasks ids and the hop distances from the tasks. Then the UGV will wait a predefined time period to construct its own local hash map map_c . After that, it will choose a task that is the shortest distance away and claim this to the whole network by broadcasting the pair (T_s, d_h) where T_s is the identification of the chosen task and d_h the distance to this task cluster. In response, nodes that receive this claim will check their local memory to see whether there is another appropriate UGV that has already taken the task. When there is a conflict, such as two UGVs ugv_1 and ugv_2 claiming the same task T_1 and their distances to T_1 are equal, the node will forward the information to the leader node of T_1 . The leader will select one UGV by sending a reject message to the other UGVs. For example, if the leader node takes ugv_1 to fulfill the task, then ugv_2 will receive a rejection and will need to claim another available task.

4.2.4 Special Case: Traveling Salesman Problem

The problem presented here is similar to the Traveling Salesman Problem, which is a classic NP-complete problem [32]. Though there exists many heuristic methods for solving the problem, global information is required to optimize the solution, and that is not always available in distributed systems.

The Nearest Neighbor Algorithm is usually applied when only local information can be obtained. However, the simple nearest neighbor algorithm might result in moving back and forth between certain points. A typical situation is shown in Figure 4.8, where a UGV is located at point o , and there are six tasks located at points a, a', b, b', c , and c' . Distances between nodes are shown in the figure.

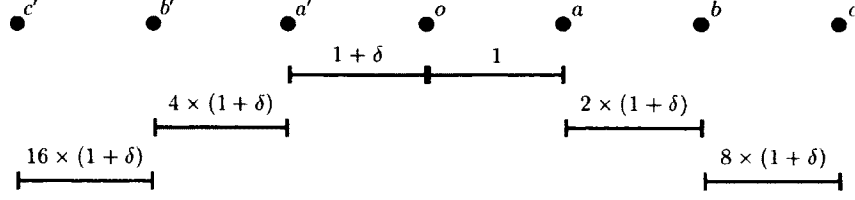


Figure 4.8: A critical case for the nearest neighbor algorithm in MTSU. The distances between nodes are labeled underneath, where δ is a very small positive real value.

For the simple nearest neighbor algorithm, the sequence to visit all tasks is $a \rightarrow a' \rightarrow b \rightarrow b' \rightarrow c \rightarrow c'$. In the systems where the UGV cannot be considered as a point mass, these sharp turns can be a waste of energy and time. To evaluate the performance and take the angle change into the account (besides distance), the following cost function is formulated:

$$J = P \cdot \sum_i d_{\psi_i \psi_{i+1}} + Q \cdot \sum_i \phi_{\psi_i \psi_{i+1}}, \quad (4.1)$$

where all visited tasks are stored in sequence in ψ , P is the weighting factor corresponding to the traveling distance, $d_{\psi_i \psi_{i+1}}$ is the distance from destination ψ_i to destination ψ_{i+1} , Q is the weighting factor corresponding to the turning angles, and $\phi_{\psi_i \psi_{i+1}}$ is the angle change from destination ψ_i to destination ψ_{i+1} .

Intuitively, to avoid moving back and forth (large turning angles), a UGV should first finish servicing destinations in the same direction before returning to service other destinations. A WSA-Aided Nearest Neighbor Algorithm (Algorithm 9) where the UGV maintains a distance map which stores all the level numbers of last visited destination, which denotes the distances between the UGV and the other destinations. After arriving at a new destination, the UGV updates the distance map.

By comparing the updated distances to the former ones, the UGV is aware of which destinations are getting closer. This way, the UGV does not need the distance map since every node, including the destinations, can get hop level numbers to every other destinations after the level assignment algorithm. The UGV only needs to request the level number map of the current destination.

Algorithm 9 WSAN-Aided Nearest Neighbor Algorithm

```

1: {Code for the UGVs}
2: Initialize:
3:  $T_c = -1$ ,  $map_t = NULL$ ,  $map_{tem} = NULL$ 
4: while listening do
5:   if receive a level number  $h$  from task  $T_s$  then
6:     if  $map_t.get(T_s) == NULL \parallel map_t.get(T_s) > h$  then
7:        $map_t.add(T_s, h)$ 
8:     end if
9:   end if
10: end while ▷ initializing listening mode
11:  $map_{tem} = map_t$ 
12: while  $map_t.size() > 0$  do
13:   if  $T_c == -1$  then
14:     find the  $T_s$  which is with the lowest hop distance in  $map_{tem}$ ,  $T_c = T_s$ 
15:   else if reached  $T_s$  then
16:      $map_t.remove(T_s)$ 
17:     listen for existing tasks, and build up  $map_{tem}$  with existing tasks and corresponding hop
    distances
18:     for iterator  $i$  of  $map_{tem}$  do
19:       if  $map_t.get(i) == NULL$  then
20:         continue
21:       else if  $map_{tem}.get(i) > map_t.get(i)$  then
22:          $map_{tem}.remove(i)$ 
23:       end if
24:     end for
25:     if  $map_{tem}.size() > 0$  then
26:       find the  $T_s$  which is with the lowest hop distance in  $map_{tem}$ ,  $T_c = T_s$ 
27:        $map_t = map_{tem}$ 
28:     else
29:       find the  $T_s$  which is with the lowest hop distance in  $map_t$ ,  $T_c = T_s$ 
30:     end if
31:   else
32:     keep moving to current task  $T_c$ 
33:   end if
34: end while ▷ moving mode
35:
36: {Code for the nodes}
37: nodes will send their local map  $(T_1, h_1), (T_2, h_2) \dots (T_m, h_m)$  to UGV upon request
  
```

Algorithm 9 is triggered on when the UGV receives a message containing a level number, which means there is a task coming out. Then, the UGV will listen for incoming messages for a predefined period of time (determined by the application) and will construct the initial distance map_t , in which tasks' *ids* are set as the map's keys and the hop distances are set as the map's values. After the time elapses, the UGV will simply take the nearest destination as the first destination. When arriving at the first destination, the UGV will request a new distance map map_{tem} from the current destination and compare map_t to map_{tem} . The destinations with shorter distances will be regarded as the destinations in the same main direction. Then, the next destination should be the closest one in the same main direction. Every time arriving at a new target, the UGV will take the newer map map_{tem} to replace map_t .

4.3 Hole Patching Algorithm

A deterministic hole patching algorithm in a coordinate-free network is presented. The only information available is the coverage hole boundary nodes or coverage hole edges. Since Assumption 8 (in Chapter 3) specifies that the coverage holes are far away from each other, one side of each coverage hole edge should be the hole area, while the other side should be the area that is already covered. A check up process is necessary since there is no coordinate or other information available, and one cannot tell which side of the edge is a hole. This check up process is described later in this section.

In general, new nodes are added around each edge of a coverage hole. To each hole edge, a new node will be deployed along the perpendicular bisector of the

edge. Sensor's communication range r_c and sensing range r_s are dependent since it is not desired to introduce new coverage holes during the patching process. As discussed in [10], there are two options: if $r_s \geq r_c/\sqrt{3}$, as long as the new node can connect to both of the two nodes, there will be no new hole introduced; else a more precise technique is needed to measure distances between nodes to avoid introducing new holes. Figure 4.9 shows the cases where three nodes are pairwise connected at length of r_c . As can be seen, when $r_s \geq r_c/\sqrt{3}$, no hole will be added once the three nodes are connected by r_c . In the case when $r_s < r_c/\sqrt{3}$ and all communications are preserved, a new node should not go further than $\sqrt{3} \cdot r_s$. For simplicity, it is assumed that $r_s \geq r_c/\sqrt{3}$ in this dissertation.

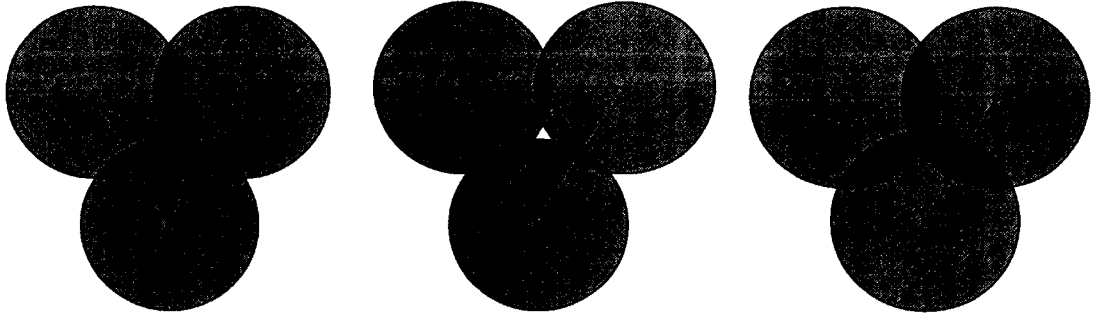


Figure 4.9: Area coverage where nodes are connected at the length of r_c . Left: no coverage hole when $r_s = r_c/\sqrt{3}$; middle: a coverage hole exists in the middle when $r_s < r_c/\sqrt{3}$; right: no coverage hole when $r_s > r_c/\sqrt{3}$.

At least one side of the edge is not covered, which is the hole that needs to be patched. It is proposed to run a hole detecting algorithm to determine which side the new node should be deployed. As stated in Algorithm 10, on each edge, one side will be tried first and it will be seen if the new node will be a new hole boundary node. If yes, deployment at this side is valid. Otherwise, there are two situations that need to

be checked. First, if there is no hole existing any more, deployment of a node at this side is still valid and the patching is done. If not, it means the new node has dipped into the area that is covered already, and deployment should go to the other side of the edge along the perpendicular bisector.

Algorithm 10 Centralized Hole Patching Algorithm

```

1: for coverage hole  $i$  do
2:   pushing all the edges of this hole to a stack  $S_{hi}$  (by clockwise or anti-clockwise order)
3:   while this hole still exists do
4:     moving UGV to the next edge (denote the two ends as  $a$  and  $b$  for description) in  $S_{hi}$ 
5:     moving UGV to any one side of edge  $ab$  along its perpendicular bisector
6:     at a position  $p$  where  $ap \leq r_c$  and  $bp \leq r_c$ , activate a new node  $c$  on the UGV
7:     run a hole detection algorithm
8:     if this hole is NOT changed then
9:       moving UGV to the opposite side of edge  $ab$  along  $ab$ 's perpendicular bisector
10:    else if no hole exists then
11:      this hole is patched, break
12:    else
13:      removing the edges that disappear in current hole, pushing new edges to  $S_{hi}$ 
14:      go to Line 4
15:    end if
16:  end while
17: end for

```

Figure 4.10 shows the hole patching process. Only the area inside the boundary is considered as the coverage hole. New nodes are deployed around the hole. If the hole still exists after a round, a new round will start from the newly identified hole, as demonstrate in the lower figure of Figure 4.10. The simulation testbed will be introduced in next chapter, which covers design, introduction of functions, and some examples as well.

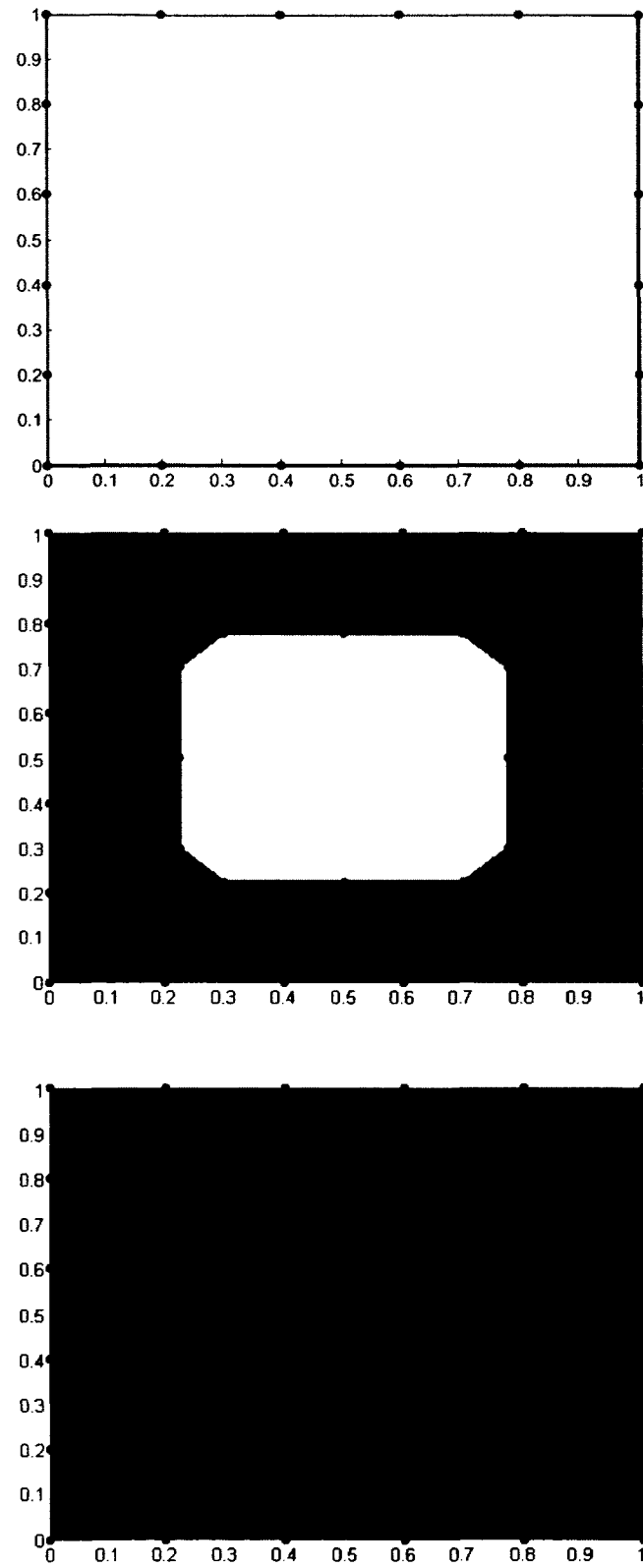


Figure 4.10: Hole patching process. Upper: an identified coverage hole; middle: new nodes are added along the edges of former coverage hole; lower: the hole is fully patched.

CHAPTER 5

WIRELESS SENSOR NETWORK SIMULATOR

A simulation testbed has been deployed in Java which is capable of simulating both centralized and decentralized sensor and actuator network algorithms. In the centralized algorithm simulation, actions of nodes and UGVs are managed directly by a central controller. In contrast, the distributed algorithms run in an event-driven manner and are built using multi-threads, where each individual node is designated as a thread. The node is active only when the corresponding thread is running. Although no communication delay models are considered in this dissertation, the testbed simulates asynchronous working patterns based on the properties of multi-threading. Messages might not be received in the proper sequence since the running sequence of the threads is not enforced, which inherently simulates the communication delays to some extent. The generated data are saved to text files, which can be read and analyzed using MATLAB or other software of the user's choice.

5.1 Architecture Design

The architecture of the testbed is shown in Figure 5.1, which has four main layers. Some existing components are provided in the figure. The top layer is the simulator layer. Various types of simulators can be built regarding different applications.

For example, a GUI application was developed which is able to intuitively demonstrate running algorithms. Examples of the GUI are shown in Section 5.3.

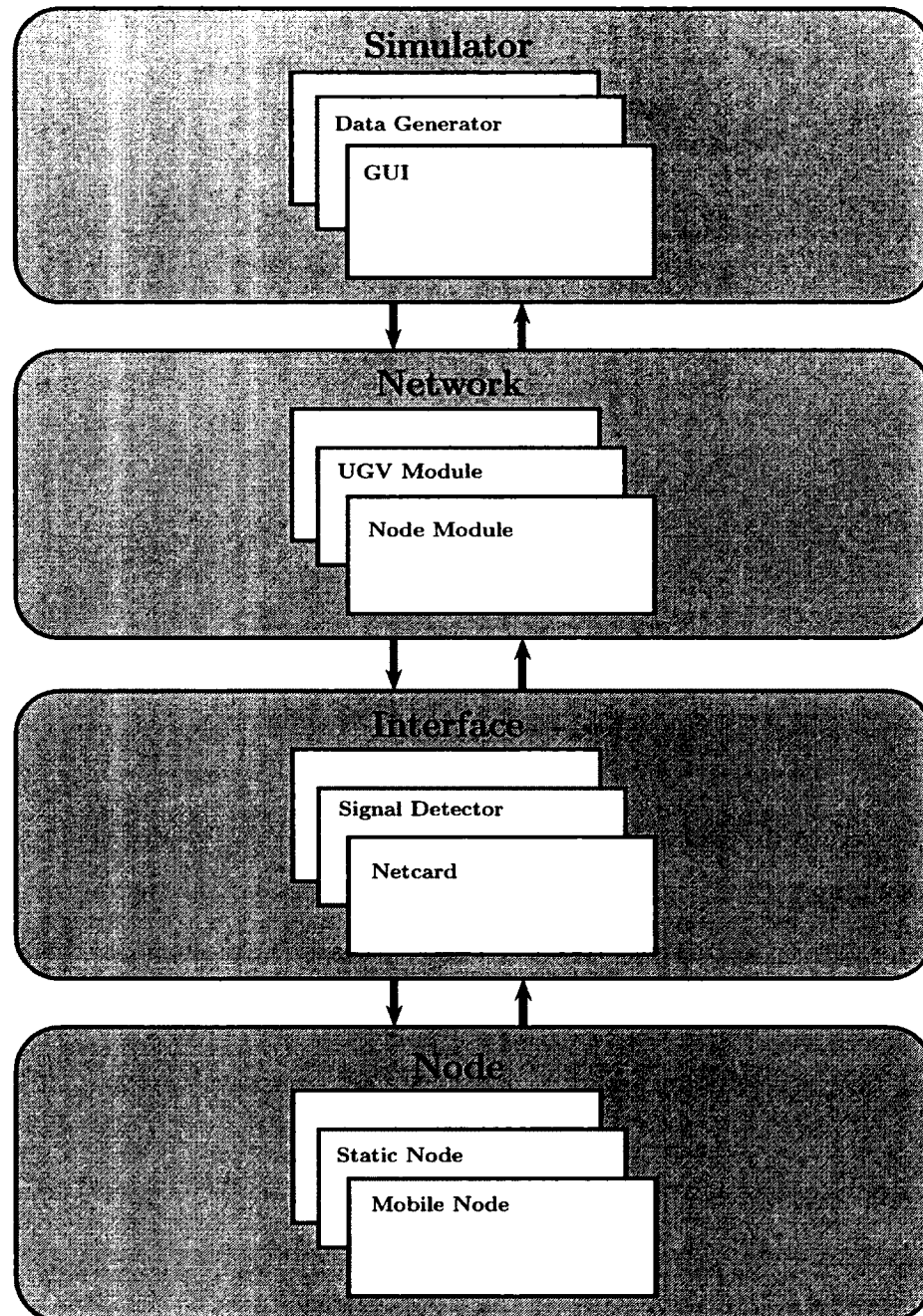


Figure 5.1: Architecture of the simulation testbed.

There is also another simulator which generates and writes large sets of data to text files that MATLAB can use for analysis. Simulation data of Chapter 7 are all generated by this type of simulators.

The Node layer represents the lowest level in the testbed. It includes different types of nodes (classes) that could exist in the network, like static nodes and mobile nodes (UGVs). Specific functions are defined according to certain type of nodes. For example, in distributed algorithms, an event-driven manner is strictly followed. That means that the node's reactions are triggered only by the received events. Also, the status of all the memory units (variables, arrays, etc.) inside one particular node object should not be directly altered by any other node objects. According to the official Java tutorial [59], it is said that an object is created after the corresponding class is instantiated. Therefore, an Interface layer is required to allow the node objects to interact with each other. During the simulation, this interaction is realized by accessing networking data in the Network layer through the Interface layer. Thus, the Interface layer is actually the medium between the Node layer and the Network layer. For example, the *Netcard* class is in charge of message transmitting, which requires access to communication graph stored in the Network layer. The *Signal Detector* (*SigDetector* is the real name) class is in charge of detecting signal strength, which requires coordinates to calculate distance in the simulation.

The Network layer is the second layer which is underneath the Simulator layer. Networking information, such as the communication graph, is stored in the Network layer. In the meantime, information that has to be kept private from lower layers is also stored in the Network layer, such as the nodes' coordinates, which can be used to

get the network's communication graph as well as the node's signal strength. These coordinates might also be needed to draw the network for demonstration purpose. However, the access to coordinates and the communication graph has to be strictly controlled to ensure they will not be used directly by the lower layer *Node* class or *UGV* class.

The basic unit stored in the network layer is called a module. Functional *node modules* as well as UGV modules can be constructed when interface objects are mounted to node objects. The *node module* consists of a *Static Node* object and a *Netcard* object; the *UGV module* consists of a *Mobile Node* object, a *Netcard* object and a *Signal Detector* object. A more detailed introduction is provided in the following section.

5.2 Function Development

In this section, the developments of some major components (classes) in each layer are introduced.

5.2.1 Node Layer

The Node layer is the lowest layer in the simulator, which currently contains one *Node* class and one *UGV* class. *Node* class defines functions for the static sensor and actuator node. Some major functions are listed in Table 5.1. A *Netcard* object is mounted to the *Node* class by a function *setNetcard()*. A *Node* combined with a *Netcard* is considered as a *node module*, which can perform all the expected node functions. For each *node module*, *Node* class takes care of data processing while *Netcard* class takes care of message sending and receiving. The *checkToken()* function

is the core function in *node module* since it checks all received events and determines the follow up actions. Except target nodes, which initiate a process (the node which initiates the level assignment process), all other node's actions are determined by checking tokens. Received tokens are added to a buffer queue in order in the *Netcard* object. For instance, *updateLevel()* function will be called if its predefined token is found. Thus, functions like the listed *onAct()* and *returnMSG()* function will be called inside *checkToken()* function regarding corresponding tokens.

Table 5.1: Selected functions in the *Node* class

Function	Description
<i>setNetcard()</i>	connect a <i>Netcard</i> interface to this node
<i>updateLevel()</i>	update the local level number if a received level number plus one is smaller than the local level number; a follow up broadcasting of the updated level number will be initiated if a level number is changed
<i>joinMSG()</i>	broadcast a "join" message and then construct a neighbor list based on the nodes replied
<i>returnMSG()</i>	in response to <i>joinMSG()</i> with a message which includes its ID and level number
<i>onAct()</i>	turn on actuating based on a received message
<i>checkToken()</i>	add all the received messages into a string tokenizer queue; go through all the tokens until the queue is empty
<i>run()</i>	executions for applications
<i>killNode()</i>	terminate executions if the node's tasks completed or exceptions happen
<i>packString()</i>	pack useful information into one string, which can be used for the transmission

Since every *node module* runs as an independent thread, according to Java documentation [58], only functions inside *run* can be executed during a thread's runtime. The main structure of *run()* function is described in Figure 5.2. A *node module* (the thread) keeps running if all the *while* flags are true. These flags can be altered by timers or status changes. Meanwhile, the *checkToken()* function is scanned

in the “while” loop. Normally, the input *token* (or event) is added to the buffer queue and popped from the buffer queue in the order it arrived. Buffer queue is a memory unit constructed in *Netcard* class in the Interface layer.

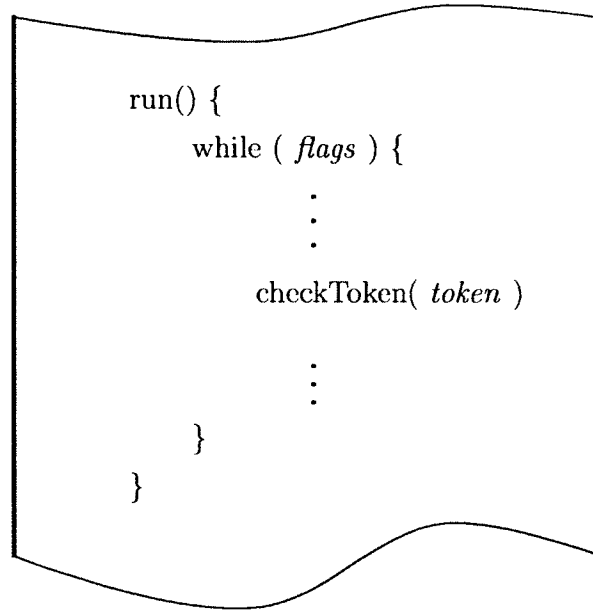


Figure 5.2: Structure of the *run()* function.

UGV class is a class of the *Node* class. Thus, the *UGV* class inherits all the functions from the *Node* class. For example, the *UGV* class can call *Node* class' *joinMSG()* function directly to construct a neighbor list without the need to define a new one. In the meantime, the *UGV* class needs to override some classes like *checkToken()* and *run()* to define its own functionalities. Most importantly, some new functions are added for specific *UGV* functions, some of which are listed in Table 5.2. As stated in Algorithm 5, *UGV module* controls activations of actuators (by function *pickNode()*, *offActu()*, etc) as well as navigation control (by function *getDirection()*, *moveUGV()*, etc).

Table 5.2: Selected functions in the *UGV* class

Function	Description
setSigDetector()	connect a signal detector interface to this node
pickNode()	choose and return the node with lowest level number among all the neighbors
offAct()	turn off active actuator nodes when the UGV arrives at the local minimum point
getDirection()	calculate the next moving direction using received signal strengths based on Eq. 3.3
moveUGV()	move in a predefined step along the calculated direction
killUGV()	terminate executions once the UGV's tasks completed or exceptions happen

Since the UGV class inherits from the Node class, a *Netcard* is automatically mounted. Additionally, a signal detector *SigDetector* is added to give the *UGV module* the ability to detect signal strength. From the UGV's point of view, the potential field is built by accumulating signal strength of actuating signals. The listeners array, which is described in Chapter 3, is constructed inside *UGV* class. Then, the UGV's moving direction can be determined by comparing each listener's potential value, which is calculated by the signal strength. The signal strength can be retrieved by the *Signal Detector* object.

More interfaces can be added to give nodes more functionalities in the future. For example, a sonar array interface can also be built for obstacle avoidance applications.

5.2.2 Interface Layer

The Interface layer is the medium between the Network layer and the Node layer. Thus, classes in the Interface layer have access to classes in the Network layer as well as classes in the Node layer. While some data are not available directly to lower

layer classes, interface classes are used to get these data. For example, a neighbor list will be constructed in the *Netcard* class after network connections are established. Thus, while coordinates or communication graph are unknown for *Node* objects and *UGV* objects in this simulation, data can still be exchanged through interface classes. Currently, there are two classes constructed in the Interface layer: the *Netcard* class and the *Signal Detector* class. While the *Netcard* class takes care of message sending and receiving, the *SigDetector* class is in charge of detecting actuator signals and finally giving directions in navigation.

Major functions of the *Netcard* class are listed in Table 5.3. Two objects, *Network* and *Node*, are connected to this *Netcard* object. A public-access queue is claimed as a receiver's memory buffer in the *Netcard* class. Sending a message can be emulated by adding the message to the queues of the destinations through the *Netcard* object. For example, if node n_1 sends a message "*msg*" to node n_2 , the underlying operation in the simulation is n_1 's *Netcard* writes "*msg*" to n_2 's *Netcard* buffer queue through the *Network* access. For each node, receiving a message is emulated by popping or peeking data from its own *Netcard*'s queue.

Table 5.3: Selected functions in the *Netcard* class

Function	Description
setNetwork()	set the network this <i>netcard</i> belongs to
setNode()	set the node this <i>netcard</i> connects to
broadcastMessage()	broadcast a message to all nodes in communication range
sendMessage()	send a message to specific node(s)
peekMessage()	get the next message from the message queue

Major functions of the *Signal Detector* class are listed in Table 5.4. There are also two objects connected: the *Network* object and the *Node* object. The *Signal Detector* class is able to get signal strengths, which is related to distances as stated in [63]. The Signal Detector calculates distances by accessing coordinates stored in the *Network* class.

Table 5.4: Selected functions in the *SigDetector* class

Function	Description
setNetwork()	set the network this <i>netcard</i> belongs to
setNode()	set the node this signal detector connects to
getSignal()	get the signal strength from one other node
getDirection()	give a direction for the next movement

5.2.3 Network Layer

There is only one class, the *Network* class, built in the Network layer. Some major functions are listed in Table 5.5.

Table 5.5: Selected functions in the *Network* class

Function	Description
createNode()	create a node object
buildNetwork()	add a node and UGV objects to the network
getConnections()	build a communication graph for the network
startNetwork()	turn on all the nodes and UGVs

The network can be considered as a central storage, which instead of issuing control commands, can only provide information. Network layer is also in charge of constructing a network. As a result, all the *node module* objects, *UGV module* objects, are created following the creation of the *Network* object. All the network information

is stored or can be accessed by the *Network* object, like a number of sensor nodes, a number of UGVs, a communication graph, etc.

In the *Network* class, *node module* objects are stored in a hash map, where the node's ID number is the index number, and a new structure called the *NodeStore* is the value. This *NodeStore* is defined as an inner class inside the *Network* class. Figure 5.3 shows the components that are included in the *NodeStore* structure. Once a *NodeStore* object is created, one *Node* object, one *Netcard* object and one *Thread* object will be simultaneously created. In the decentralized simulation, each *node module* runs as an independent thread. Here the *Thread* object is used to generate and start the running of a new thread, or activate a new node in the current case.

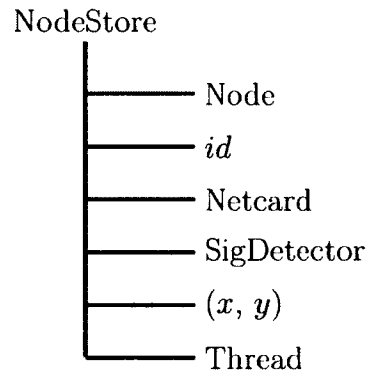


Figure 5.3: Components in *NodeStore*.

5.3 Applications

Applications are built in the highest Simulator layer. Currently, there are two types of applications developed to test the proposed algorithms. One is developed to run certain algorithms repeatedly and generate sets of data for analysis. The other one is a Graphical User Interface (GUI) application. As the startup interface shown

in Figure 5.4, there are three main parts. The first is the control part, which includes buttons, sliders and mouse actions as well; the second part is the network display canvas, which shows the topology of the network; the third is the output of results, which can display debug and statistical results. As shown in Figure 5.5, a network is plotted on the canvas, where the sensor and actuator node is represented by a small dot. The UGV is represented by a polygon, and the destination is represented by a star. When there is a communication connection between two nodes, a straight line is plotted. While new nodes can be added by mouse clicking in the canvas area, all nodes, UGVs and destinations can be moved by mouse pressing and dragging. Figure 5.6 shows an example of multiple level assignment. After deploying a small network by mouse clicking (destinations are predefined), the algorithm runs when the *level* button action is fired. A result of complete single UGV and single destination navigation simulation is shown in Figure 5.7, where active actuators are highlighted with small stars and the UGV moving trace is plotted as well. In next chapter, the hardware used for the experiment is introduced as well as the experiment setup.

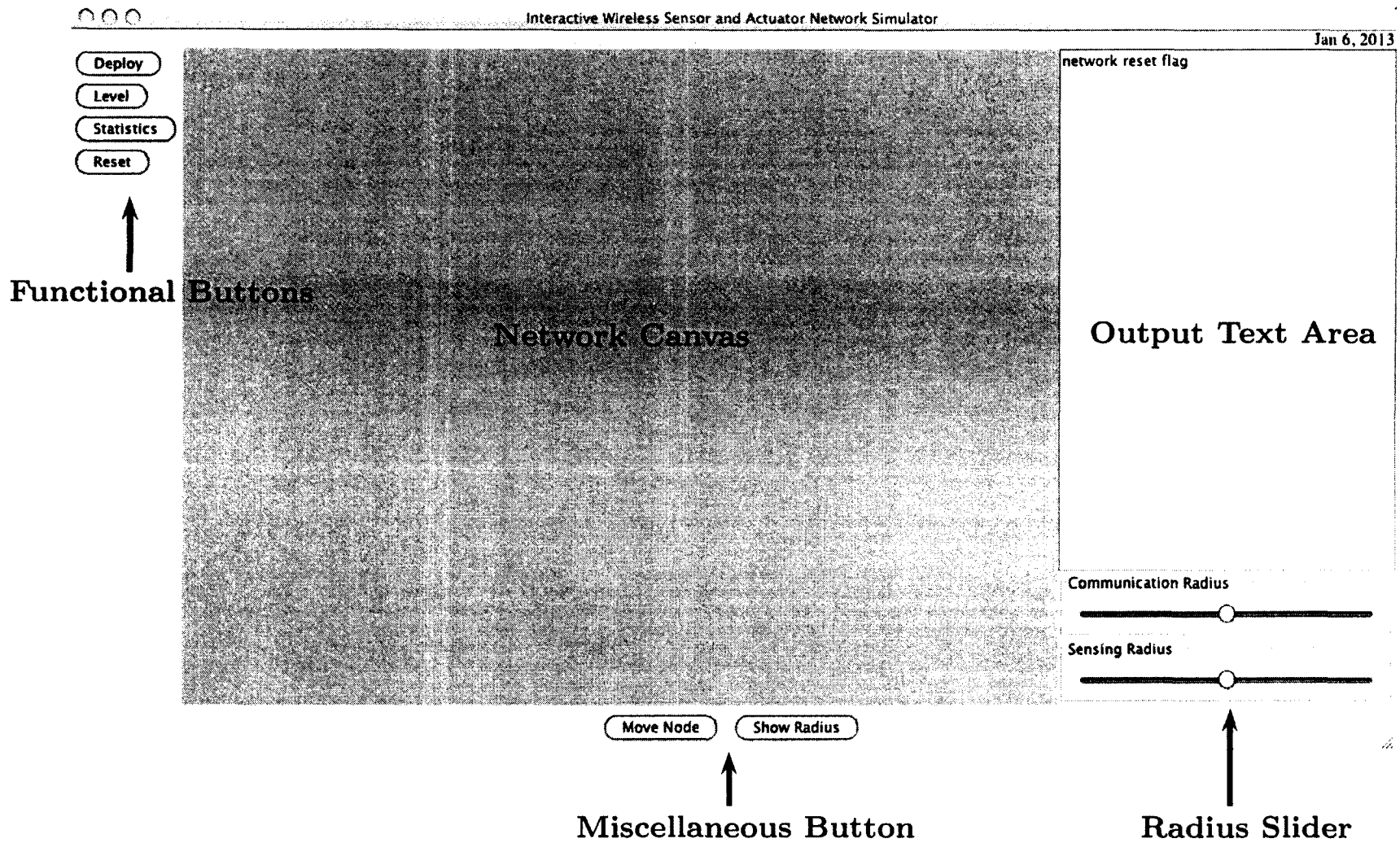


Figure 5.4: GUI Application Interface.

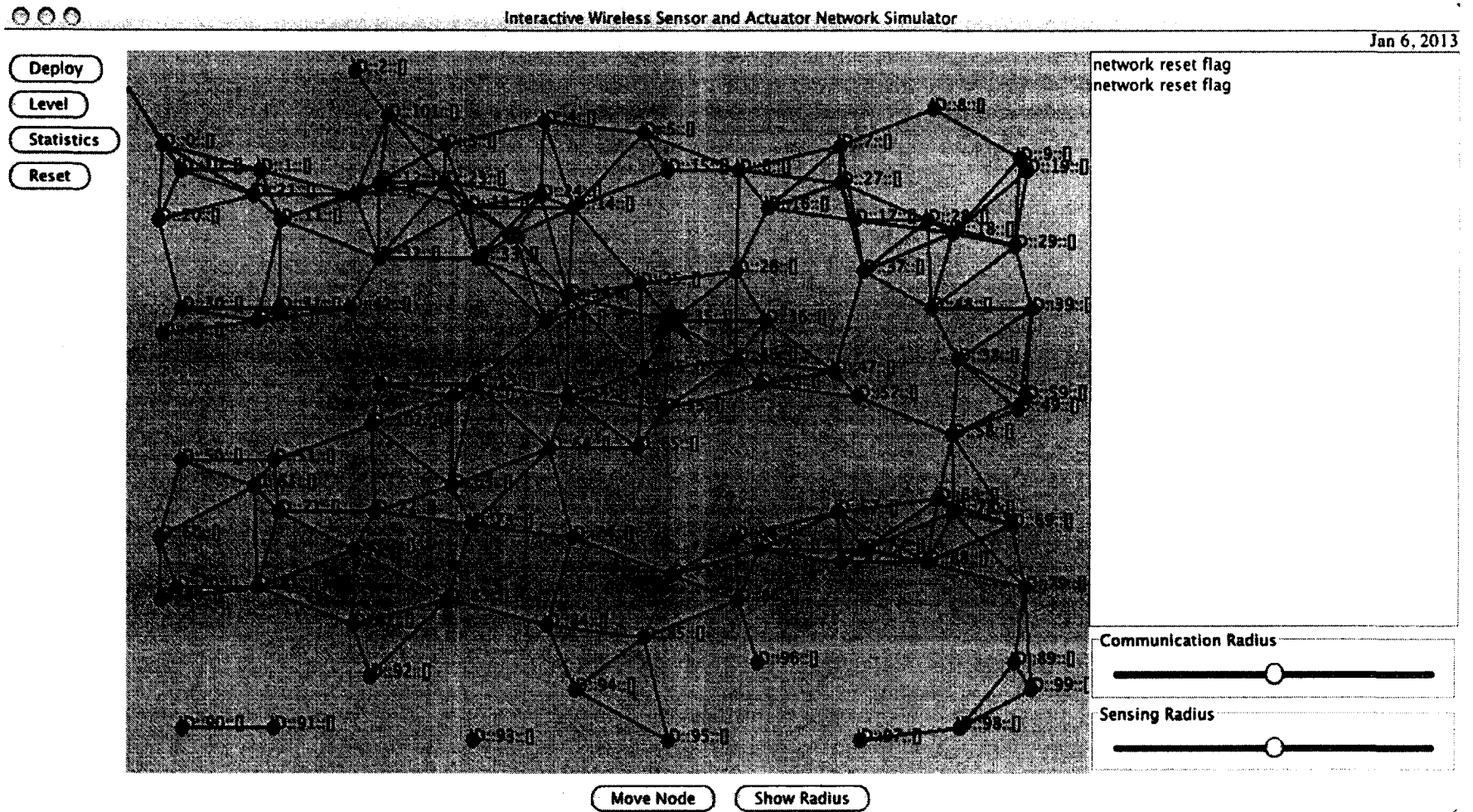


Figure 5.5: GUI with a network deployed.

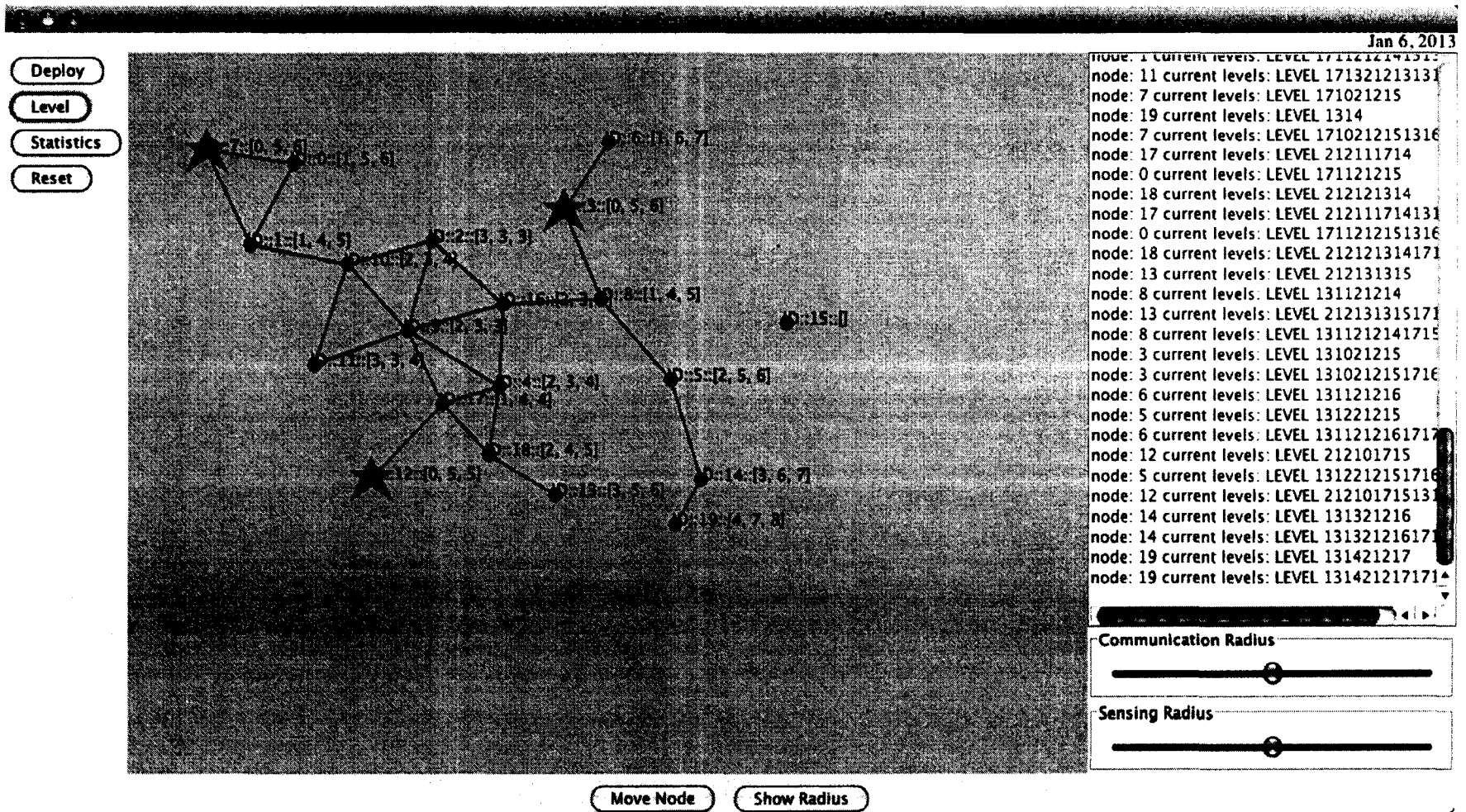


Figure 5.6: GUI shows a network with multiple levels assigned.

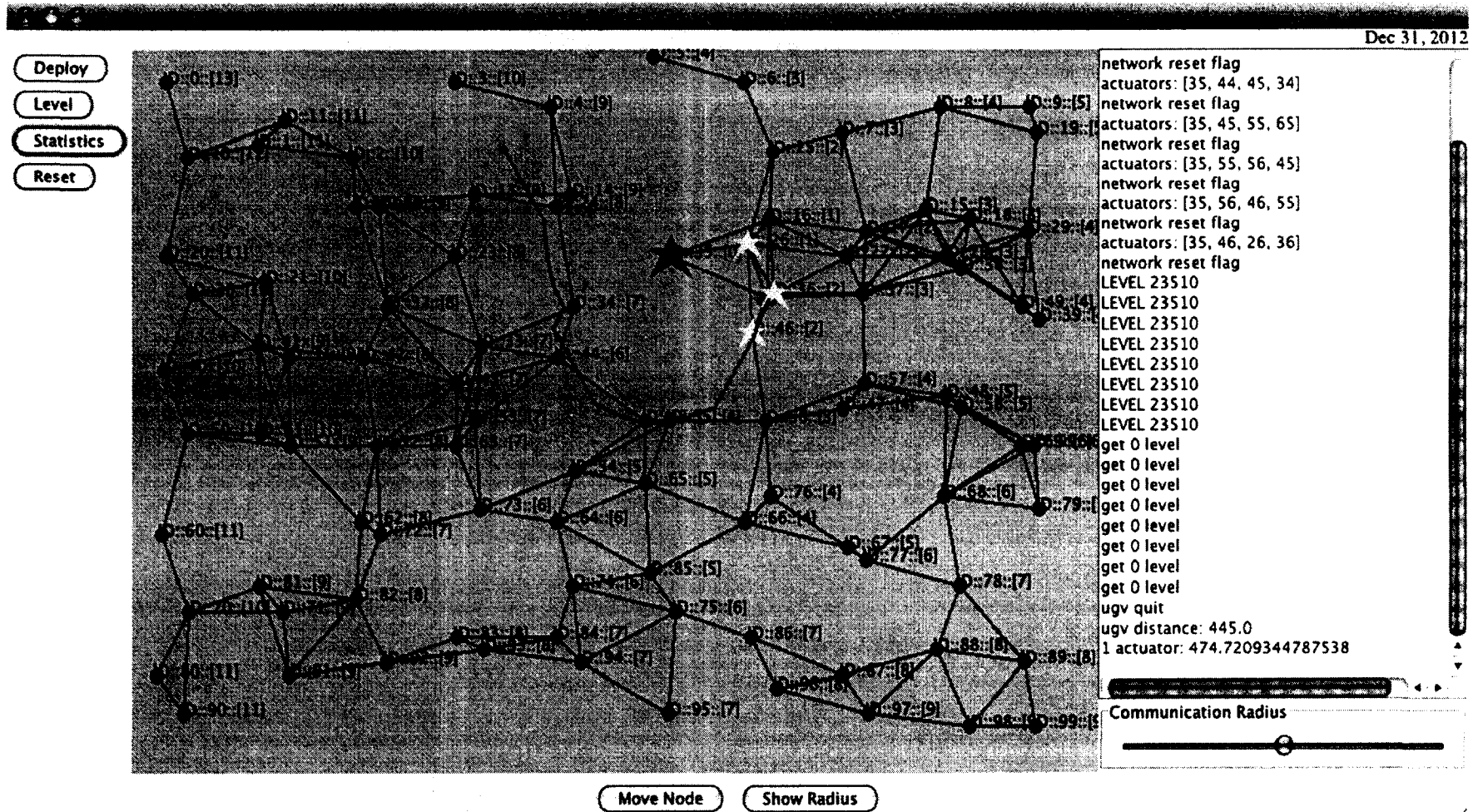


Figure 5.7: GUI Application with results displayed.

CHAPTER 6

HARDWARE TESTBED

6.1 Equipment Design

Currently, only the algorithm that is in the single-UGV, single-destination configuration is realized. The Cricket platform [53] is used for the wireless sensor and actuator network. Beside some basic sensing and communication capabilities, the Cricket platform can estimate the range between nodes by using the combination of RF and ultrasound signals. The technique is based on the time difference of arrival between two simultaneously sent signals such as RF and ultrasound. The precise measurement of the time difference of arrival allows for an accurate calculation of the distance between a pair of sensor nodes. In the experiment, the estimated distances is used to form potential fields directly, instead of the estimates from signal strength as stated in Equation 3.1.

As can be seen in Figure 6.1, the Cricket mote is equipped with one ultrasound transmitter and one ultrasound receiver. To better receive the ultrasound signals, it is suggested that the Cricket motes are positioned face to face. As shown in Figure 6.2, the Cricket nodes are placed on the ceiling with the face down to the ground. The robot is equipped with five Cricket motes (which serve as listener nodes) with the face up to the the ceiling, as shown in Figure 6.3. This setup serves as a hardware

realization of steepest descent algorithm in a given potential field. Given the potential field, the robot, at every step, searches for the minimum of the potential field. A hardware-based solution is proposed, where controller electronically searches for the minimum of the potential field using the on-board listeners.

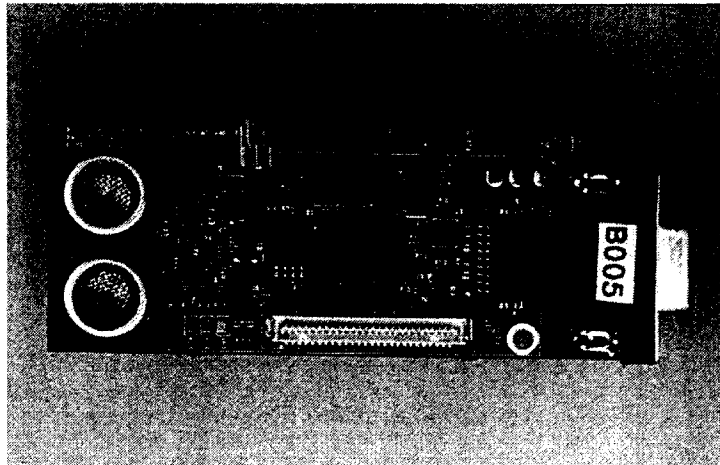


Figure 6.1: The Cricket mote.

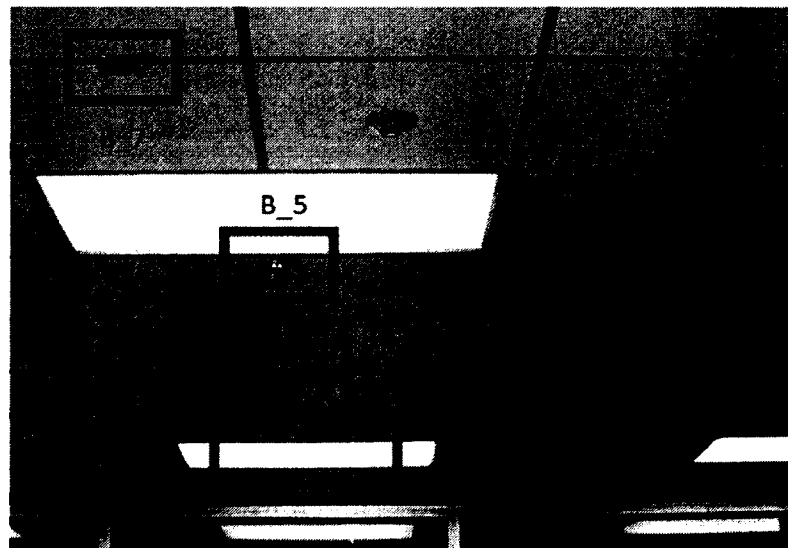


Figure 6.2: Cricket nodes hung on the ceiling.

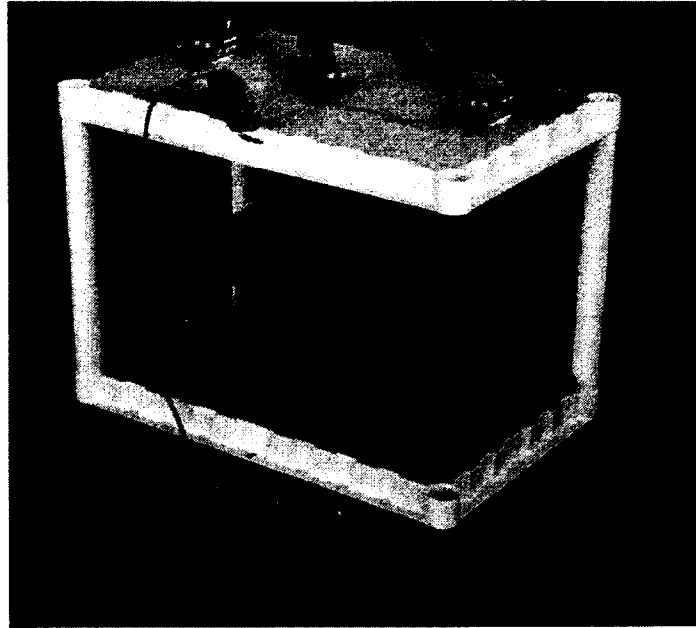


Figure 6.3: Cricket nodes on the UGV.

The robotic device is the Pioneer 3-DX, a standard unmanned ground vehicle. The UGV is controlled by a laptop piggybacked on the UGV. The laptop serves as the controller and can communicate with listeners and send commands to the UGV.

The navigation system consists of four parts: the WSAN (considered as a beacon group), listeners, a laptop serving as a robot controller, and a UGV.

There are three connections inside these four components. As shown in Figure 6.4, *COMM1* is the connection between the laptop (controller) and the UGV. Once the controller determines an updated direction for the UGV, a command is sent to the UGV through the serial port; *COMM2* is the connection between the controller and listeners. The controller grabs and processes the data received by listeners and sends back commands to listeners through the serial port. A connection between listeners and the WSAN is established through *COMM3*. The potential fields are established using wireless communication between beacons and listeners.

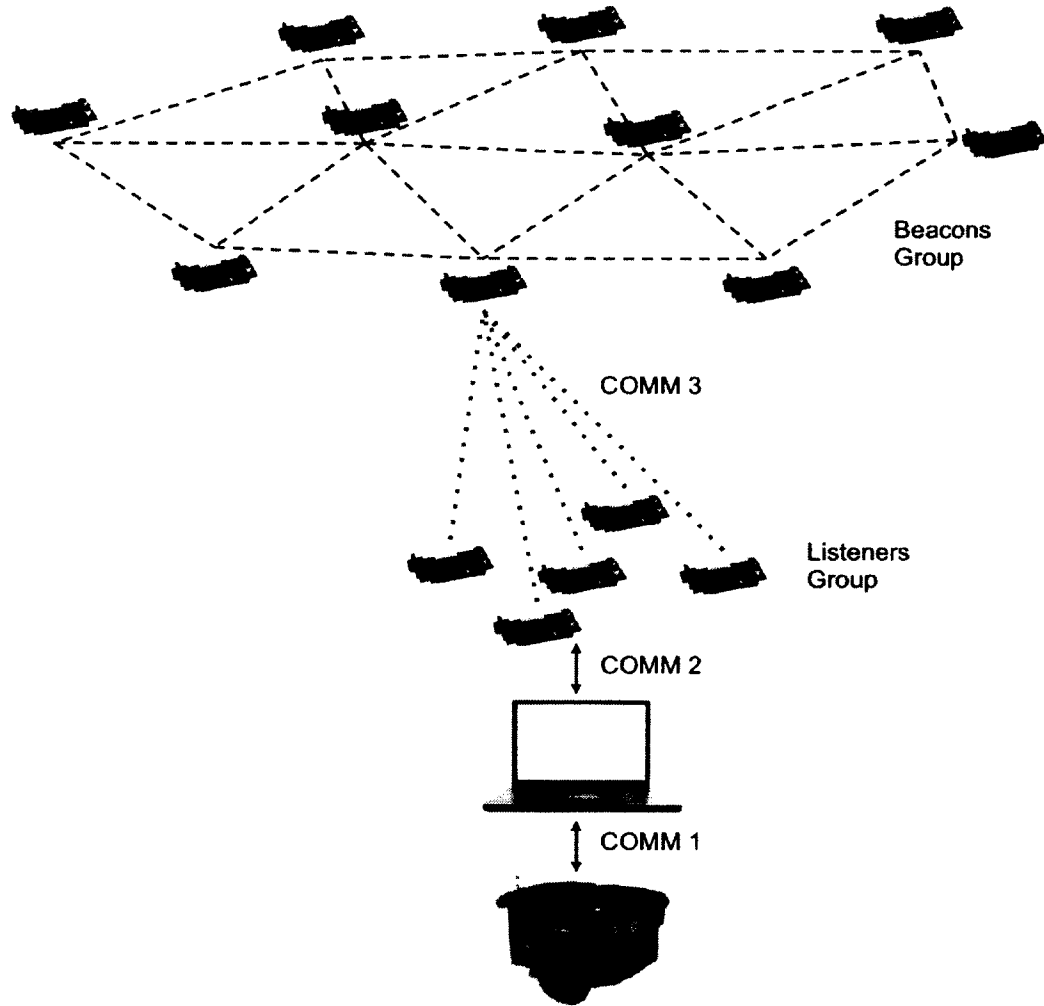


Figure 6.4: System block diagram of the experimental setup.

6.2 Experiment Setup

In the experiment, each sensor node on the ceiling is assigned with a level number, where nodes with the lowest level are assumed to be the destination. Once connected to a higher level number sensor node, the UGV will move from higher level nodes to the node with lowest level (destination). The navigation terminates when the UGV arrives at the last minimum potential point, i.e., when it does not move any further. To avoid oscillating around the local minimum, in this experiment, the moving step of the UGV is set to 15 cm. During the navigation, after every 15 cm

step, the UGV will recalculate the potential field. The potential field received by each listener is calculated by adding the estimated distances from the listener to all the three active Cricket nodes.

6.3 Experiment Results

The moving trajectory of the navigation path is shown from one experiment trial in Figure 6.5. The Cricket nodes and the UGV are projected onto the same plane, where the nodes are shown as beacons in the figure. The starting point of the UGV is at the origin $(0, 0)$, and the navigation path can be seen from nodes with level 3 to the node with level 0. All steps (which are made after moving 15 cm) are represented by the small dots, and the local minimum points are shown as the large dots. Some parts of the real moving trajectory mismatch the ideal trajectory, which is caused by mechanical errors. In addition, as the arrow points out, the UGV might move to a wrong direction caused by error readings of listeners. However, it will go back to the right track as long as listeners and beacons can work properly. Figure 6.6 shows the potential fields (count by distance) received by listeners in certain steps (from step 24 to step 34) during the navigation. The UGV aims to the local minimum of each potential field and, consequently, it is noted that the sensed potential field reduces its value.

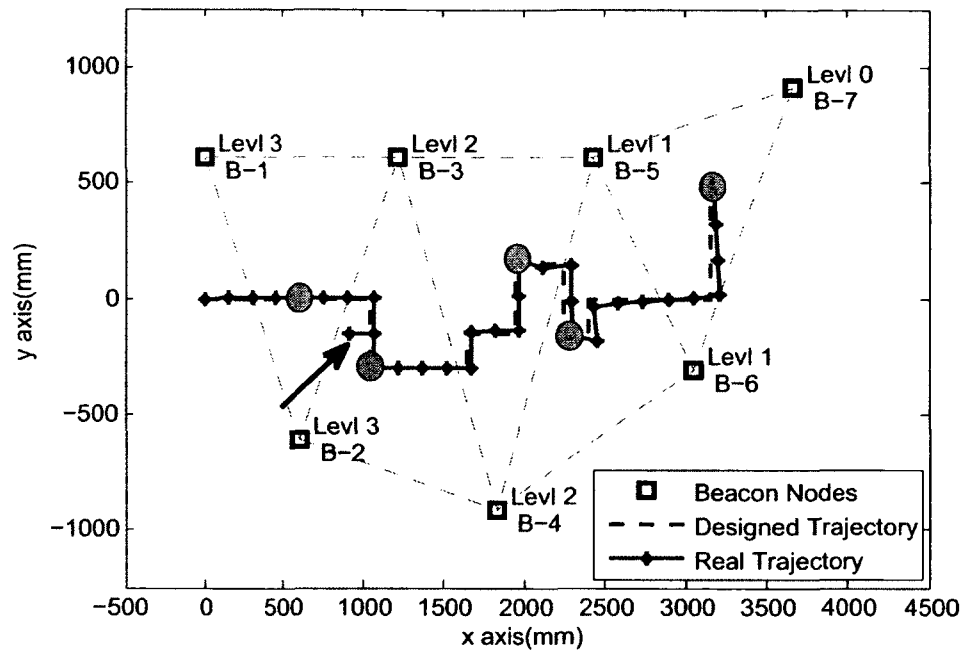


Figure 6.5: Trajectory of the one navigation. Besides the symbols labeled in the figure, local minimum points are highlighted by large dots. The part that the large arrow points to is an error movement, which might be caused by reading errors of listeners.

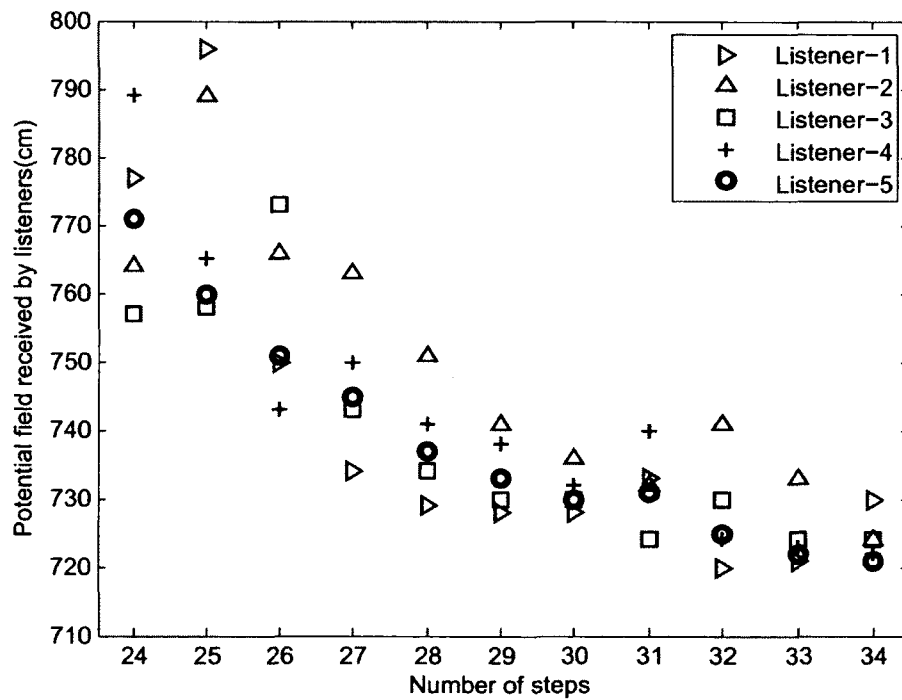


Figure 6.6: Potential fields calculated from listener readings during runtime.

CHAPTER 7

RESULTS AND DISCUSSION

This chapter includes the simulation comparisons and analysis of the proposed algorithms.

For illustrative purposes, in Figure 7.1, a network is shown as having a simple topology.

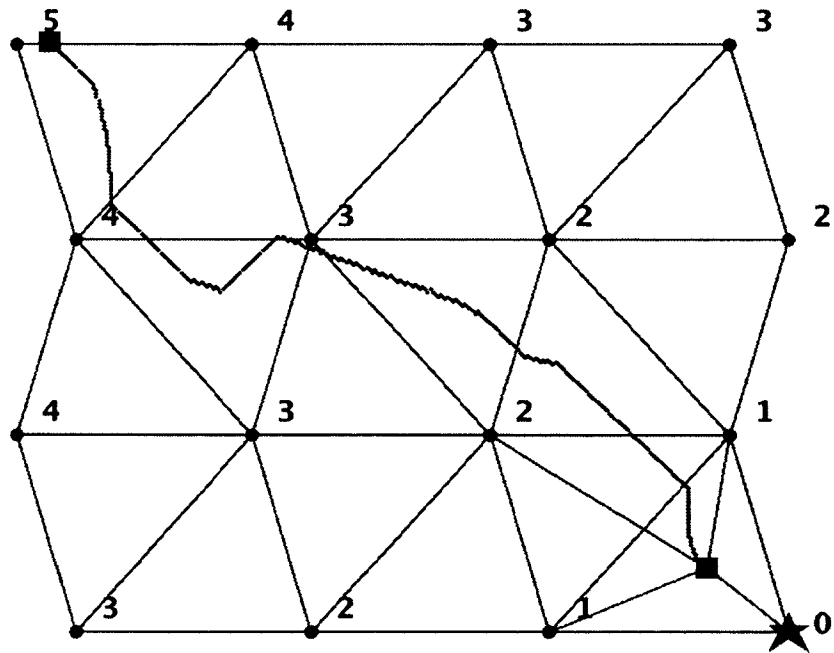


Figure 7.1: Illustration of a path taken by the UGV using our distributed navigation algorithm in a simple WSN.

The path taken during the UGV navigation in the WSN is highlighted. The destination node is designated with a star, and the start and end positions of the UGV

are designated with two black boxes. The dots and edges represent network nodes, whose level numbers are also shown, and the communication connections between them, respectively.

From the theoretical analysis, in the worst case, where the underlying communication graph of the WSN is a complete graph, Algorithm 2 is not practical as it has a communication complexity of $O(n^2)$. However, in most real-life cases, particularly in applications related to sensor network coverage, sensor nodes are sparsely deployed in order to increase the coverage area. In addition, because of the processing and energy limitations in sensor nodes, each node can only connect with a few other nodes.

Network density is calculated by $\sigma = n\pi r_c^2/s$ [79] with n being the number of nodes and s being the area of the sensor field. A fixed sensing area is used; the size is 800×600 and it deploys 400 nodes. The density is altered by varying the value of r_c . To illustrate, consider a network in which the underlying communication graph G is a plane graph such that every minimal region bounded by the edges of G , except the outside region, is an equilateral triangle with an edge length r_c . (The shape of G is similar to the graph shown in Figure 7.1.) Setting $r_c = 35$ yields a network density of $\sigma = 400 \times (35^2)\pi/(800 \times 600) \approx 3.2$.

Four hundred nodes are randomly deployed onto a sensor field of fixed size and the communication radius r_c is adjusted to control network density. The average number of messages sent by each node is first analyzed for Algorithm 2 using different network densities; see Figure 7.2. From the results, one can see that even when the network density is relatively high, the total number of messages sent is still far less than n^2 , which indicates that Algorithm 2 can be used for coverage related applications.

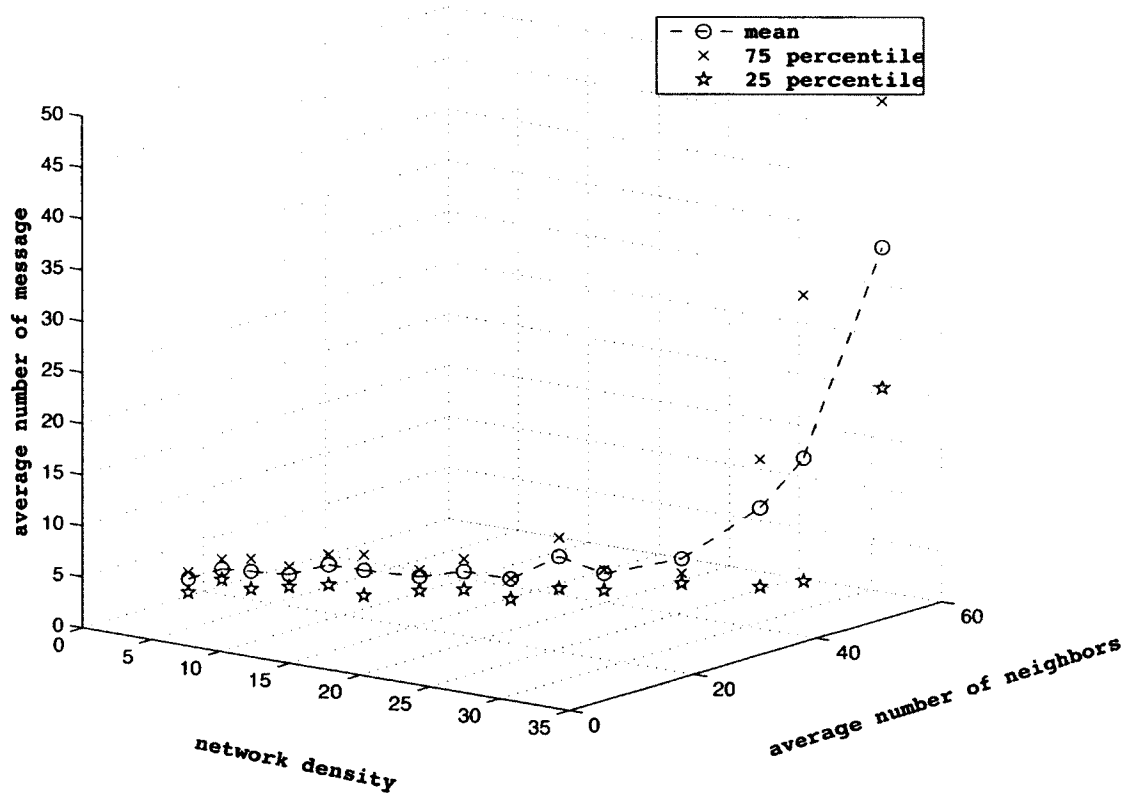


Figure 7.2: Average number of messages sent by each node in the distributed level assignment algorithm.

Both the centralized and distributed algorithms share the same core idea. From a computational perspective, one could say that the centralized algorithm performs faster than the distributed one because of the difference in data transmissions. For the centralized algorithm, the time used in transmitting data can generally be neglected since all essential data are located in the central controller, while in the distributed algorithm, the time used in sending and receiving data is much more significant. However, when one considers the physical navigation of the UGV, under reasonably efficient computation times, a better test of performance is to take into consideration the length of the path taken by the UGV, whose physical movement will easily dominate the overall time of the algorithm.

For the UGV control algorithms, the centralized version is compared with the distributed version by measuring the ratio $R_1 = d/d'$, where d is the UGV's actual moving distance and d' is the Euclidean distance from the start point of the navigation to the end point of the navigation. The ratio R_1 is a measure of the navigation efficiency: as R_1 approaches 1, the total distance taken approaches that of the ideal straight distance. Lower values of R_1 indicate a higher navigation efficiency. Figure 7.3 shows R_1 values for different network densities. Based on the analysis of R_1 , there is no significant difference between the two algorithms regarding their navigation efficiency, which as previously mentioned is expected since they share the same core idea.

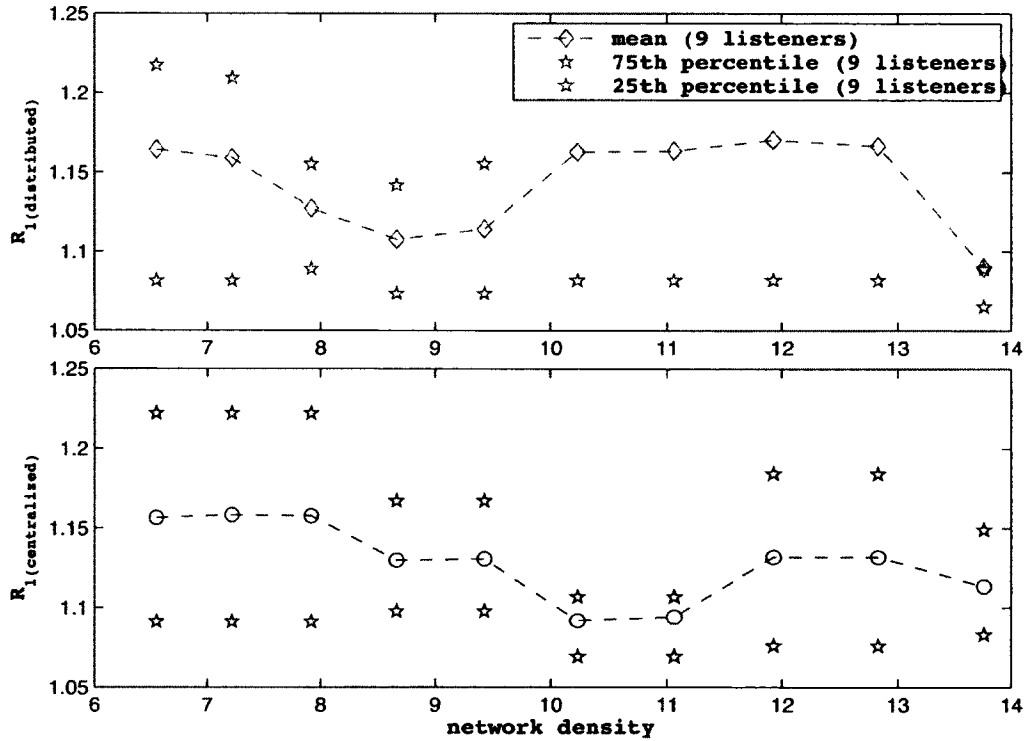


Figure 7.3: Evaluation of the distributed (top) and centralized (bottom) UGV control algorithms. R_1 shows the ability to control the main navigation direction.

Besides d' , the consecutive distances of local minimum points are also evaluated. Let u_1, u_2, \dots, u_k be the k local minimum points encountered in the navigation control algorithm, and let d^* be the length of the shortest path from the starting point of the UGV through each of these minimum points in succession. The ratio $R_2 = d/d^*$ is analyzed. Whereas R_1 measures the overall navigation efficiency, R_2 measures the ability to control the accuracy of the moving direction. A smaller value of R_2 indicates better accuracy since a straight path is more preferable than a zigzag path. From Equation 3.3 and the UGV control algorithms, better accuracy is obtained if the UGV compares more potential fields, which can be read and calculated from listeners placed on the UGV. Figure 7.4 shows the validation results from Algorithm 5.

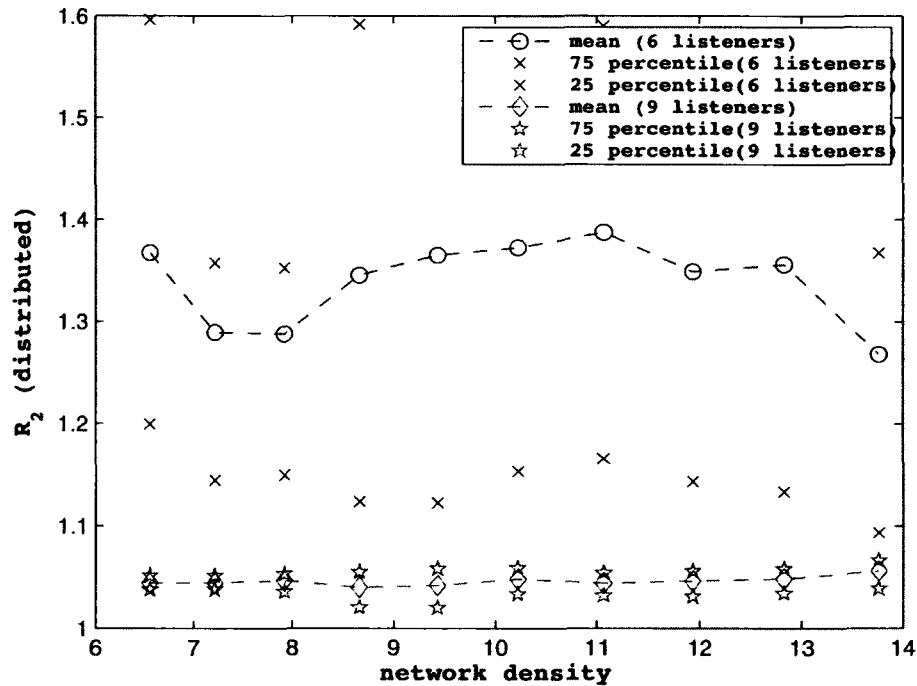


Figure 7.4: Evaluation of the UGV control algorithm: R_2 shows the control ability for the step movement.

Since they use the same general control logic, the simulation results for Algorithm 3 (Centralized UGV Control Algorithm) are omitted. Notice that the values of R_2 do not vary significantly with changes in network density, indicating that the moving direction of the UGV depends mainly on the number of listeners used.

The three-beacon navigation algorithm can easily be transformed to a one-beacon algorithm. In the centralized version, this can be accomplished by only finding and using the base nodes; in the distributed version, it is only necessary to assign and use node B . Figure 7.5 shows the comparison of R_1 from the distributed three-actuator control algorithm (Algorithm 5) and its one-actuator variant.

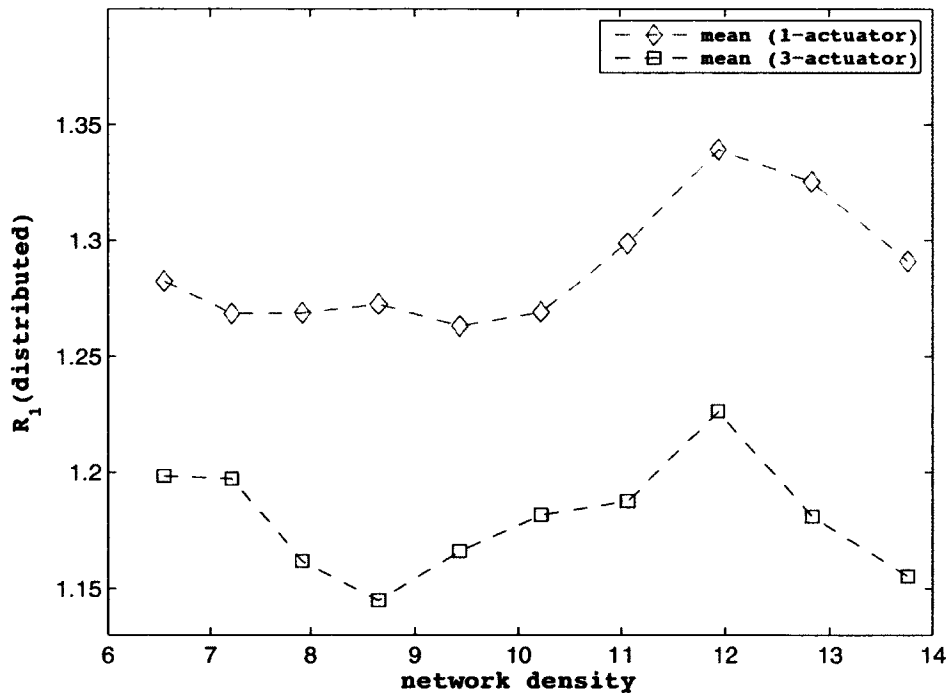


Figure 7.5: Comparison of one-actuator and three-actuator navigation with regards to R_1 .

It is worth considering the trade-off between using fewer beacons at any step, consequently conserving energy, and overall robustness when using more beacons; for example, a shorter travel distance saves the UGV considerable energy. Notice that navigating using three actuators performs better than using one actuator considering the total traveled distance. In real life, connection failures, node failures, or long transmission delays when the receiving of signals is interrupted are likely to occur due to changes in the environment. These connection failures are simulated by making certain random nodes “dead”, meaning the nodes no longer perform any function. As this situation is most applicable to distributed cases, the discussions here are regarding Algorithm 5 (Distributed UGV Control Algorithm). The UGV navigation can fail if the UGV cannot find active beacons.

In these simulations, the navigation mission is considered to have failed if the UGV cannot find a lower-leveled node. Specifically, for the one-actuator navigation, the navigation has failed if the UGV cannot connect to any of the possible B nodes. By contrast, for the three-actuator navigation, the UGV can tolerate up to two connection failures, where the node failure(s) should happen after nodes A , B and C are chosen, and at least one active node should not have a higher level than node B . In this case, even when node B is dead, the UGV can keep moving forward, without stopping to reset communication, in contrast to one-actuator navigation where every time the UGV loses a connection to a node B , the UGV must try to set up a new link with an alternative actuator, whose level number cannot be higher than node B . However, if the connection is lost before a node is chosen to be the node B , then there is no

difference between three-actuator navigation and one-actuator navigation since in three-actuator navigation, nodes A and C are chosen by node B .

For simplicity and ease of comparison, in the simulation examples, it is assumed all connection failures happen after node B has been chosen. Dead nodes are randomly picked in the simulation, and three-actuator and one-actuator navigation are run with a certain percentage of dead nodes. As shown in Figure 7.6, the mission failure rate is calculated $F = n_f/n_t$, where n_f is the number of failures and n_t is the total number of navigation missions. It is found that the three-actuator navigation algorithm performs much better than one-actuator navigation regarding connection failure and, in general, that values of F get smaller when the network density increases. The latter is mainly because the UGV can connect to more nodes when the network gets denser, allowing the UGV more opportunities to find another actuator to replace a failed node.

To test the efficiency of the Timer-based Leader Election Algorithm, the algorithm is run in a randomly generated network where nodes with distinct *ids* are deployed uniformly in the sensing field. Network density is still calculated by $\sigma = n\pi r_c^2/s$ as before. A network with 100 nodes was simulated. Very large networks were not considered since the leader election algorithm is not expected to run in a large group of nodes. Network density can be adjusted by changing the value of r_c . For example, setting $r_c = 98$ yields a network density of $\sigma = 100 \times (98^2)\pi / (800 \times 600 \approx 6.28)$. As Figure 7.7 shows, in a uniformly deployed network, the average number of messages transmitted is relatively small. Meanwhile, the number of transmitted messages is not related to network density.

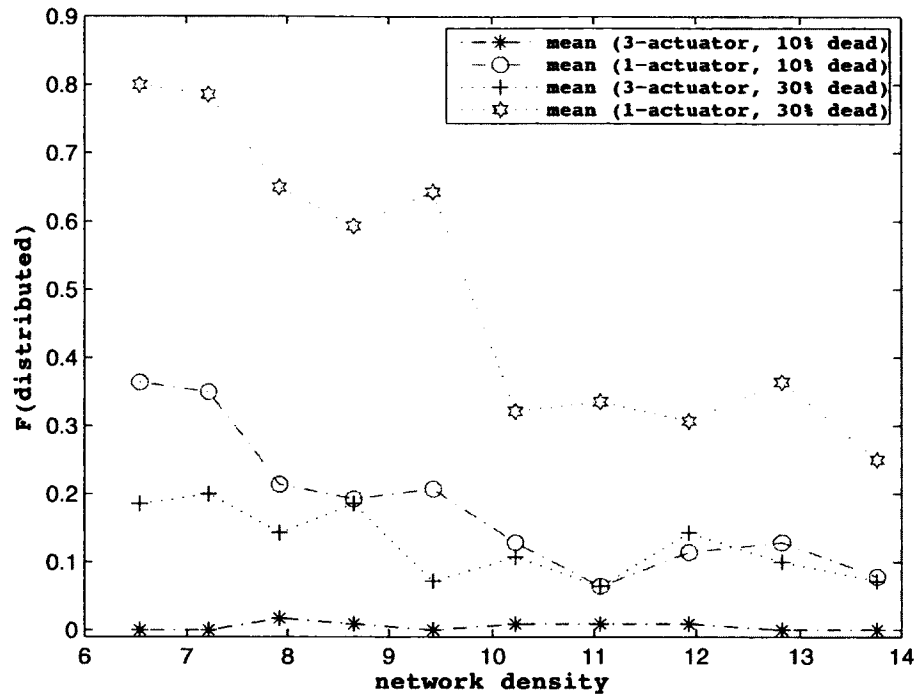


Figure 7.6: Comparison of the one-actuator and three-actuator navigation algorithms regarding the mission failure rate F .

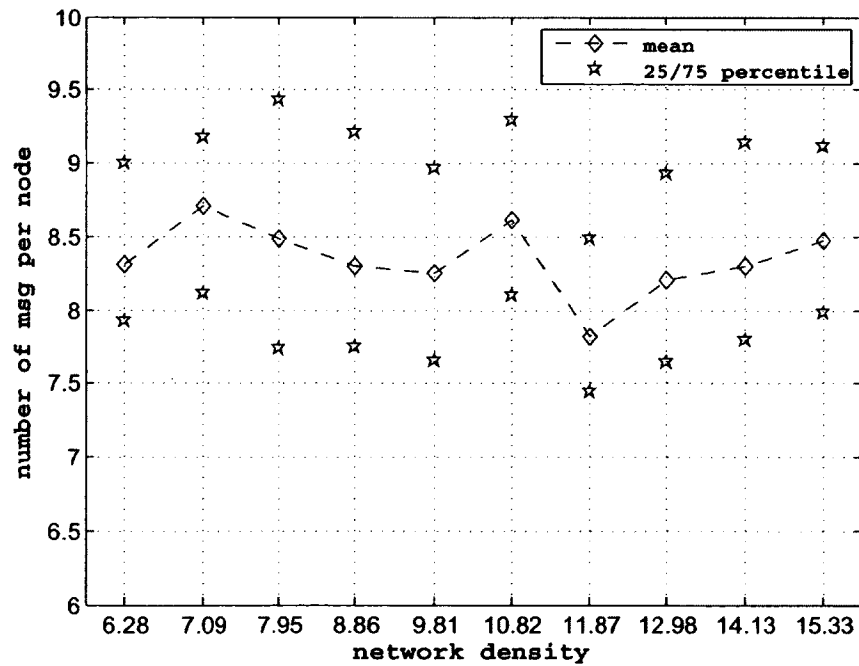


Figure 7.7: Number of messages sent per node in the leader election process.

The proposed WSAN-aided Greedy Task Allocation Algorithm was also evaluated, considering only traveling distance without turning angles. First, a genetic algorithm is used as the benchmark for comparison purposes. The genetic algorithm tests 80 samples in each iteration and runs 2000 iterations for each simulation. The results obtained by the genetic algorithm are roughly considered very close to the optimal solutions. A simple algorithm that does not include any allocation process is also run. When there exist destinations that have not been serviced yet, each UGV will just converge to the nearest destination without checking if any other UGV is heading towards the same destination.

As shown in Figure 7.8, the proposed algorithm and simple contrast algorithm are both compared with the genetic algorithm when multiple UGVs start at the same positions. It can be seen that while the proposed algorithm is far better than the simple algorithm, it is, in general, not worse than twice the genetic algorithm. The performance of the proposed algorithm deteriorates when the number of destinations is significantly larger than the number of UGVs, such as 3/12, that is, when there are 3 UGVs for 12 destinations. The proposed algorithm requires around 2.5 times the distance than the one by the genetic algorithm. On the other hand, the simple algorithm requires 7 times more than the genetic one. No optimal solution can be guaranteed since the system has no global information, and coordinates of nodes and UGVs are not available. Therefore, it is not expected that the proposed WSAN-Aided Nearest Neighbor Algorithm has a better performance than the Nearest Neighbor Algorithm in each and every case. However, simulation results demonstrate that the

proposed algorithm performs better than the simple Nearest Neighbor Algorithm when angle change is taken into account in the cost function.

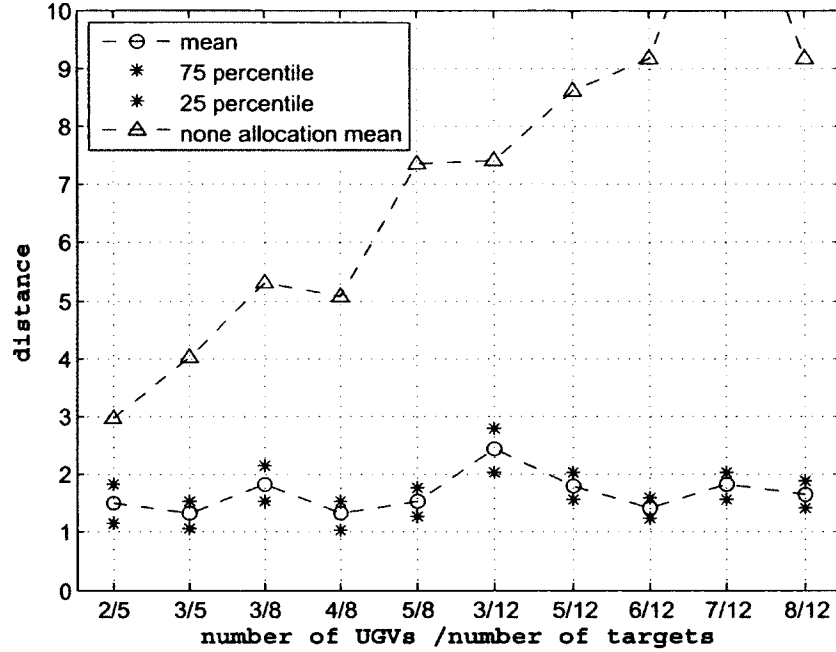


Figure 7.8: Performance ratio of the traveled distance by the proposed algorithm and the genetic algorithm (circles), and performance ratio of the traveled distance by the simple algorithm and the genetic algorithm (triangles) in the multi-UGV, multi-destination scenario when UGVs start at the same locations.

In Figure 7.9 two values are plotted: $ratio_1$ shows the percentage of networks where the proposed algorithm performs better than or equal to the simple Nearest Neighbor Algorithm considering distance and angle change, respectively; $ratio_2$ shows the ratio of the cost function (4.1) when both of the distance and angle change are both considered. Experiments were run using a Pioneer 3-DX mobile robot to estimate the weighting factors P and Q . By setting Pioneer 3-DX's velocity to the maximum value, linear velocity was measured as 170 centimeters per second and angular velocity as 90 degrees per second. Based on this measurement, normalized weighting factors were

assigned as $P = 0.35$ and $Q = 0.65$. Note that the proposed algorithm outperforms the simple algorithm regarding the angle change only. See Figure 7.9. In contrast, it under-performs when a large number of targets is present because the network gets dense and turns, with large angle changes needed to navigate among targets. Although the proposed algorithm does not show much advancement regarding ratio_2 at the selected P and Q , it can be claimed that the proposed algorithm has the potential to outperform based on the results of ratio_1 .

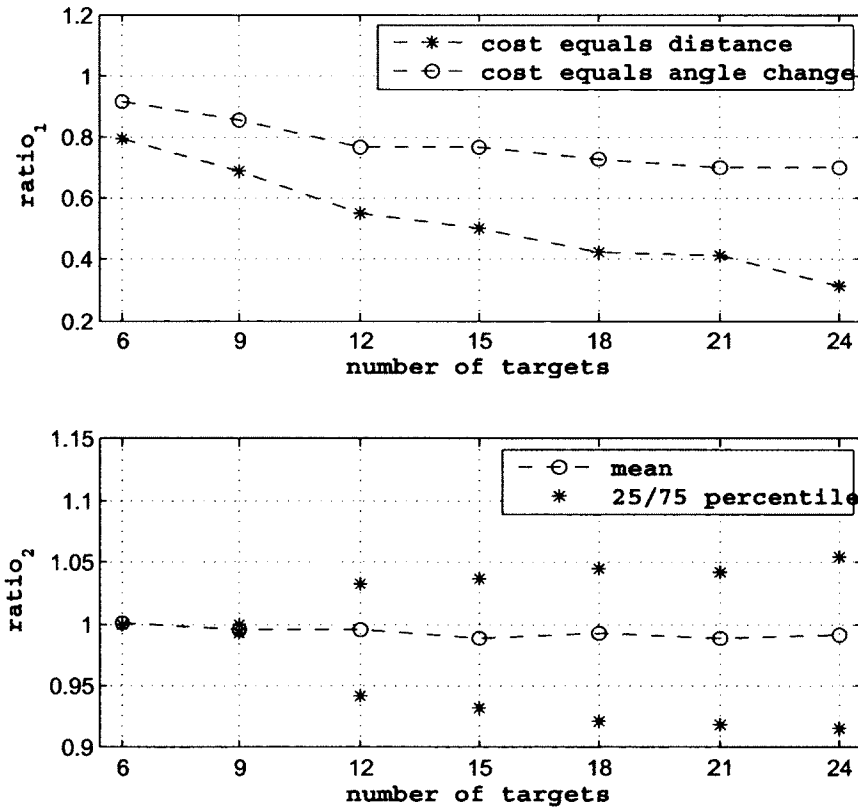


Figure 7.9: Comparison of results by the proposed algorithm to the simple nearest neighbor algorithm in the single UGV and multiple destination scenario.

A randomly deployed network is shown in Figure 7.10. The communication connections are simplified by maximum simplicity to detect coverage holes, which

are plotted in the figure. Figure 7.11 shows the results after running the proposed hole patching algorithm. It demonstrates that this hole patching algorithm works well when the coverage holes are an irregular shape.

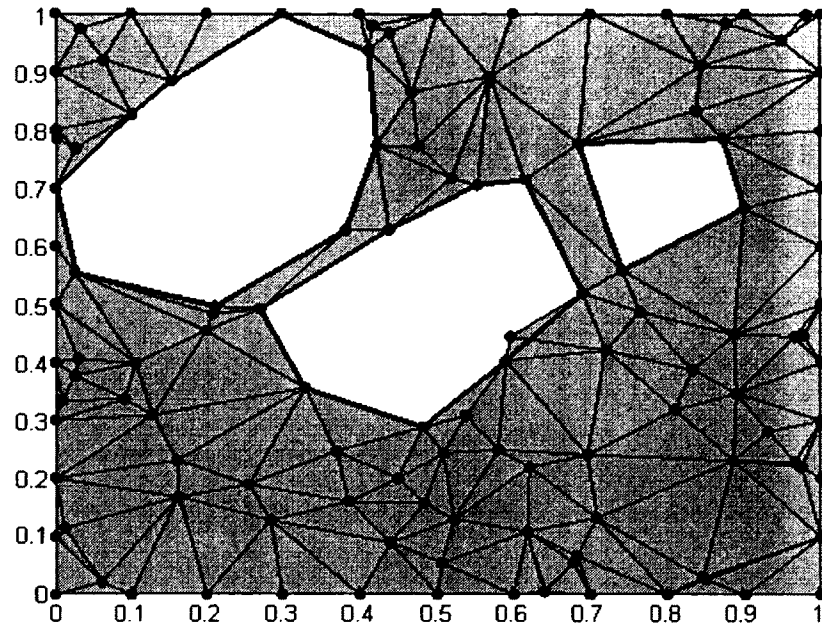


Figure 7.10: A randomly deployed network with three coverage holes.

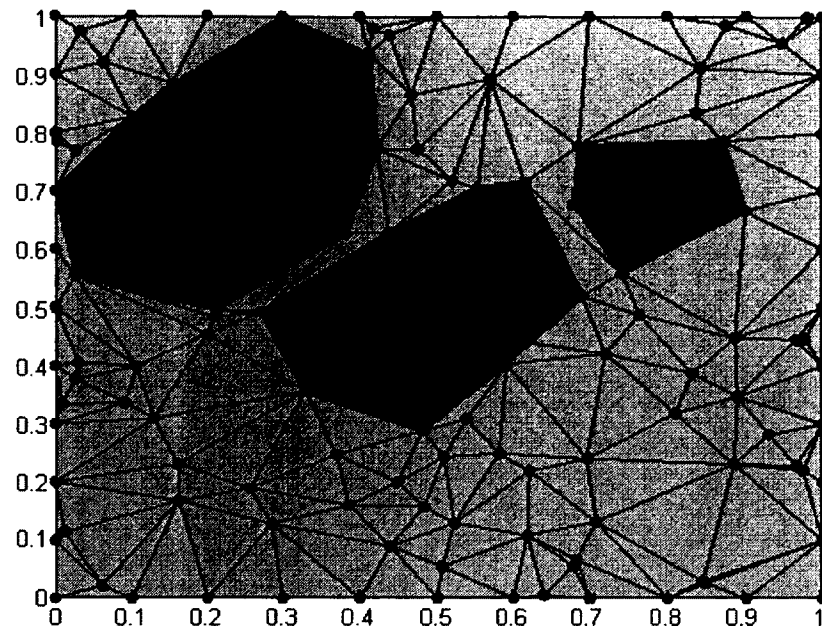


Figure 7.11: The coverage holes are patched after the patching algorithm.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

Presented in this dissertation is the theoretical analysis and the simulation verification of the proposed algorithms in a coordinate-free wireless sensor and actuator network environment. The algorithms are described for network hop-distance identification, UGV navigation control, and analysis is provided of the time and space complexity of the centralized version algorithms and the more relevant communication complexity of the distributed cases. A coverage hole patching algorithm is presented for networks with holes in sensing coverage.

Though the current work considered problems in an open field devoid of obstructions, for more diverse applications, algorithms can be extended to be capable of obstacles avoidance. The problem under consideration as well as some assumptions are stated in the following.

Wall-following is the simplest obstacle avoidance method, by which a robot just follows the edges of obstacles until return to the initial track. For example, the Pioneer 3DX robot used for the experiment can follow a wall with an array of sonars. There are various complicating factors in a model with obstacles. For one, in a distributed unmapped terrain, the positions of the obstacles would be unknown, necessitating avoidance being done as the obstacles are encountered. Second, assumptions must

be on the hardware available to a UGV in order to detect obstacles, ranging from ultrasonic sensors that operate in one general direction to video cameras that have a much larger field of vision but also a much larger energy requirement. When trying to reach an unknown target or incrementally a local actuating signal minimum, determining which direction ideally to traverse an obstacle, i.e. clockwise or counter-clockwise, is not necessarily straight-forward, particularly when the end destination is not known. When an obstacle is on the UGV navigation path, as demonstrated in Figure 8.1, the proposed UGV control algorithm can possibly fail if the UGV cannot connect to an alternate node. Although the wall-following method can always find a path if the target is not completely cut off by the obstacle, the final goal is to find an efficient solution which can travel a relatively shorter distance most of the time.

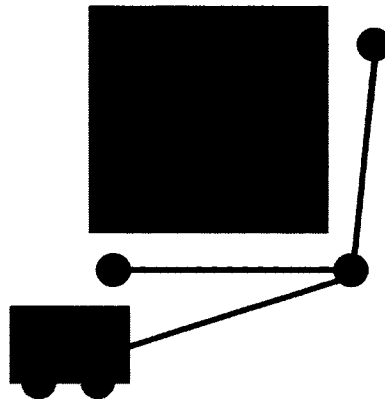


Figure 8.1: Local minimum point is occupied by an obstacle.

To avoid having a “maze-like” environment where the main solution is to do a wall following algorithm, some reasonable assumptions can be made on the size, shape, and relative proximity of obstacles.

To simplify the problem, it is assumed that first, each obstacle is convex, and obstacles are spread far enough to each other, thus excluding the scenario that multiple obstacles are overlapped to form a non-convex obstacle shape. Second, to ensure the actuating signal can be received, while ignoring the distance between listeners and ultrasonic sensors on the UGV, the diameter d_o (the largest distance between two points on the perimeter of the obstacle) of single obstacle is bounded by $d_o \leq r_a - 2\rho - d_t - \delta$ ($r_a \geq d_o + 2\rho + d_t + \delta$), where r_a is the radius of actuation signal, ρ is the radius of listener ring on the UGV, d_t is a variable which defines the shortest distance between the UGV and the obstacle, and δ is a small value that defines the distance between an active actuator node and the obstacle, as shown in Figure 8.2.

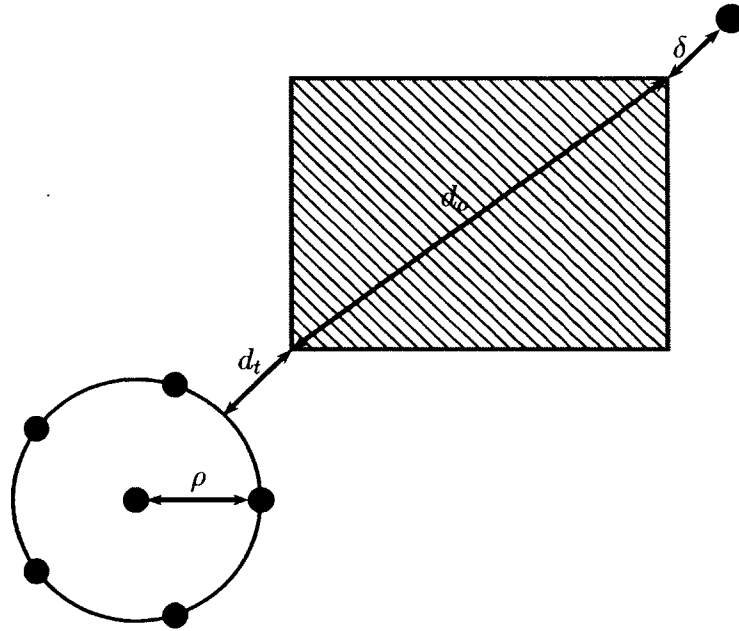


Figure 8.2: Representation of the size limit on the obstacle. The listeners ring represents the UGV.

Note that this assumption has an underlying condition – there is no attenuation when signal goes through an obstacle. This condition can later be removed when

applying certain attenuation factor. Third, the UGV will not detect another obstacle while trying to go around an obstacle.

In the future, navigation that integrates sensing data from sensors in the network will also be considered. For example, the amplitude of each individual actuating signal could be adjusted by readings from certain sensors, thus allowing the navigation path to be controlled according to the environment. A node energy model will also be formulated for consideration in optimizing the navigation path.

BIBLIOGRAPHY

- [1] G. Alankus, N. Atay, C. Lu, and O. B. Bayazit. Spatiotemporal query strategies for navigation in dynamic sensor network environments. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, pages 3718–3725, 2005.
- [2] H.M. Ammari and S.K. Das. On computing conditional fault-tolerance measures for k-covered wireless sensor networks. In *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, pages 309–316, New York, NY, USA, 2006.
- [3] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(2):224 –241, March/April 1992.
- [4] M.A. Batalin, G. S. Sukhatme, and M. Hatting. Mobile robot navigation using a sensor network. In *Proc. IEEE International Conference on Robotics and Automation*, volume 1, pages 636–641, 2003.
- [5] M.A. Batalin and G.S. Sukhatme. Sensor network-based multi-robot task allocation. In *IROS '03*, volume 2, pages 1939 – 1944, Orlando, FL, October 2003.
- [6] M.A. Batalin and G.S. Sukhatme. Using a sensor network for distributed multi-robot task allocation. In *ICRA '04*, pages 158–164, New Orleans, LA, April 2004.

- [7] D. Bertsekas and R. Gallager. *Data networks*, chapter 5.2.4. Prentice-Hall, Inc, 1987.
- [8] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(5):1179 – 1187, September/October 1989.
- [9] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.
- [10] J.G. Buchart. Detecting coverage holes in wireless sensor networks. Master’s thesis, Louisiana Tech University, May 2008.
- [11] C. Buragohain, D. Agrawal, and S. Suri. Distributed navigation algorithms for sensor networks. In *Proceedings of 25th IEEE International Conference on Computer Communications*, pages 1–10, April 2006.
- [12] J.E. Burns. A formal model for message passing system. Technical report, Indiana University, Bloomington, IN, September 1980.
- [13] K. Chakrabarty, S.S. Iyengar, H. Qi, and E. Cho. Grid coverage for surveillance and target location in distributed sensor networks. *Computers, IEEE Transactions on*, 51(12):1448 – 1453, December 2002.
- [14] A. Chandrakasan, R. Amirtharajah, S. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang. Design considerations for distributed microsensor systems. In *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, pages 279 –286, 1999.

- [15] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22:281–283, May 1979.
- [16] D. Chen, B. Kumar, C.K. Mohan, K.G. Mehrotra, and P.K. Varshney. In-network path planning for distributed sensor network navigation in dynamic environments. In *Proc. IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 511–513, October 2008.
- [17] P. Chen, W. Chen, and Y. Shen. A distributed area-based guiding navigation protocol for wireless sensor networks. In *Proc. IEEE International Conference on Parallel and Distributed Systems*, pages 647–654, December 2008.
- [18] W. Chen and X. Li. Sensor localization under limited measurement capabilities. *IEEE Network*, 21(3):16–23, May-June 2007.
- [19] W. Chen, T. Mei, H. Liang, Z. You, S. Li, and M.Q.-H. Meng. Environment-map-free robot navigation based on wireless sensor networks. In *Proc. IEEE ICIA*, pages 569–573, August 2007.
- [20] L.P. Clare, G.J. Pottie, and J.R. Agre. Self-organizing distributed sensor networks. In *SPIE Conference on UGSTA*, pages 229–237, April 1999.
- [21] B. Coltin and M. Veloso. Mobile robot task allocation in hybrid wireless sensor networks. In *IEEE/RSJ International Conference on IROS*, pages 2932–2937, Taipei, October 2010.
- [22] P. Corke, R. Peterson, and D. Rus. Localization and navigation assisted by cooperating networked sensors and robots. *International Journal of Robotics Research*, 24(9):771–786, October 2005.

- [23] P. Corke and G. Sukhatme. Deployment and connectivity repair of a sensor net with a flying robot. In *Proc. the 9th International Symposium on Experimental Robotics*, pages 333–343, 2004.
- [24] V. De Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *Int. J. Rob. Res.*, 25(12):1205–1222, December 2006.
- [25] X. Deng, C. Xu, F. Zhao, and Y. Liu. Repair policies of coverage holes based dynamic node activation in wireless sensor networks. In *IEEE/IFIP 8th International Conference on EUC*, pages 368 –371, December 2010.
- [26] M.J. Dong, K.G. Yung, and W.J. Kaiser. Low power signal processing architectures for network microsensors. In *Pros. International Symposium on Low Power Electronics and Design*, pages 173 – 177, August 1997.
- [27] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.
- [28] M. Foskey, M. Garber, M.C. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *Procs. 2001 IEEE/RSJ International Conference on IRS*, volume 1, pages 55 –60, 2001.
- [29] S. Fu, Z. Hou, and G. Yang. An indoor navigation system for autonomous mobile robot using wireless sensor network. In *Proc. IEEE International Conference on Networking, Sensing and Control*, pages 227–232, May 2009.
- [30] R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5:66–77, January 1983.

- [31] J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Transactions on Networking*, 5:148–160, 1997.
- [32] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [33] R. Ghrist and A. Muhammad. Coverage and hole-detection in sensor networks via homology. In *IPSN*, pages 254–260, 2005.
- [34] C. Huang and Y. Tseng. The coverage problem in a wireless sensor network. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 115–121, New York, NY, USA, 2003.
- [35] P. A. Humblet. An adaptive distributed dijkstra shortest path algorithm. Technical report, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1988.
- [36] J.A. Janet, R.C. Luo, and M.G. Kay. The essential visibility graph: an approach to global motion planning for autonomous mobile robots. In *Procs. IEEE International Conference on RA*, volume 2, pages 1958 –1963, May 1995.
- [37] A. Kapse, Dongbing Gu, and Zhen Hu. Using cricket sensor nodes for pioneer robot localization. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 2008 –2013, August 2009.
- [38] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. IEEE International Conference on RA*, volume 2, pages 500–505, 1985.
- [39] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems*, 12:84–101, 1990.

- [40] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 1398 –1404, April 1991.
- [41] W. Lee, K. Hur, and D. Eom. Navigation of mobile node in wireless sensor networks without localization. In *Proc. IEEE International Conference on MFIIS*, pages 1–7, Seoul, August 2008.
- [42] G. LeLann. Distributed systems - towards a formal approach. In *IFIP Congress'77*, pages 155–160, 1977.
- [43] Q. Li, M. DeRosa, and D. Rus. Distributed algorithms for guiding navigation across a sensor network. In *Proc. the 9th Annual International Conference on Mobile Computing and Networking, MobiCom '03*, pages 313–325, 2003.
- [44] X. Li, D.K. Hunter, and K. Yang. Wlc12-1: Distributed coordinate-free hole detection and recovery. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1 –5, December 2006.
- [45] Y. Li and M.T. Thai, editors. *Wireless Sensor Networks and Applications*. Springer, 2008.
- [46] J. Liu, Y. Feng, H. Xu, J. Xue, and M. Qian. Greedy approximation algorithm of patching hole in wireless sensor networks. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 2604 –2608, April 2012.
- [47] N. Marchetti, N.R. Prasad, J. Johansson, and Tao Cai. Self-organizing networks: State-of-the-art, challenges and perspectives. In *Communications (COMM), 2010 8th International Conference on*, pages 503 –508, june 2010.

- [48] S. Martinez, J. Cortes, and F. Bullo. Motion coordination with distributed information. *Control Systems, IEEE*, 27(4):75–88, August 2007.
- [49] E Masehian and M.R. Amin-Naseri. A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *J. Robot. Syst.*, 21(6):275–300, June 2004.
- [50] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1380–1387, 2001.
- [51] T. Mercker, M. Akella, and J. Alvarez. Robot navigation in a decentralized landmark-free sensor network. *Journal of Intelligent and Robotic Systems*, 60(3-4):553–576, 2010.
- [52] P.M. Merlin. Design and analysis of distributed routing algorithms. Master’s thesis, University of California, Santa Cruz, 1994.
- [53] MIT Computer Science and Artificial Intelligence Lab. *Cricket User Manual*, second edition, January 2005.
- [54] MobileRobots ActivMedia Robotics. *Pioneer 3 Operations Manual*, third edition, January 2006.
- [55] A.K. Mohammad. Mobile robot navigation using cricket indoor location system. Master’s thesis, Louisiana Tech University, 2008.
- [56] L. Montestruque and M. Lemmon. Csonet: a metropolitan scale wireless sensor-actuator network. In *International Workshop on Mobile Device and Urban Sensing (MODUS)*, 2008.

- [57] K.J. O'Hara, D.B. Walker, and T.R. Balch. Physical path planning using a pervasive embedded network. *IEEE Transactions on Robotics*, 24(3):741 – 746, June 2008.
- [58] Oracle. Class thread. <http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html>.
- [59] Oracle. Java object. <http://docs.oracle.com/javase/tutorial/java/java00/objects.html>.
- [60] C.H. Papadimitriou and E.B. Silverberg. Optimal piecewise linear motion of an object among obstacles. *Algorithmica*, 2:523–539, 1987.
- [61] L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220 –240, April 1998.
- [62] D. Puccinelli and M. Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *IEEE Circuits and Systems*, 5(3):19 – 31, 2005.
- [63] T. Rappaport. *Wireless Communications: Principles & Practice*. Prentice-Hall, Inc, New Jersey, 1996.
- [64] J. Schiff, A. Kulkarni, D. Bazo, V. Duindam, R. Alterovitz, D. Song, and K. Goldberg. Actuator networks for navigating an unmonitored mobile robot. In *Proc. of IEEE Conference on Automation Science and Engineering*, pages 53–60, Washington DC, August 2008.
- [65] V. De Silva and R. Ghrist. Homological sensor networks. *Notices of the American Mathematical Society*, 54:2007, 2007.
- [66] J. Spinelli. Broadcasting topology and routing information in computer networks. Master's thesis, Massachusetts Institute of Technology, May 1985.

- [67] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, 2000.
- [68] J. Tsitsiklis and G. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. *Journal of the ACM*, 42(2):382–400, 1995.
- [69] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In *DARPA Information Survivability Conference and Exposition*, volume 1, pages 261 – 272, April 2003.
- [70] A. Verma, H. Sawant, and J. Tan. Selection and navigation of mobile sensor nodes using a sensor network. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 41 –50, March 2005.
- [71] A. Viguria and A.M. Howard. An integrated approach for achieving multirobot task formations. *Mechatronics, IEEE/ASME Transactions on*, 14(2):176 –186, April 2009.
- [72] S. Wang and H. Hu. Three-dimensional localization using cricket system. Technical report, School of Computer Science and Electronic Engineering, University of Essex, December 2011.
- [73] R. Wu, J. He, T. J.. Li, and H. Shi. Energy-efficient coverage hole self-repair in mobile sensor networks. In *Proceedings of the 2009 International Conference on New Trends in Information and Service Science*, NISS '09, pages 1297–1302, Washington, DC, USA, 2009. IEEE Computer Society.
- [74] N. Xu. A survey of sensor network applications. *IEEE Communications Magazine*, 40, 2002.

- [75] J. Yao, G. Zhang, J. Kanno, and R.R. Selmic. Decentralized detection and patching of coverage holes in wireless sensor networks. In *Proc. SPIE Defense and Security*, pages 73520V–73520V–10, 2009.
- [76] G. Zhang, C.A. Duncan, J. Kanno, and R.R. Selmic. Unmanned ground vehicle navigation in coordinate-free and localization-free wireless sensor and actuator networks. In *Proc. 2010 IEEE Multi-conference on Systems and Control*, pages 428–433, Yokohama, Japan, September 2010.
- [77] G. Zhang, C.A. Duncan, J. Kanno, and R.R. Selmic. Distributed unmanned ground vehicle navigation in coordinate-free and localization-free wireless sensor and actuator networks. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 7262 –7267, December 2011.
- [78] G. Zhang, J. Li, C.A. Duncan, J. Kanno, and R.R. Selmic. Ugv navigation in wireless sensor and actuator network environments. In *Proc. SPIE Defense and Security*, pages 83870G–83870G–11, Baltimore, MD, April 2012.
- [79] H. Zhang and J. C. Hou. Maximizing α -lifetime for wireless sensor networks. *Int. J. Sen. Netw.*, 1:64–71, September 2006.