SYMBIOSIS: COOPERATIVE ALGORITHMS FOR MOBILE ROBOTS AND A SENSOR NETWORK

by

Maxim Alexander Batalin

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2005

Copyright 2005 Maxim Alexander Batalin
# Contents

## List Of Tables

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv</td>
</tr>
</tbody>
</table>

## List Of Figures

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
</tbody>
</table>

## Abstract

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Motivation .................................................. 1
1.2 Design Principle - Symbiosis ............................... 7
1.3 Problem Description and Thesis Statement ................. 9
1.4 Contributions ............................................... 10
1.5 Outline .................................................... 15

## 2 Coverage and Exploration through Sensor Network Deployment

2.1 Related Work and Assumptions ............................... 18
2.2 LRV: Least Recently Visited Algorithm ................... 21
2.3 The Graph Model ............................................ 22
   2.3.1 LRV on Graphs ......................................... 24
   2.3.2 LRV on a Square Lattice: Empirical Results from Simulation .......... 26
   2.3.3 LRV on a Tree .......................................... 34
2.4 LRV Implementation Details ................................. 36
2.5 Simulation Experiments ...................................... 39
2.6 Implicit Sensor Network Repair and Maintenance ........... 42
2.7 Remarks on Generalization .................................. 42
2.8 Summary ................................................... 45

## 3 Mobile Robot Navigation using a Sensor Network

3.1 Related Work ................................................ 48
3.2 Probabilistic Navigation ..................................... 50
   3.2.1 Computing Transition Probabilities .................... 51
   3.2.2 Computing the Navigation Field ......................... 52
      3.2.2.1 Theoretical Framework - Value Iteration ............. 52
      3.2.2.2 Distributed Computation and In-network Processing .... 54
   3.2.3 Navigation and Practical Considerations ............... 55
3.3 Navigation Experiments ..................................... 59
3.4 Combined Coverage and Navigation Experiments . . . . . . . . . . . . . . . . . . 61
3.4.1 First Phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.4.2 Second Phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
3.4.3 Third Phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
3.4.4 Fourth Phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
3.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

4 Network-mediated Multi-Robot Task Allocation 68
4.1 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.2 Task Allocation: Offline vs. Online . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.3 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.4 DINTA: Distributed In-Network Task Allocation . . . . . . . . . . . . . . . . . 74
4.5 The Multi-Field Approach: DINTA-MF . . . . . . . . . . . . . . . . . . . . . . 78
4.6 Simulation Experiments with MRTA . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.6.1 Efficient Exploration vs. DINTA . . . . . . . . . . . . . . . . . . . . . . 81
4.6.2 DINTA vs. DINTA-MF . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
4.7 Optimality Considerations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.8 Explicit Sensor Network Repair and Maintenance . . . . . . . . . . . . . . . . . 83
4.9 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84

5 Task Allocation in a Networked Info-Mechanical System 88
5.1 Introduction to Networked Info-Mechanical Systems . . . . . . . . . . . . . . . . 88
5.1.1 NIMS Test Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
5.2 NIMS Task Allocation System . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
5.3 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
5.3.1 Task Allocation vs. Raster Scan . . . . . . . . . . . . . . . . . . . . . . . 96
5.3.2 Experimental Results for Time and Distance Policies . . . . . . . . . . . . 99
5.3.2.1 Experiments in the Mountain Reserve . . . . . . . . . . . . . . . . 102
5.3.2.2 TA Policies: Time vs. Distance . . . . . . . . . . . . . . . . . . . . 102
5.3.3 Multi-Robot System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
5.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107

6 Conclusion 109

Reference List 116

Appendix A
Publications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
List Of Tables

3.1 Experimental data (distance to goal at finish, in meters). Five goals, ten trials per goal. ................................................................. 62

4.1 Tradeoffs between different approaches. ........................................ 86
# List Of Figures

1.1 Schematics of an Ecosystem. (Courtesy of Thomas Harmon and William Kaiser.)  

1.2 a) A schematic showing a static sensor network deployed for ecosystem monitoring. b) A schematic showing mobile robots deployed for ecosystem monitoring.  

1.3 Spatiotemporal variation of light intensity. (Courtesy of the NIMS project [45])  
a)-d) Change in solar light intensity at the same location on a forest floor over time. e) Map of solar radiation intensity obtained in a forest ecosystem by a NIMS system (Chapter 5) transporting a light intensity sensor within the canopy. Solar radiation intensity is indicated in contours as it varies spatially according to horizontal and vertical sensor displacement.  

1.4 A schematic of a ‘symbiotic’ system. The robots deploy and maintain the network of static sensors while achieving high fidelity temporal sampling. The network mediates robot navigation and tasking while achieving high fidelity spatial sampling.  

1.5 The process of coverage and exploration. a)-c) A robot deploys the sensor network, while exploring the environment. d)-f) A robot detects the gap in the network and repairs it.  

1.6 Using a network for a)-c) Robot navigation. d)-f) Multi-robot task allocation.  

2.1 Modeling the network as a graph.  

2.2 Illustration for Theorem 1. At every point in time during the execution of LRV a graph $G = G_{exp} + G_u$, where $G_{exp}$ is the explored part of $G$ and $G_u$ is an unexplored part. An edge $e_{ij}$ connects $G_{exp}$ and $G_u$.  

2.3 Comparison of graph coverage algorithms. The $n \ln n$ curve is shown for reference.  

2.4 Comparison of Cover time of DFS, 1-LRTA* and LRV.
2.5 Comparison of the convergence speed for LRV, 1-LRTA* and RW. a) On the interval 0 - 100%. b) Magnified view of a, interval 50 - 100%.

2.6 a) A map of the environment with an embedded sensor network with a tree-like topology. b) Tree isomorphic to embedded sensor network topology of a.

2.7 Illustration for Lemma 1. A tree $T$ at a point of time when a robot enters vertex $v_i$ through an incoming edge $e_{ij}$.

2.8 Illustration for Lemma 2. A tree $T$ at a point of time when a robot enters vertex $v_i$ of an unexplored subtree $T'$ through an incoming edge $e_{ij}$.

2.9 Illustration for Theorem 3. A tree $T$ augmented with a vertex $v'$ and an edge $e_{v'v}$ connecting $v'$ to vertex $v \in T$.

2.10 a) System architecture showing robot behaviors; b) Node architecture.

2.11 Behavior Switching. a) The robot is executing $SearchBeacon$ behavior traversing suggested direction; b) The robot is executing $AtBeacon$ behavior, analyzing sensor readings; c) The robot is executing $SearchBeacon$ behavior, supposing the node suggests direction $UP$ and there are no obstacles detected in the sensor data; d) The robot is executing $SearchBeacon$ behavior traversing direction, not originally suggested by the node.

2.12 Simulation results for different environment sizes across 50 trials. Our algorithm consistently outperforms random walk by an order of magnitude, and is significantly more stable.

2.13 Average cover times for three different grid sizes in simulation. Environment sizes are $25m^2$, $49m^2$ and $100m^2$.

2.14 Deployment of nodes in a representative simulation trial. Note that due to noise added in simulation, the deployed nodes do not form a perfectly square lattice.

3.1 An example of a discrete probability distribution of vertex (node) $k$ for direction (action) "East" (i.e. right).

3.2 Navigation - node-wise approach.

3.3 Illustration of Delta Percent Algorithm. Arc $R$ represents an ideal signal strength space that the robot receives starting from node $t$, going towards node $x$. Note that metric data is not involved in the figure and the desired 'switching' place is around areas $M_1$ or $M_2$. 
3.4 Map of the experimental environment. Nodes (marked 1 - 9) were manually predeployed ........................................... 58
3.5 Mobile robot and a Mote in experimental setting ........................................... 59
3.6 Trajectories of robot navigating to five different goals during different trials. The start location in each case is near node 1 ........................................... 60
3.7 Network deployment, exploration and coverage ........................................... 63
3.8 Network repair. NR is the area requiring repair ........................................... 64
3.9 Deployment of additional nodes into the discovered open space ................... 65
3.10 Coverage over the first three stages of the experiment. The robot continued coverage of the environment and was not affected by the problems in the network ........................................................................................................ 65
3.11 Robot navigation through the environment using directives from the sensor network ........................................... 66
4.1 Examples of a navigation field computed by DINTA ........................................... 75
4.2 Node architecture in DINTA ........................................... 76
4.3 Node Architecture in DINTA-MF ........................................... 77
4.4 Comparison between implementation of DINTA and exploration (LRV) ................................. 80
4.5 Experimental comparison between DINTA-MF and DINTA ........................................... 82
5.1 NIMS system deployed in the forest reserve for continuous operation. (Courtesy of the NIMS project [45]) ........................................... 89
5.2 NIMS deployed at the James San Jacinto Mountain Reserve (http://www.jamesreserve.edu). a) This image shows the NIMS cable infrastructure, horizontal transport node (carrying an embedded computing platform, image sensor, vertical transport control, and vertically mobile meteorological sensor node. (Courtesy of the NIMS project [45]) b) Schematic representation of the deployed NIMS. Both mobile and static nodes are shown. Static nodes are arranged in strands ........................................................................................................ 90
5.3 The NIMS Lab System (NIMS-LS). (Courtesy of the NIMS project [45]) a) Schematic of NIMS-LS. b) NIMS-LS main node ........................................... 92
5.4 Different static sensor network topologies and corresponding projections onto the transect ........................................... 94
5.5 Comparison of event OnTime between TA and Raster Scan. Number of events varies between 3 and 20. ................................................. 98

5.6 Comparison of energy consumption in units of time-in-motion (t.i.m.) between TA and Raster Scan. Number of events varies between 3 and 20. ................................................. 99

5.7 PAR data acquired by the first sensor strand during one of the field experiments. Events generated and serviced are shown for Time and Distance policies. Note that events are rendered time of occurrence vs. the PAR value of the event. .......................... 100

5.8 OnTime in form of a zero-mean Gaussian distributions for Time and Distance policies. The OnTime of events generated by all sensors is considered. Dotted (blue or lighter) graphs show the distributions at original mean. .......................... 101

5.9 PAR data acquired by the sensor strands on August 21st 2004, from 10:33 till 20:00. 101

5.10 Estimation of PAR fluctuations with event generation densities by both policies. 103

5.11 Event generation and servicing by both policies. Note that in some cases events are generated at different times and the OnTime of some events varies depending on a policy. .............................................................. 103

5.12 Comparison of an average OnTime and OnTime in a form of a zero-mean Gaussian distributions for Time and Distance policies (p). Three different thresholds (t) used. The OnTime of events generated by all sensors is considered. ................................................. 104

5.13 The change in the average OnTime and the number of serviced events as the number of robots varies between 1 and 5. ................................................. 106
Abstract

We address the problem of monitoring spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment. Our thesis is that a cooperative system comprised of mobile robots (suitable on their own for efficient temporal sampling) and a static sensor network (suitable on its own for efficient spatial sampling) is an effective means of addressing this problem. We provide theoretical as well as empirical evidence supporting this thesis by decomposing the design of the proposed collaborative system into three constituents. These are respectively, robot coverage and exploration through sensor network deployment, sensor network-assisted robot navigation and sensor network-mediated multi-robot task allocation.

The first subproblem we address is the embedding of an active infrastructure (sensing, communication and computation) into the environment using robots, which simultaneously use this infrastructure for coverage and exploration. Our algorithm for this is provably complete, decentralized, scalable, robust, fault tolerant and can be used on simple robots. We present experimental and simulation results which verify the performance of the algorithm. We also present theoretical results which illustrate its asymptotic behavior.

Once a network is deployed it can be used for robot navigation. We present an algorithm that allows robots to navigate precisely and reliably using a deployed sensor network. This network-directed navigation approach can be used by simple (modest computation, communication and computation and communication)
sensing requirements) heterogeneous robots. Extensive empirical testing confirms the validity of our approach.

The final subproblem we address for efficient spatiotemporal monitoring is multi-robot task allocation. The network is used as a spatially diverse 'sensor' for event detection and as a mediator which assigns and navigates robots to high-value sampling locations. This builds on the deployment, coverage, and navigation capabilities presented earlier. Network-mediated task allocation allows robots to 1. respond to tasks they cannot directly sense, 2. communicate at long ranges, 3. efficiently repair and maintain network. We validate our algorithm for task allocation in field and lab experimental settings.
Chapter 1

Introduction

We address the problem of monitoring spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment. Our thesis is that a cooperative system comprised of mobile robots (suitable on their own for efficient temporal sampling) and a static sensor network (suitable on its own for efficient spatial sampling) is an effective means of addressing this problem. We provide theoretical as well as empirical evidence supporting this thesis by decomposing the design of the proposed collaborative system into three constituents. These are respectively, robot coverage and exploration through sensor network deployment, sensor network-assisted robot navigation and sensor network-mediated multi-robot task allocation.

1.1 Motivation

A broad class of environmental monitoring objectives in fundamental science [48], environmental resource management [44], and public health protection [10] demand distributed sensing capabilities [18]. Understanding critical phenomena, for example the nature of carbon flux from the atmosphere to forest biomass, requires direct experimental characterization of spatiotemporally distributed phenomena [48]. This includes measurement of solar radiation (driving fundamental
photosynthesis and growth), atmospheric water vapor, temperature, and chemical composition. Distributed measurements are required since these phenomena are sensitive to (and in turn influence) the heterogeneous structure of the natural environment. Figure 1.1 shows a schematic of an ecosystem, where scientists are interested in measuring time-varying phenomena in the atmosphere, water, soil and forest biomass.

Virtually all environmental monitoring applications require a high fidelity characterization capability for environmental variables. This implies a high spatiotemporal sampling rate. For example, solar radiation and atmospheric properties must be mapped in natural environments over a spatial extent comparable to those of the forest size. At the same time, some phenomena display a characteristic spatial variability on the scale of centimeters. Thus, in considering high sampling
fidelity for these phenomena over a two-dimensional plane with the required spatial extent of over
1000 $m^2$ and resolution of greater than 10 $sample/m^2$ (a modest spatial sampling frequency)
requires an impractically large number of sensing elements (10,000 measurement points).

An example of a fundamental phenomenon that controls processes within natural ecosys-
tems is the flux of solar radiation. Solar radiation is spatially filtered by the complex ecosystem
structure and ultimately controls photosynthesis and growth [50]. The characterization of solar
radiation spatiotemporal patterns is of primary interest to understanding growth, evolution, and
global change trends [48]. Light is measured as Photosynthetically Active Radiation (PAR) which
is defined as radiation in the 400-700 nm waveband. PAR is the general radiation term that covers
both its measurement in terms of photons and the radiant energy in watts [7, 6]. Figure 1.3a-d
shows a temporal change of the light distribution on the forest floor. Figure 1.3e shows the spatial
variance of light distribution, measured in units of PAR.
Figure 1.3: Spatiotemporal variation of light intensity. (Courtesy of the NIMS project [45]) a)-d) Change in solar light intensity at the same location on a forest floor over time. e) Map of solar radiation intensity obtained in a forest ecosystem by a NIMS system (Chapter 5) transporting a light intensity sensor within the canopy. Solar radiation intensity is indicated in contours as it varies spatially according to horizontal and vertical sensor displacement.
In order to study spatiotemporal phenomena one may use either a predeployed static sensor network or a system of sensors which are portable. If an automated solution is desired, portability is achieved by using mobile robots to carry sensors from one location to another. Using a static sensor network alone (Figure 1.2a) leads to measurement distortion; often the result of improper spatial-sampling distribution (specifically due to the mismatch between the spatial structure of the phenomenon and sensor node placement). This introduces further challenges for environmental monitoring, i.e. distributed sensor node deployment planning. Specifically, the inherent unpredictability and variability of environmental structure and phenomena precludes the possibility of achieving adequate spatial sampling density by advance planning. The conventional solution for reducing this source of distortion has been to increase the spatiotemporal sampling rate. However, this results in excessive disturbance to the sensed environment and is economically infeasible since every node in the network has to be equipped with a sensor capable of characterizing the phenomenon of interest. The three key problems of using a static sensor network are as follows:

1. **Deployment planning:** The time-varying nature and unpredictability of the phenomenon of interest as well as the dynamics of the environment makes preplanning the sensor network deployment difficult or impossible.

2. **Emplacement:** Certain environments (e.g. environments with hazardous materials present, etc.), are not easily accessible or safe for manual deployment.

3. **Maintenance:** Many monitoring problems in the natural sciences require long-term, unattended data acquisition. Some sensor nodes can run out of power or be damaged/destroyed; these need to be replenished, repaired or replaced.
Clearly, a fixed sensor network alone, is not adequate for many important environmental monitoring phenomena.

Since environmental dynamics drive unpredictable and variable sensor coverage requirements, sensor node *mobility* may be exploited to ensure adequate coverage. Mobile robots (Figure 1.2b) provide high fidelity spatial sampling. Mobility allows the number of robots (and hence the number of sensors used) to be small. Despite the benefits of a mobile robot-only approach measurement distortion may still occur. Since navigation takes time and robots may not have information about the precise sampling location of a time-varying phenomenon, in the worst case robots may need to carry sensors to sample the whole environment. This could lead to delayed data acquisition and potential loss of data. Moreover, the environment in which the sampling occurs will in general be unknown, unstructured and dynamic. Hence, navigation and localization - capabilities crucial for mobile robots - are difficult or impossible.

We propose a new system that combines *both fixed and mobile* sensor nodes to achieve spatiotemporal environmental coverage that is dramatically advanced over that of either system alone. Mobility allows the networked sensor system to always seek the most efficient spatiotemporal sampling distribution to achieve environmental variable reconstruction. Further, mobility permits the system to respond to initially unpredictable and variable environmental evolution. In our system, the functioning of the mobile nodes critically depends on the static nodes and vice versa. Indeed, the interdependence is so tight that a fitting metaphor for it is Symbiosis.
1.2 Design Principle - Symbiosis

“Symbiosis - the intimate living together of two dissimilar organisms in a mutually beneficial relationship”

- Merriam-Webster Online Dictionary (http://www.webster.com)

This definition highlights a special kind of symbiosis, mutualism, in which each species benefits from the other. Nature provides a variety of examples of mutualism. [56]. One of the most important examples of mutualism is the symbiotic relationship between a Rhizobium (a nitrogen-fixing bacteria) and its legume hosts. Every living organism requires nitrogen-compounds (e.g. proteins and nucleic acids). Even though air is rich in nitrogen gas (79%), most organisms can use only nitrogen incorporated in compounds (i.e. in fixed form). Rhizobium contains the required metabolic machinery to fix nitrogen, whereas the legume host provides nutrients and oxygen. As a result, nitrogen compounds are secured by legumes, and can be further secured by animals that are feeding on these particular legumes. This constitutes the nitrogen cycle [19].
We are inspired by the idea that a system of mobile robots and a network of static sensors could leverage strengths from each other. Motivated by the sampling application described earlier we design algorithms for such a symbiotic system where mobile robots and a sensor network exist in the same physical space and cooperate for mutual benefit.

We treat the sensor network as a distributed medium for:

1. **Sensing.** The network provides a sensor that is 'stretched' over the environment, allowing the robots to sense the environment with high temporal sampling density.

2. **Communication.** The network can and is used for communication among robots.

3. **Computation.** The network is used for distributed processing, which reduces computational complexity of processing onboard the robots.

As a consequence of these three properties the sensor network is able to monitor the environment for tasks and compute assignments of tasks to robots. Mobile robots deploy, repair and maintain the network, while accomplishing tasks that require mobility as directed by the network.

The proposed system is self-contained; it can deploy and maintain the infrastructure it uses. As shown in Figure 1.4, mobile robots and a sensor network form a symbiotic system. In this system, mobile robots deploy and maintain a sensor network, while at the same time use it for distributed sensing, computation and communication.

Thus, much like in nature, the two dissimilar ‘artificial organisms’ - the sensor network and mobile robots - work cooperatively to solve a variety of problems relying on each other. One of the most interesting results of such cooperation is that by using the deployed sensor network as a distributed infrastructure, mobile robots can solve a variety of problems relying purely on local information.
1.3 Problem Description and Thesis Statement

**Problem:** Design a system to monitor spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment.

**Thesis:** A symbiotic system combining the capabilities of mobile robots and a static sensor network efficiently provides the ability to monitor spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment.

Let’s examine the problem at hand in greater detail. We start in an unknown, unstructured, dynamic environment. There is no access to global information (e.g. poor to no GPS, no map). Since the environment is unknown and dynamic it needs to be explored and covered. However, efficient coverage and exploration requires recognition of landmarks so that unexplored areas are dealt with in preference to already visited areas. Natural landmark recognition is impossible or at best unreliable in the target environment. However, we can use nodes of the sensor network as artificial landmarks. One task then is to deploy a sensor network. To avoid a chicken and egg situation we need to solve the coverage and exploration problem *simultaneously* with the sensor network deployment, or at a minimum interleave them in some convenient way. Once the sensor network is deployed it can be used for precise robot navigation and multi-robot task allocation - enablers for the actual application (spatiotemporal sampling) by which we are motivated. Following this reasoning we set forth the three hypotheses that drive the development of our system and establish the thesis:

**Hypothesis 1 [Coverage and Exploration through Sensor Network Deployment]:**

*There exists an efficient, robust and scalable algorithm to embed an active infrastructure*
(sensing, communication and computation) into the environment while simultaneously using this infrastructure for coverage and exploration.

**Hypothesis 2 [Network mediated Robot Navigation]:**

*An embedded sensing and communication network of static nodes can be used to guide the robot navigation.*

**Hypothesis 3 [Network mediated Multi-Robot Task Allocation]:**

*An embedded sensing and communication network of static nodes can efficiently mediate multi-robot task allocation to enable high fidelity spatiotemporal sampling.*

### 1.4 Contributions

In this section we outline the significant contributions of this dissertation.

1. **A novel symbiotic approach enabling an efficient solution to the high fidelity spatiotemporal monitoring problem in an unknown, unstructured, dynamic environment.**

   As discussed in the previous section, an accurate and efficient solution to the spatiotemporal monitoring problem using either a sensor network or a mobile robot system alone may not work. We propose a novel self-contained system combining the strengths and mitigating the weaknesses of mobile robots and a sensor network in a mutually beneficial relationship. We show that such a symbiotic system correctly and efficiently solves the
Figure 1.5: The process of coverage and exploration. a)-c) A robot deploys the sensor network, while exploring the environment. d)-f) A robot detects the gap in the network and repairs it.
stated problem. Further, we decompose this problem into three tractable and flexible sub-problems: coverage and exploration through sensor network deployment, robot navigation and multi-robot task allocation.

2. **An algorithm that solves the coverage and exploration problem through sensor network deployment and repair in unknown, unstructured, dynamic environments.**

We propose an algorithm that is decentralized, scalable, robust, fault tolerant and can be used on simple robots. We examine the trade-offs that should be considered in choosing this algorithm over the others to solve this problem. We empirically verify the performance of our algorithm and develop theoretical results about its asymptotic behavior using analysis on graphs and verification in simulation. There exists a direct correspondence between the results obtained from the theoretical analysis and the data from simulation experiments. We make the case that our methodology (robots embedding a sensor network into the environment) is compatible with other search and coverage algorithms. Figure 1.5 schematically shows the process of coverage and exploration through sensor network deployment. Figure 1.5d-f highlights another requirement - sensor network repair (maintenance).

3. **An algorithm that solves the robot navigation problem using a deployed sensor network.**

We propose an algorithm that allows a robot to navigate precisely and reliably using a deployed sensor network. Our approach differs from systems described in the literature by assuming that a map, localization or GPS are not available. The navigation approach can be used by simple (modest computation, communication and sensing requirements) heterogeneous robots. The navigation occurs through node-wise motion from node to node
Figure 1.6: Using a network for a)-c) Robot navigation. d)-f) Multi-robot task allocation.
on the path from starting node to the goal node. Figure 1.6a-c schematically shows the network being used to aid robot navigation by guiding the robot from node to node till the goal point is reached.

4. **Algorithms that solve the multi-robot task allocation problem using a deployed sensor network.**

In the end, efficient spatiotemporal monitoring requires multi-robot task allocation. We propose algorithms for multi-robot task allocation based on the interaction between a sensor network and mobile robots. Tasks, upon arrival, are allocated to robots by the network. We propose Distributed In-Network Task Allocation (DINTA). DINTA differs from the traditional MRTA in that it relies on a static network, leading to distributed communication, sensing and computation. The utilities of task assignments are propagated and computed by the network based on purely local communication between the network nodes. The system does not require mobile robots to be within communication range of each other. The network is used for propagating messages between the robots. The system does not place a limitation on the number of robots. There is no computation or communication overhead associated with increasing the number of robots. The system does not require one robot to recognize another robot. We also show that such network-mediated multi-robot task allocation algorithms can also be used for efficient network repair and maintenance. As shown in Figure 1.6d-f, the network detects a phenomenon locally and then assigns and navigates robots to inspect it.

This thesis proposes a novel symbiotic system consisting of mobile robots and a sensor network. We show that such a system addresses problems (e.g. spatiotemporal monitoring) that
are impossible or infeasible to solve by traditional means. We also show that our system can efficiently, reliably and robustly solve traditional problems in robotics (e.g. coverage and exploration, navigation and multi-robot task allocation) and sensor networks (e.g. deployment, maintenance and repair). In addition, we demonstrate that to solve a given problem in our system, robots rely on a deployed active infrastructure and as a result do not require global information about the environment (no GPS, map, localization, etc). Hence, the robots can be minimalist or can dedicate their resources to other tasks (e.g. data processing, mission planning, etc.).

1.5 Outline

This dissertation is organized as follows. Chapter 2 presents the design of an efficient, robust and scalable algorithm which validates the first hypothesis - coverage and exploration through sensor network deployment and repair. The theoretical analysis as well as simulation experiments compare performance of our algorithm to the approaches in the literature. Chapter 3 describes our approach that validates the second hypothesis - network mediated robot navigation. We model the navigation problem as a Markov Decision Process. Extensive experimental results on physical platforms are presented. Chapter 4 presents our approaches to validate the final, third, hypothesis (multi-robot task allocation), which also establishes our thesis - efficient spatiotemporal monitoring. The general idea behind the two proposed network-mediated MRTA algorithms is described first, the experimental results are presented next. We also show that network-mediated MRTA algorithms can be used for efficient network repair and maintenance. Finally, Chapter 5 describes our work on Networked Infomechanical Systems that focuses on extending and applying the task
allocation algorithms to a system deployed and running in a natural ecosystem. We present exper-
imental results that show the successful application of our methodologies to the problem of
spatiotemporal monitoring.
Chapter 2

Coverage and Exploration through Sensor Network Deployment

In this Chapter we present the design and analysis of a novel algorithm that validates the first hypothesis:

Hypothesis 1 [Coverage and Exploration through Sensor Network Deployment]:

There exists an efficient, robust and scalable algorithm to embed an active infrastructure (sensing, communication and computation) into the environment while simultaneously using this infrastructure for coverage and exploration.

Hence, our algorithm should be able to perform coverage and exploration of an unknown, unstructured and dynamic environment, while deploying and maintaining an active infrastructure.

The coverage problem has been defined [22] as the maximization of the total area covered by robot’s sensors. The problem of coverage can be considered as a static or more generally as a dynamic problem. The static coverage problem is addressed by algorithms [46, 27, 4] which are designed to deploy robot(s) in a static configuration, such that every point in the environment is under the robots’ sensor shadow (i.e. covered) at every instant of time. Clearly, for complete static coverage of an environment the robot group should be larger than a critical size (depending on environment size, complexity, and robot sensor ranges). Determining the critical number is difficult or impossible if the environment is unknown a priori. Dynamic coverage, on the other
hand, is addressed by algorithms which explore and hence ‘cover’ the environment with constant motion and neither settle to a particular configuration [3], nor necessarily to a particular pattern of traversal.

2.1 Related Work and Assumptions

In this Chapter we consider a single robot in a planar environment whose layout is unknown. The environment is assumed to be large enough, so that complete static coverage of the environment is not possible with one robot. The robot must thus continually move in order to observe all points in the environment frequently. In other words we address the dynamic coverage problem with a single robot.

Exploration, a problem closely related to coverage, has been extensively studied [67, 68]. The frontier-based approach [67] concerns itself with incrementally constructing a global occupancy map of the environment. The map is analyzed to locate the ‘frontiers’ between the free and unknown space. Exploration proceeds in the direction of the closest ‘frontier’. The multi-robot version of the same problem was addressed in [9].

Our algorithm differs from these approaches in a number of ways. We use neither a map, nor localization in a shared frame of reference. Our algorithm is based on the deployment of static, communication-enabled, sensor nodes into the environment by the robot. For purposes of analysis in this Chapter we treat this collection of sensor nodes as the vertices of a graph even though no explicit adjacency lists are maintained at each node. The graph is thus purely an aid to our analysis of the coverage and exploration algorithm, not an entity used by the algorithm itself.
The problem of exploration using passive nodes (read only devices) was considered from the graph theoretic viewpoint in [16, 5]. In both cases the authors studied the problem of dynamic single robot coverage on a graph world. The key result was that the ability to tag a limited number of vertices (in some cases only one vertex) with unique passive nodes dramatically improved the cover time. We note that [16, 5] consider the coverage problem, but in the process also create a topological map of the graph being explored. [16, 5] also show that in certain environments exploration is impossible without tagging. There are four key differences between our algorithm and the work reported in [16, 5]:

1. We do not assume the robot can navigate from one node to another in any reliable fashion.
   The robot does not localize itself, nor has a map of the environment (the structure of the graph corresponding to the environment is not known to the robot, nor does it construct it on the fly).

2. We assume the number of sensor nodes available for drop-off is unlimited; in [16, 5] a limited number of nodes is used.

3. We assume that each node being dropped off is capable of simple computation and communication - the nodes are active; in [16, 5], the nodes are passive - they neither compute nor communicate.

4. We do not assume that nodes need to be retrieved; in [16, 5] retrieval and reuse of nodes by the robot is implied.

Our work is closely related to the ant robots literature [59, 60, 33, 55, 63, 62, 64] where the idea of a node with decaying intensity (a semi-active node) is used. The robots sense the
Algorithm 1 Least Recently Visited (LRV) Algorithm

Robot Loop:
if no sensor node within communication range threshold then
    deploy sensor node
else
    Move in direction suggested by nearest sensor node
    Update sensor nodes if necessary

Sensor Node Loop:
Emit least recently visited direction = ANY_OF(argmin_{d\in D(i)} W(d))
Update sensor nodes weight if necessary: W(i) := W(i) + 1

change in intensity and are able to change the direction of exploration to cover environment efficiently. Our algorithm differs from these approaches - we assume that each deployed node is capable of sensing, simple computation and communication. We exploit the computation and communication capabilities of the nodes to address problems beyond coverage and exploration.

The nodes we use, act as a support infrastructure which the mobile robot uses to solve the coverage problem efficiently. The robot explores the environment, and based on certain local criteria, drops a node into the environment, from time to time. Each node is equipped with sensing, a small processor and a radio of limited range. The ensemble of nodes forms a sensor network. Our algorithm performs the coverage task successfully using only local sensing and local interactions between the robot and the sensor network.

The problem of coverage and deployment in Sensor Network community was considered from a different perspective. For example, [41] considers quality of service of the deployed network, [40] discusses algorithms to achieve low energy deployment. Collaborative target tracking and surveillance is considered in [12, 15].
2.2 LRV: Least Recently Visited Algorithm.

In this section we present the Least Recently Visited (LRV) algorithm for sensor network deployment and maintenance, coverage and exploration. As shown in Algorithm 1, LRV is the concurrent execution of two algorithms - one on a robot (Robot Loop) and another on every node (Sensor Node Loop). For every node $i$, let $D(i)$ be the set of directions possible to traverse from $i$. Then $\forall d \in D(i), W(i, d)$ is the weight (cached on a node) maintaining the number of times $d$ was traversed from $i$. In some cases, we will refer to the weight of direction $d$ as $W(d)$ if the node id is implicit. The function $\text{ANY\_OF}(T)$ returns a single element of a set $T$ according to some arbitrary rule (i.e. in order, random, etc).

When a robot is deployed into the environment initially, according to Algorithm 1, it deploys a node because there is no sensor node within communication range. Over time LRV causes a network of nodes to be deployed since every time a new node is deployed it must be able to communicate with at least one other sensor node in the network. As will be shown in section 2.7, maintaining network connectivity is the minimal requirement that the deployment function should have.

Once deployed, each sensor node starts to emit the locally least recently visited direction (hence the name LRV), which is one of the directions with smallest weight $W$ (if there are multiple directions of the same weight, one is arbitrary picked). In practice the number of directions per node is often bounded and application dependent. In our experimental work we set this bound to 4.

Another important aspect of the algorithm is weight $W$ update. The weight of a direction is incremented in two cases: right before a direction is traversed and on the destination node right
after a direction is traversed. Suppose the robot is in the vicinity of a node $i$ and is suggested to move in direction $d$. The weight $W(i, d)$ is incremented right before direction $d$ is traversed.

Suppose the robot enters node $i$ through direction $d$. The weight $W(i, d)$ is incremented right after direction $d$ is traversed.

### 2.3 The Graph Model

For purpose of analysis, consider an open bounded environment with no obstacles. In this case, given our node deployment algorithm (LRV) described in the previous section, we can model the steady state spatial configuration of the nodes as a finite graph $G = (V, E)$, where $V$ is a set of vertices (the deployed nodes) and $E$ is a set of edges such that $\forall i, j \in V$ there is an edge between $i$ and $j$ iff 1. $i$ and $j$ are within communication range; 2. there is a physical path between $i$ and $j$. Consider the schematic of the environment in Figure 2.1a. We represent the LRV-deployed
network in this environment as a graph $G = (V, E)$ (shown in Figure 2.1c). A graph model is a natural choice because of its flexibility and ubiquity of usage in such problems.

Before discussing the theoretical properties of LRV we provide working definitions for coverage and exploration on graphs and corresponding performance metrics.

**Definition (Coverage on a graph) 2.1:** Coverage on a graph is the act of visiting every vertex of a graph.

The performance of a coverage algorithm is measured using the *cover time* [37] metric defined as follows:

**Definition (Cover time) 2.2:** Cover time is measured in terms of the number of edges traversed such that every vertex of a graph is visited at least once, i.e. the graph is covered.

Note that in order to cover a graph, a robot needs to at least traverse one edge per node (consider a spanning tree of a graph). This notion is distinct from graph exploration or ‘complete’ graph coverage (where the robot needs to traverse every edge of a graph). This later notion is called graph exploration defined as follows:

**Definition (Exploration on a graph) 2.3:** Exploration on a graph is the act of traversing every edge of the graph.

An exploration algorithm is evaluated using the *exploration time* metric defined as follows:

**Definition (Exploration time) 2.4:** Exploration time is measured in terms of the number of edges traversed such that every edge is traversed at least once.

It follows from the above definitions that exploration is a superset of coverage. Therefore, 

$\text{Cover Time} = O(\text{Exploration Time})$. 

23
Algorithm 2 Least Recently Visited (LRV) Algorithm on Graph.

\[
\text{if Covered/Explored the graph then} \\
\text{Exit} \\
\text{else} \\
\text{\hspace{1em} next\_node = ANY\_OF(\arg\min_{ij \in E(\text{current\_node})} W(\text{current\_node}, j))} \\
W(\text{current\_node}, \text{next\_node}) := W(\text{current\_node}, \text{next\_node}) + 1 \\
\text{current\_node := next\_node}
\]

Figure 2.2: Illustration for Theorem 1. At every point in time during the execution of LRV a graph \( G = G_{\text{exp}} + G_{\text{u}} \), where \( G_{\text{exp}} \) is the explored part of \( G \) and \( G_{\text{u}} \) is an unexplored part. An edge \( e_{ij} \) connects \( G_{\text{exp}} \) and \( G_{\text{u}} \).

2.3.1 LRV on Graphs

Given the graph model we exhibit two important properties of LRV. First, we show that LRV is complete, and second we establish a relationship between its cover time and exploration time.

For purposes of this analysis we are interested in the behavior of LRV in the ‘steady-state’ when all nodes have been deployed. In this special case one can consider a simple version of LRV on a graph as follows. For every vertex \( i \), \( E(i) \) is the set of edges incident to \( i \). For clarity we identify an edge in \( E(i) \) with the node this edge connects node \( i \) to. Then \( \forall e \in E(i) : W(i, e) \) is the weight (cached on node \( i \)) maintaining the number of times edge \( (i, e) \) was traversed from \( i \).
In some cases, we will refer to the weight of edge \( e \) as \( W(e) \) if the originating node is implicit.

Note that in LRV the weight of an edge is incremented twice: before and after traversal, but on different nodes. This redundancy is required for practical purposes: the weights are cached on nodes and since the environment is dynamic, sensing and actuation are noisy, starting at the same node and traversing the same direction at different points in time does not guarantee that robot would arrive at the same node. In the graph model we study the steady state spatial configuration of the nodes on a finite unchanging graph. Hence, for clarity of presentation, \( \forall i, j \in V \) consider storing the weight associated with \( i \to j \) transition on the edge \( e_{i,j} \in E \). This weight is identical to the one associated with \( j \to i \) transition (e.g. \( W(i, j) = W(j, i) \)). We increment the weight just in one case: right before an edge is traversed, and associate it with the edge \( e_{i,j} \in E \).

Algorithm 2 shows this simplified LRV on a graph. Note that the deployment function is removed since we are in the steady state.

We now state and prove two results: Completeness of LRV on finite graphs and the relationship of cover time to exploration time.

**Theorem 1 (Completeness):** The exploration time of LRV on a finite graph is finite.

**Proof:** The goal is to show that LRV traverses every edge of any finite graph in finite time. The proof is by contradiction. Suppose the exploration time of LRV is infinite. Therefore, there is a time \( t \) after which LRV traverses only those edges that it traverses infinitely many times (edges of the graph \( G_{exp} \) in Figure 2.2). Weights of these edges grow without bound, including the edge that is considered for traversal infinitely many times but is never picked after time \( t \) (edge \( e_{i,j} \) in Figure 2.2). By definition, LRV will be forced to traverse this edge after time \( t \), which is a contradiction.

**Theorem 2:** For a graph \( G=(V,E) \) with maximum degree \( d \), if Cover Time = \( O(f(V)) \), then Exploration Time = \( d \times O(f(V)) \).

**Proof:** Suppose LRV executes on a graph \( G \) until every vertex is visited at least once. It is obvious that at least one edge per vertex is traversed. Thus, after the first execution of
the algorithm, the number of untraversed edges at every vertex is at most $d - 1$. Note, that at a given vertex, while there are untraversed edges, LRV will choose one arbitrarily. Hence, after at most $d$ executions of the algorithm every vertex would be covered and every edge would be traversed. Thus, if $\text{Cover Time} = O(f(V))$, then $\text{Exploration Time} = d * O(f(V))$.

### 2.3.2 LRV on a Square Lattice: Empirical Results from Simulation

In this section we consider the performance of LRV on the following special graph $G$:

1. $G$ is undirected.

2. $G$ has degree $\deg_G \leq 4$. If all nodes have degree 4, then $G$ is a square lattice i.e. a regular graph of degree 4.

3. $|V| = \Theta(|E|)$.

We consider this special graph because in practical implementations of LRV, a physical compass on the sensor node determines direction. If this compass has $k$ bits of resolution, then each node is capable of identifying $2^k$ directions resulting in a graph of degree $\leq 2^k$. As we shall see in Section 2.5, our experiments were all done with $k = 2$, resulting in square lattice-like deployments. Hence we analyze LRV on a square lattice. It has been shown [37] that the cover time of a random walk (RW) on a regular graph with $V$ vertices is bounded below by $V \ln V$ and above by $2V^2$. If we assume that passive nodes can be used, and the graph $G = (V, E)$ is known (a topological map is available) and the robot can drop nodes of three independent colors, then the problem of coverage can be solved optimally by applying Depth-First Search (DFS) which is linear in $V$. DFS assumes that all resources are available - nodes, map, localization and perfect navigation.
(a) Comparison of Cover Time for DFS, RW and LRV. The $n \ln n$ curve is shown for reference.

(b) A comparison between DFS and LRV. This graph is a magnified view of (a)

Figure 2.3: Comparison of graph coverage algorithms. The $n \ln n$ curve is shown for reference.
In [16] the problem of coverage is considered in the context of mapping a graph-like environment with $V$ vertices. Their algorithm explores the environment and constructs a topological map on the fly. The assumptions of the algorithm are that the robot has $k (k < V)$ nodes, and perfect localization and navigation within the graph. The cover time of their algorithm is bounded by $O(V^2)$. It is important to note that the problem addressed in [16] is more complex than simple coverage, since they build a map while exploring.

We have conducted simulation experiments running RW, DFS and LRV on graphs with $V \in [100^2, 1000^2]$ nodes. For every experiment the steady state (or recurrent\(^1\)) cover time is reported. The results of the experiments are shown in Figure 2.3. The figure also shows the $n \log n$ curve for reference. These experiments lead us to

**Conjecture 1:** The asymptotic cover time of LRV is $O(V^{1+\epsilon})$.

From Figure 2.3 it is clear that cover time of LRV is less than $V \ln V$, however the bound is not tight. Assuming that the function representing LRV is monotonic we can analyze the sequence $\frac{LRV(i)}{f(i)}$ on the subject of increase (asymptotically $LRV(i) = \Omega(f(i))$), decrease (asymptotically $LRV(i) = O(f(i))$) or does not change (asymptotically $LRV(i) = \Theta(f(i))$). The following is the sequence for $i \in [100^2, 1500^2]$ nodes and $f(i) = i^{1.0005}$.

\[
\frac{LRV(i)}{f(i)} = [3.9406, 3.9587, 3.9639, 3.9672, 3.9682, 3.9684, 3.9685, 3.9685, \ldots, 3.9682] 
\]

Note that the sequence increases initially, then stabilizes at value $3.9685$ for $i \in [900^2, 1000^2]$ and finally decreases. It follows that in this example $\epsilon \leq 5 \times 10^{-4}$ and asymptotic cover time is $O(V^{1.0005})$. Hence, the asymptotic cover time of LRV is $O(V^{1+\epsilon})$.

Using Theorem 2 and Conjecture 1, we get the following result for a square lattice.

**Corollary 1:** LRV explores the environment in asymptotic time $O(V^{1+\epsilon})$.

\(^1\)Steady state or recurrent cover time is taken when the cover time does not change significantly from coverage to coverage.
Lets consider different tradeoffs between the above mentioned techniques. As mentioned earlier, the clear performance boundaries for the coverage task are given by RW (upper) and DFS (lower). The more interesting comparisons are between LRV and DFS and our algorithm and an algorithm with a limited number of passive node markers [16].

Figure 2.3b shows that the asymptotic performance of our algorithm is similar to DFS. Note that in order to determine the identity of neighboring vertices and to navigate perfectly from node to node, DFS assumes that a map of the environment is available and that the robot is perfectly localized. Our algorithm, on the other hand, does not have access to global information and the robot does not localize itself. The nodes used in our algorithm are more complicated than those used in DFS, and the cover times are asymptotically somewhat larger than the cover times of DFS.

In [16] the algorithm builds a topological map of the environment and assumes perfect navigation (and thus, localization) on the graph. The node markers are very simple (the only function is to mark the vertex) and the robot cannot differentiate between them. In addition the algorithm assumes that there exists a local enumeration of edges. The cover time of this algorithm, however, is bounded by $O(V^2)$. Our algorithm, on the other hand, does not have a map and the robot does not localize itself. Another important difference is that we assume that the number of nodes available to us is equal to the number of vertices. In addition, the nodes used in our algorithm are more complex, since they keep a certain state per direction, and are uniquely identifiable. The cover time of our algorithm, however, is conjectured to be less than $V \log V$. Thus, the apparent trade off is using a large number of "smart" nodes (and no global information or localization) vs. a limited number of simple nodes (with mapping and partial localization within the graph).
Algorithm 3 1-LRTA* Algorithm on a graph.

if Covered/Explored the graph then
    Exit
else
    next node = ANY_OF(argmin \( j \in E(\text{current node}) \) \( W(j) \))
    \( W(\text{current node}) := W(\text{next node}) + 1 \)
    current node := next node

---

The cover time achieved by our algorithm is clearly better. However, if the nodes are a precious resource, the algorithm described in [16] would be preferred.

Another algorithm we compare LRV to is 1-LRTA* [32]. 1-LRTA* is a well known graph search algorithm that can be applied to graph coverage. Algorithm 3 shows the details of 1-LRTA*. In 1-LRTA*, a weight is associated with a node. The edge to traverse is chosen based on weights of neighboring nodes. The weight of a node is incremented with the weight of a node the robot transitions to. Hence, 1-LRTA* requires nodes to communicate.
Figure 2.4 shows that generally 1-LRTA* outperforms LRV. However, it should be noted that LRV is a deployment and exploration algorithm, whereas 1-LRTA* is a graph exploration algorithm which assumes the graph is given.

Finally, let’s examine how fast LRV, 1-LTRA* and RW converge to a full coverage. Figure 2.5 shows a comparison of the convergence speed for LRV, 1-LRTA* and RW on a 1000x1000 regular square lattice. In other words, Figure 2.5 shows the percentage of the whole graph covered as the algorithms progress (i.e. visit more and more nodes or as more actions are taken). Figure 2.5a shows a linear convergence of LRV and 1-LRTA*, whereas, RW on the interval of 50-100% of complete coverage (Figure 2.5b) exhibits a nonlinear increase in running time. The curve for DFS is not included, since the convergence factor is optimal and the cover time is at most twice the number of nodes.
Figure 2.6: a) A map of the environment with an embedded sensor network with a tree-like topology. b) Tree isomorphic to embedded sensor network topology of a).
Figure 2.7: Illustration for Lemma 1. A tree $T$ at a point of time when a robot enters vertex $v_i$ through an incoming edge $e_{ij}$.
2.3.3 LRV on a Tree

In this section we study the performance of LRV on trees. Figure 2.6a shows a map of the environment with an embedded sensor network. The sensor network has a tree-like topology. Figure 2.6b shows the tree which represents the embedding. A tree differs from square lattice in two major ways: 1. The vertex degree is not bounded by 4; 2. A tree does not contain cycles. The next two Lemmas establish local properties of LRV needed for the main result of this section: performance of LRV on trees is linear or asymptotically optimal.

**Lemma 1:** An incoming edge $e_{ij}$ is traversed twice iff every other edge incident to $v_i$ is traversed twice.

*Proof:* Initially the weights of all edges are zero. Suppose a robot enters vertex $v_i$ through an incoming edge $e_{ij}$ (refer to Figure 2.7). The weight of $e_{ij}$ is incremented and equal 1, whereas the weights of other edges incident to $v_i$ are 0.
Figure 2.9: Illustration for Theorem 3. A tree $T$ augmented with a vertex $v'$ and an edge $e_{v'v}$ connecting $v'$ to vertex $v \in T$.

Next, LRV picks one of the 0-weighted edges, say $e_{ik}$, and traverses it. The weight of $e_{ik}$ is incremented and equal 1, the weight of $e_{ij}$ is equal 1 and the weights of other edges incident to $v_i$ are 0. Due to Completeness Theorem, eventually robot returns back to $v_i$ by traversing an edge $e_{ik}$. The weight of $e_{ik}$ is incremented and equals 2, the weight of $e_{ij}$ is equal to 1 and the weights of other edges incident to $v_i$ are 0.

Apply the same reasoning to every other 0-weighted edge incident to $v_i$. The weight of $e_{ij}$ is equal 1, whereas the weights of other edges incident to $v_i$ are 2. At this point LRV is forced to pick $e_{ij}$ as the only edge of minimum weight incident to $v_i$. Hence, an incoming edge $e_{ij}$ is traversed twice iff every other edge incident to $v_i$ is traversed twice.

It follows from Lemma 1 that if before traversing an edge $e_{ij}$, the weights of all edges incident to $v_i$ are equal (initially all 0), then after an incoming edge $e_{ij}$ is traversed twice the weights of all edges incident to $v_i$ are equal and incremented by two.

**Lemma 2:** An incoming edge $e_{ij}$ is traversed twice iff in a subtree $T' = T - (T_{ij} + e_{ij})$ every edge is traversed twice.

**Proof:** Consider a subtree $T'$ (refer to Figure 2.8). LRV starts at vertex $v_i$. Applying Lemma 1 to $v_i$ results in every edge incident to $v_i$ traversed twice. Applying Lemma 1 recursively to every vertex of $T'$ results in every incident to every vertex edge traversed.
twice. Hence, an incoming edge $e_{ij}$ is traversed twice iff in a subtree $T' = T - (T_{ij} + e_{ij})$ every edge is traversed twice.

Using the results of Lemma 1 and 2 we can prove the main result of this section, stated as Theorem 3.

**Theorem 3:** The exploration time of LRV on a tree is no more than $2|E|$.

**Proof:** Consider a tree $T$ (refer to Figure 2.9). Augment $T$ with a vertex $v'$ and an edge $e_{v'v}$ connecting $v'$ to vertex $v \in T$. Consider LRV on the augmented tree starting at $v'$. It follows from Lemma 2 that the robot executing LRV would traverse $e_{v'v}$ twice when in tree $T$ every edge is traversed twice. Hence, the exploration time of LRV on a tree is no more than $2|E|$.

Theorem 3 asserts that the performance of LRV on trees is linear or asymptotically optimal.

### 2.4 LRV Implementation Details

In this section we describe implementation details associated with LRV to obtain a system that functions on real hardware and in simulation. Loosely put, in LRV, the task of each node is to
recommend a locally preferred direction of movement for the robot within its communication range. Thus each node acts as a local signpost telling the robot which direction to explore next. In practice, the robot treats this information as a recommendation, and combines it with local range sensing (to avoid obstacles) to make a decision about which direction to actually pursue.

As shown in Figure 2.10(b), each node has a state associated with four cardinal directions (South, East, North, West). The choice of four directions is arbitrary. It implies that each node is equipped with a 2 bit compass. For each direction, the node maintains a binary state ($T$), a counter ($C$) and block $E$ which might be used for additional information. The state $T$ can be either OPEN or EXPLORED, signifying whether the particular direction was explored by the robot previously. The counter $C$ associated with each direction stores the time since that particular direction was last explored.

When deployed, a node emits two data packets with different signal strengths. The packet with the lower signal strength is called the $MIN$-packet and the one with the higher signal strength is called the $MAX$-packet. The $MAX$-packet is used for data propagation within the deployed network. The $MIN$-packet contains information about the suggested direction the robot should take for coverage/exploration. This implies that the robot’s compass and the node’s compass agree locally on their measurement of direction. Given the coarse coding of direction we have chosen, this is not a problem in realistic settings. The policy used by the nodes to compute the suggested direction for exploration/coverage to recommend the least recently visited directions preferentially. All OPEN directions are recommended first (in order from South to West), followed by the EXPLORED directions with least last update value (least value of C). Note that this algorithm does not use inter-node communication.
The robot is programmed using a behavior-based approach [39] with arbitration [51] for behavior coordination. Priorities are assigned to every behavior a priori. As shown in Figure 2.10(a), the robot executes four behaviors: ObstacleAvoidance, AtBeacon, DeployBeacon and SearchBeacon. In addition to priority, every behavior has an activation level, which decides, given the sensory input, whether the behavior should be in an active or passive state (1 or 0 respectively). Each behavior computes the product of its activation level and corresponding priority and sends the result to the Controller, which picks the maximum value, and assigns the corresponding behavior to command the Motor Controller for the next command cycle.

During motion, the robot maintains the notion of a current node (Figure 2.11a). This is the node whose MIN-packets are received by the robot most frequently. When the robot moves to the vicinity of a new node, the AtBeacon behavior is triggered and the robot’s current node is updated (Figure 2.11b). AtBeacon analyzes the MIN-packets received from the current node and orients the robot along the suggested direction contained in those packets. In addition, the robot sends
an update message to the node telling it to mark the direction from which the robot approached it as EXPLORED. This ensures that the direction of recent approach will not be recommended soon. We term this the last-neighbor-update. After the robot has been oriented in a new direction, it checks its range sensor for obstacles. If the scan does not return any obstacles, the robot proceeds in the suggested direction (Figure 2.11c), while sending a message to its current node updating the state of the suggested direction to EXPLORED (the node also resets the corresponding C value).

If, however, the suggested direction is obstructed, the AtBeacon behavior updates the node with this information and requests a new suggested direction (Figure 2.11d). The Obstacle Avoidance behavior is triggered if an obstacle is detected in front of the robot, in which case an avoidance maneuver takes place.

Once the robot is oriented in a new direction (whether as a result of taking the advice of the node, or as a result of avoiding an obstacle), the SearchBeacon behavior is triggered. SearchBeacon causes the robot to travel a predetermined distance without a change in heading (assuming there are no obstacles in the way). The DeployBeacon behavior is triggered if the robot does not receive a MIN-packet from any node after a certain timeout value. In this case the robot deploys a new node into the environment.

2.5 Simulation Experiments

In our experiments we used the Player/Stage [26, 61] simulation engine populated with a simulated Pioneer 2DX mobile robot equipped with two 180° field-of-view planar laser range finders positioned back-to-back (equivalent to a 2D omnidirectional laser range finder), wireless communication and a set of nodes. The implementation of the algorithm in simulation proceeds as
Figure 2.12: Simulation results for different environment sizes across 50 trials. Our algorithm consistently outperforms random walk by an order of magnitude, and is significantly more stable.
Figure 2.13: Average cover times for three different grid sizes in simulation. Environment sizes are $25m^2$, $49m^2$ and $100m^2$.

follows. The robot explores the environment in a direction suggested by the last heard node until it does not receive a message from any of the nodes. In this case a new node is deployed into the environment. The robot continues to explore as long as there are OPEN directions left. If all regions are EXPLORED, then the robot picks the direction which was least recently explored. As discussed in Section 2.4, decisions about which direction to explore next are made by the nodes. The robots, however, may alter those decisions if sensor measurements of obstacles are made. In addition, the last-neighbor-update rule prevents the robot from going back in the direction from which it came recently. Values for communication radius and range of the laser were set to 1500mm in the simulations. Figure 2.12 shows the cover times for the random walk algorithm.
and LRV on environments of different sizes: $25m^2$, $49m^2$ and $100m^2$. For every grid size 50 experiments were conducted for both algorithms. The experiments show that our algorithm outperforms random walk and is more stable. In addition, Figure 2.13 shows the average cover times for three different grid sizes. Note the direct correspondence between the results obtained in the graph world and the results of the simulation.

2.6 Implicit Sensor Network Repair and Maintenance

An emergent property of LRV is the ability to perform network repair and maintenance. Since the algorithm is shown to be complete, it is guaranteed to visit the same node over and over again. Suppose that one of the nodes, say node $k$, ran out of power or was damaged. Further consider a moment in time just before the robot traverses direction $d$ towards the damaged node. Now, the robot is moving along direction $d$ towards node $k$. According to the deployment function that is used, there should be a communication/sensing gap in the deployed sensor network (unless the network was overdeployed and does not require repair). Hence while facing the same deployment situation and using the same deployment function at the location where node $k$ was deployed the robot simply deploys a new node, thereby solving the problem of sensor network repair and maintenance implicitly. Note that if the robot can recognize the nodes then it can attempt repairing the node first (or retrieving for later repair at the base) before deploying the new node.

2.7 Remarks on Generalization

In this Chapter we have presented an algorithm based on the policy of choosing the least recently visited directions preferentially. At the same time depending on the application requirements
other policies might be better. It is possible to extend other search/coverage algorithms using physical network embedding to function in unknown, unstructured, dynamic environments. The basic idea is to augment a coverage algorithm with an ability to deploy and maintain physical network infrastructure, while simultaneously using the network to aid in coverage and exploration. Thus what we want to highlight is the underlying philosophy of network deployment as an integral part of coverage and exploration.

Sensor network deployment is one of the building blocks of this work. A robot decides when to deploy a node based on a deployment condition or more generally deployment function. A deployment function should be designed based on the application. However, there are two main characteristics that every deployment function should have: 1. The deployed sensor network is connected (i.e. between any two nodes there should be a communication path); 2. Deployed static network configuration should be such that static coverage is maximized. In the proposed implementation of LRV both characteristics are captured. The deployment is based on communication range thresholding. Hence, the two consecutive nodes are guaranteed to be connected (thus, the deployed sensor network is connected) and we adjusted communication range threshold so that if every sensor node is equipped with high fidelity sensor, static coverage is maximized.

Other deployment functions could also encode such parameters as desired topology (i.e. by specifying the number of nodes deployed in different directions, etc), desired boundary, phenomenon density (deploy more nodes in places with high phenomenon density), deploy where landmarks detected, etc.
Figure 2.14: Deployment of nodes in a representative simulation trial. Note that due to noise added in simulation, the deployed nodes do not form a perfectly square lattice.
2.8 Summary

In this Chapter we have presented an algorithm that validates the first hypothesis:

**Hypothesis 1 [Coverage and Exploration through Sensor Network Deployment]:**

*There exists an efficient, robust and scalable algorithm to embed an active infrastructure (sensing, communication and computation) into the environment while simultaneously using this infrastructure for coverage and exploration.*

The algorithm (LRV) is based on visiting the least recently visited directions preferentially and is decentralized, scalable, robust, fault tolerant and can be used on simple robots. LRV is based on the idea of deploying sensor nodes from time to time. Once deployed, every node acts like a signpost recording which directions the robot have explored recently. When a robot is in the vicinity of a node, it recommends to the robot a direction that has been least recently visited (hence, the name LRV).

We analyzed the characteristics of LRV theoretically, modeling the static steady state of the deployed sensor network as a finite graph G. We proved that LRV is complete on G (i.e. the exploration time of LRV on a finite graph is finite). For a graph G=(V,E) with maximum degree d, if Cover Time = O(f(V)), then Exploration Time = d*O(f(V)). We proved that Exploration Time is \( \leq 2|E| \) (twice the number of edges, or asymptotically optimal) for the special case, when G is a tree. For another special case, when G is a square lattice, we empirically conjectured that both cover and exploration times are asymptotically \( O(V^{1+\epsilon}) \). The special case of a square lattice is also interesting from practical perspective, because in our LRV implementation and experiments we chose to maintain at most 4 directions, which results in a static steady state of the deployed sensor network resembling a square lattice.
We examined the tradeoffs that should be considered in choosing one exploration algorithm over another to solve this problem. The bounds for the coverage task are given by random walk (the robot has no information and explores randomly) and depth first search (a map of the environment is available in the form of a graph) which solves the problem optimally.

The data shown in Figure 2.3, suggest strongly that our algorithm asymptotically outperforms the \( k \) node algorithm presented in [16].

In addition, it is shown in [16] that if the number \( k \) of available nodes reduces, the cover \( \text{time} \) increases rapidly. Therefore, in dynamic environments the performance of the algorithm decreases drastically even if one node is destroyed. Whereas in our algorithm such a problem does not exist, since a new node will be deployed in place of the destroyed one automatically.

We compared LRV to 1-LRTA* [32]. 1-LRTA* is a well known graph search algorithm that can be applied to graph coverage. In 1-LRTA*, a weight is associated with a node. The edge to traverse is chosen based on weights of neighboring nodes. The weight of a node is incremented with the weight of a node the robot transitions to. Hence, 1-LRTA* requires nodes to communicate. Figure 2.4 shows that generally 1-LRTA* outperforms LRV. However, in reality LRV deploys the network in addition to exploring, whereas 1-LRTA* requires the graph to operate on. An important result of this Chapter is that it is possible to extend other search/coverage algorithms using physical network embedding to function in unknown, unstructured, dynamic environments.

We verified the performance of LRV and its asymptotic behavior in simulation. There exists a direct correspondence between the results obtained from the theoretical analysis (coverage on the graph) and the data from simulation experiments. Note also, that even though the lattice grid was considered as a graph environment for the theoretical analysis, in practice the network
of deployed nodes is not required to be a perfect grid. Figure 2.14 shows a series of screen shots taken from one of the trials of the simulation in the $49m^2$ environment. Note also that the performance of our algorithm is not affected, since it does not rely on localization or mapping.

The theoretical analysis on graphs and verification in simulation shows that tradeoffs in the assumptions can affect *cover time* significantly. Simple algorithms like RW or DFS can be used for coverage, but only in the extreme cases as described above. In case, where mapping and localization are not available, but the number of available nodes is unlimited, our algorithm appears to outperform others.

In the following Chapter we show how the deployed network can direct purposeful robot navigation.
Chapter 3

Mobile Robot Navigation using a Sensor Network

In this Chapter we propose an algorithm to validate the second hypothesis:

Hypothesis 2 [Network mediated Robot Navigation]:
An embedded sensing and communication network of static nodes can be used to guide the robot navigation.

Once the sensor network is deployed and maintained (Chapter 2), it can be used for a fundamental capability of mobile robots - navigation. The local navigation problem deals with navigation on the scale of a few meters, where the main problem is obstacle avoidance. A well-known solution to this problem is presented in [57, 58], where an occupancy grid map of the immediate surroundings of the robot is created and used to determine the navigation direction such that the robot is safely guided towards a goal. Since the map is local, and resembles a ‘sliding window’, mapping of the whole environment does not occur.

3.1 Related Work

The global navigation problem deals with navigation on a larger scale in which the robot cannot observe the goal state from its initial position. A number of solutions have been proposed in
the literature to address this problem. Most rely either on navigating using a pre-specified map or constructing a map on the fly. Other approaches also rely on some technique of localization. Some work on robot navigation is landmark-based relying on topological maps [38, 34, 43, 42], which have a compact representation of the environment and do not depend on geometric accuracy. The downside of such approaches is that they suffer from sensors being noisy and the problem of sensor antialiasing (i.e. distinguishing between similar landmarks). Metric approaches to localization based on Kalman filtering [1] provide precision, however the representation itself is unimodal and hence cannot recover from a lost situation (misidentified features or states). Some of the other approaches aiming to resolve localization ambiguities are multiple-hypothesis Kalman filtering based on a mixture of Gaussians [13], using intervals for representing uncertainties [2, 35], and by deterministic triangulation [54], etc. Approaches developed in recent years based on ‘Markov localization’ [20] provide both accuracy and multimodality to represent probabilistic distributions of different kinds, but require significant processing power for update and hence are impractical for large environments. One of the attempts to solve this problem is presented in [28, 14] where a sampling-based technique is used. Rather than storing and updating a complex probability distribution, a number of samples are drawn from it. The other approaches utilize partially observable Markov decision process (POMDP) models to approximate distance information given a topological map, sensor and actuator characteristics [53]. POMDP models for robotic navigation provide reliable performance, but fail in certain environments (e.g symmetric) or suffer from large state spaces (i.e. state explosion).

An ant-like trail laying algorithm is presented in [60], where ‘virtual’ trails are formed by a group of robots. Navigation is accomplished through trail following. The shortcoming of the algorithm is that it is dependent on perfect communication between the members of the group. In
addition, the ‘virtual’ trails are shared between the robots, which means redundant sharing of the state space in the group. Moreover, a common localization space is assumed.

These approaches have different advantages, but also disadvantages or fail cases. Note that all of the above approaches assume that a map of the environment (topological and/or metric) is given a priori or can be built on the fly. Considering the characteristics of our environment (unknown, unstructured and dynamic) none of the above approaches can be used. We now discuss an algorithm which allows the robot to use deployed network for navigation.

### 3.2 Probabilistic Navigation

In order for the robot to be able to navigate through the environment from location $A$ to location $B$, assuming neither a map nor GPS are available, the robot should be able to choose an action...
that maximizes its chances of getting to its goal, to be able to measure progress and recognize
that it has arrived at the goal. Our approach consists of two stages.

3.2.1 Computing Transition Probabilities

During its exploration of the environment discussed in the previous chapter, the robot builds a
transition graph. Note that this graph is distinct from the graph used in Chapter 2 to model the
network for purposes of analysis. We call this deployed network graph. The vertices of the graph
represent the deployed nodes. A directed edge from vertex A to B is labeled with the probability
of arriving at node B from node A by proceeding in a particular direction. An important point is
that the deployed network graph is stored distributively in the sensor network. That is, the robot
caches transition probabilities associated with node $i$ on node $i$ itself. Therefore, robots that do
not have the knowledge of the transition graph or the ability to obtain this graph can still use it,
by locally communicating with nodes in the network.

The process of determining the transition probabilities is as follows. We assume a finite set
of vertices $S$ in the network graph and a finite set of actions $A$ the robot can take at each node.
Given a subset of actions $A(s) \subseteq A$, for every two vertices in the deployed network graph $s, s' \in S$ and an action $a \in A(s)$ the robot computes the transition probability $P(s'|s, a)$. This is
the probability of arriving at vertex $s'$ given that the robot started at vertex $s$ and commanded an
action $a$. In our implementation four actions are possible at every vertex - East, West, South and
North. Thus, for every action $a_i$ at a given vertex $s \in S$ and all other vertices $s' \in S - s$ the
robot computes the probability $P(s'|s, a_i)$ as the ratio of the number of transitions from $s$ to $s'$
with action $a_i$ to the number of times $a_i$ was commanded at vertex $s$. This ratio is normalized
to ensure that $\sum_{a_i} P(s'|s, a_i) = 1$. Figure 3.1 shows a typical discrete probability distribution
for a vertex (node) per action (direction). Note that in practice the probability mass is distributed around neighboring nodes and zero otherwise.

### 3.2.2 Computing the Navigation Field

When the navigation goal is specified (either the robot requests to be guided to a certain place, or a sensor node requires the robot’s assistance), the node that is closest to the goal triggers a navigation field computation. During this computation every node probabilistically determines the optimal direction in which the robot should move, when in its vicinity. The computed optimal directions of all nodes in conjunction compose the navigation field. This provides the robot with the 'best possible' direction that has to be taken in order to reach the goal. Note that the 'kidnapped' robot problem [21] is solved by our system implicitly and does not require re-computation (or re-planning).

It may be noted that a parallel approach for the construction of a navigation field has been proposed in the sensor network literature [36]. Instead of value iteration [36] uses potential fields and the hop count to compute the magnitude of the directional vectors.

#### 3.2.2.1 Theoretical Framework - Value Iteration

Our model for the proposed system is Markovian - the state the robot transitions to depends only on the current state and action. We model the navigation problem as a Markov Decision Process [66]. To compute the best action at a given vertex we use the value iteration [31] algorithm on the set of vertices $S - s_g$, where $s_g$ is the goal state. The general idea behind the value iteration is to compute the utilities for every state and then pick the actions that yield a path towards the goal with maximum expected utility. The utility is incrementally computed:
\[ U_{t+1}(s) = C(s, a) + \max_{a \in A(s)} \sum_{s' \in S \setminus s} P(s'|s, a) \times U_t(s') \] (3.1)

where \( C(s, a) \) is the cost associated with moving to the next vertex. Usually the cost is chosen to be a negative number which is smaller than \(-1/k\) where \( k \) is the number of vertices. The rationale is that the robot should 'pay' for taking an action (otherwise any path that the robot might take would have the same utility), however, the cost should not be too big (otherwise the robot might prefer to stay at the same state). Initially the utility of the goal state is set to 1 and of the other states to 0. Given the utilities, an *action policy* is computed for every state \( s \) as follows:

\[ \pi(s) = \arg \max_{a \in A(s)} \sum_{s' \in S \setminus s} P(s'|s, a) \times U(s') \] (3.2)

Since the robot maintains a probabilistic model for the transition graph, it can compute the action policy at each node for any destination point. In practice, however, this is limiting, since it requires the robot to traverse the network many times to learn the transition model. Further, another robot deployed into the same environment would need to first traverse the deployed network before it can navigate between any two points optimally.

One solution is for the robot to compute the action policy as above, and while traversing the network record the optimal action for the current node as it passes by. Each node can store this action and can emit it as part of the message directed to a robot. This would help other robots (which may not yet have explored the entire space) use the information for navigation. However, this solution is inefficient, since it is slow to adapt if the navigation goal is changed.
3.2.2.2 Distributed Computation and In-network Processing

A much more attractive solution is to compute the action policy distributively in the deployed network. The idea is that every node in the network updates its utility and computes the optimal navigation action (for a robot in its vicinity) on its own. When the navigation goal is determined, the node that is closest to the goal triggers the computation by injecting a *Start Computation* packet into the network containing its id. Every node redirects this packet to its neighbors using flooding. Nodes that receive the *Start Computation* packet initialize utilities and the cost values depending on whether the particular node is specified as a goal or not. Every node updates the utilities according to Equation 3.1. Note that the utilities of neighboring nodes are needed as well, hence, the node queries its neighbors for corresponding utilities. Since computation on some nodes can proceed faster than others, every node stores computed utilities in a list, so that even if it is queried by its neighbors for a utility several steps prior to the current one, the list is accessed and the corresponding utility is sent.

After the utilities are computed, every node computes an optimal policy for itself according to Equation 3.2. Neighboring nodes are queried once again for the final utility values. The computed optimal action is stored at each node and is emitted as part of a *suggestion* packet that a robot would receive if in the vicinity of the node.

This technique allows the robot to navigate through the environment between any two nodes of the deployed network. Note that the action policy computation is done only once and does not need to be recomputed unless the goal changes. Also, note that the utility update equations have to be executed until the desired accuracy is achieved. For practical reasons the accuracy in our algorithm is set to $10^{-3}$, which requires a reasonable number of executions of the utility update.
equation per state and thus, the list of utilities that every node needs to store is small. Since the computation and memory requirements are small it is possible to implement this approach on the real node device that we use for the experiments (the mote [52]).

Note that if neighbors of all nodes are known exactly (for every direction each node has at most one neighbor), then $P(s'|s,a) = 1$. Hence, Equations 1 and 2 reduce to the maximization of utilities of neighboring nodes only. In this case the system converges after a single iteration.

### 3.2.3 Navigation and Practical Considerations

The deployed sensor network discretizes the environment. Consider Figure 3.2. On the way from start node 1 to goal node 5, the robot would first navigate from node 1 to 2, then from 2 to 3, and so on. Hence, the navigation is node-wise. A node whose directional suggestion the robot is currently following is called the current node. Initially the current node is set to the node closest to the robot. The bottom part of Figure 3.2 shows the three phases of navigation. Suppose initially the current node is set to node 1 (robot’s position at the bottom right corner in the Figure 3.2). Node 1 suggests the robot to go ’UP’. In the first phase the robot accepts this command and positions itself in the correct direction. During the second phase, the robot moves ’forward’ using the VFH [58] algorithm for local navigation and obstacle avoidance. Note that throughout the second phase the current node is set to node 1. Phase 3 is triggered when the robot determines that it has entered the neighborhood of the next node - say, node 2 (an oval $M_2$ on Figure 3.2). During phase 3 the current node switches and the navigation algorithm starts from phase 1 again, but with the current node set to 2. A key operational question is how to determine when the robot is in the neighborhood of some node? A straightforward approach is to use signal strength thresholding. In this case, prior to the experiment an observation model can be
built which given a signal strength value would approximate the distance from the node. Hence, ideally, while in phase 2, the robot would simply collect signal strength values from the packets of all nodes in the vicinity, feed the model with these values and threshold an output picking the shortest distance.

The problem with such an approach is that raw signal strength values are neither constant nor even proportional from radio to radio, from one environmental topology to another, etc. Experimental results show that such an approach is not reliable or accurate. To reliably predict which node neighborhood the robot is in, we developed an algorithm called Adaptive Delta Percent, based on processing signal strength values in the following manner.

Let the robot’s current node be node $t$. This node suggests to the robot to travel in a certain direction. Assume that before reaching the next node the robot receives $n$ samples of radio signal strength from each of the $k$ nodes to which the robot might switch to (i.e. candidate nodes). Then for each of the $k$ nodes:
Figure 3.3: Illustration of Delta Percent Algorithm. Arc $R$ represents an ideal signal strength space that the robot receives starting from node $t$, going towards node $x$. Note that metric data is not involved in the figure and the desired 'switching' place is around areas $M_1$ or $M_2$

1. Compute an initial maximum average $A_{im}$ - an average of the first $i$ samples where $i << n$.

2. Compute a running average $A_r$ which is an average of $j$ consecutive samples where $j << i$.

3. If $R = \frac{A_r}{A_{im}} < M$, where $M$ is a threshold value, then return from the algorithm. Put $R$ into list $L_R$.

4. If $y$ consecutive elements of $L_R$ are in nondecreasing order, then return from the algorithm, else repeat 2 through 4.

In case several nodes returned from the algorithm, pick the node with the smallest ratio and switch to it. In our experimental setup $n \approx 15$, $i \approx 5$, $j \approx 10$ and $y \approx 3$. Experimentally we
determined threshold $M = 0.65$. Consider Figure 3.3. An arc $R$ represents ideal signal strength space that the robot would be moving in on its way from node $k$ to some final node $x$. Note that in reality the signal strength space is not uniform. The desired 'switching' place in the neighborhood of $x$ is around areas $M_1$ or $M_2$. Steps 1-3 of the algorithm try to estimate if the robot is in area $M_1$. Step 4, on the other hand is checking if the robot has passed area $M_1$ and now is in area $M_2$. 

Figure 3.4: Map of the experimental environment. Nodes (marked 1 - 9) were manually predeployed.
This scenario can happen if threshold $M$ is difficult or incorrectly determined or parameters $i$ and $j$ were chosen inappropriately.

### 3.3 Navigation Experiments

We conducted experiments at Intel Research facilities in Hillsboro, Oregon. We used a Pioneer 2DX mobile robot, with 180° laser range finder used for obstacle avoidance, and a base station (Mica 2 mote) for communicating with the sensor network. Mica 2 motes were used as nodes of the sensor network. A sensor network of 9 nodes was predeployed into the environment. Every node is preprogrammed with information about its neighbors. We assume that the transition probabilities are computed as described in Section 3.2.1. A map of the experimental environment and deployed sensor network of 9 sensor nodes is shown in Figure 3.4. The environment itself resembles a regular cubicle-office-like environment with narrow corridors (about 1 m), changing topology, crowded with people and obstacles. Figure 3.5 shows the mobile robot and one of the
Figure 3.6: Trajectories of robot navigating to five different goals during different trials. The start location in each case is near node 1.
deployed nodes in the environment. The algorithm proceeds as discussed in the previous sections. The task of the robot is to navigate from the ‘home base’ (around node 1) towards a (manually triggered) goal node. The requirements that we impose for the experiment to be successful are that the navigation field should yield the shortest path from any point to the goal node, the robot should follow the shortest path, and the robot should stop within 3 meters of the goal node. Note that the shortest distance between any two neighboring nodes is 7 meters.

Since the robot does not have an IMU or a compass (in the experimental environment these devices were not useful due to magnetic disturbances), the direction in which the robot is initially facing is explicitly set. During the experiments the robot maintains a notion of virtual direction, that is, given the initially set direction the robot switches virtual direction only when nodes tell the robot to switch direction. We conducted 10 experiments for five different goal nodes (we set off an alarm at five different nodes) - 3, 5, 6, 8 and 9 (50 experiments altogether). Table 3.1 shows the final distances from the robot to the goal nodes after the robot has signaled that it had completed navigation. The length of navigation paths that the robot traveled combined is over 1 km. The robot was able to navigate to the correct goal node in all cases. Representative trajectories that the robot took on its route from the start (node 1) to five goal nodes are depicted in Figure 3.6. As the results show, our algorithm provides precise and reliable navigation.

### 3.4 Combined Coverage and Navigation Experiments

We conducted a continuous experiment that tests the combined sensor network and robot sub-system for reliability and robustness to environmental changes, problems in the network and the ability to deploy and maintain a network and use it for coverage/exploration and navigation.
Table 3.1: Experimental data (distance to goal at finish, in meters). Five goals, ten trials per goal.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Goal 3</th>
<th>Goal 5</th>
<th>Goal 6</th>
<th>Goal 8</th>
<th>Goal 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>1.4</td>
<td>0.78</td>
<td>2.9</td>
<td>0.96</td>
</tr>
<tr>
<td>2</td>
<td>0.82</td>
<td>1.26</td>
<td>0.86</td>
<td>1.6</td>
<td>0.96</td>
</tr>
<tr>
<td>3</td>
<td>0.94</td>
<td>1.45</td>
<td>0.72</td>
<td>1.62</td>
<td>1.35</td>
</tr>
<tr>
<td>4</td>
<td>0.91</td>
<td>1.41</td>
<td>0.91</td>
<td>2.4</td>
<td>1.26</td>
</tr>
<tr>
<td>5</td>
<td>0.85</td>
<td>1.4</td>
<td>0.87</td>
<td>1.4</td>
<td>1.21</td>
</tr>
<tr>
<td>6</td>
<td>0.97</td>
<td>1.39</td>
<td>1.3</td>
<td>2.1</td>
<td>1.24</td>
</tr>
<tr>
<td>7</td>
<td>0.85</td>
<td>1.01</td>
<td>0.85</td>
<td>1.7</td>
<td>0.95</td>
</tr>
<tr>
<td>8</td>
<td>0.98</td>
<td>1.55</td>
<td>0.88</td>
<td>2.8</td>
<td>1.51</td>
</tr>
<tr>
<td>9</td>
<td>0.89</td>
<td>1.5</td>
<td>0.55</td>
<td>1.79</td>
<td>1.4</td>
</tr>
<tr>
<td>10</td>
<td>0.66</td>
<td>1.04</td>
<td>1.02</td>
<td>2.1</td>
<td>0.92</td>
</tr>
<tr>
<td>Average</td>
<td>0.86</td>
<td>1.34</td>
<td>0.87</td>
<td>2.04</td>
<td>1.17</td>
</tr>
</tbody>
</table>

The scenario consists of four phases. In Phase 1 the robot’s task is to deploy the network and cover/explore the environment completely. In Phase 2 we cause certain nodes in the network to fail and require them to be replaced, thus, the goal of the algorithm is to find the gaps in the network and replace the damaged nodes, while covering the environment. Phase 3 distorts the environment by introducing an extra space - a “hidden room” which also has to be covered. Then, the robot computes the transition probabilities and stores the appropriate constants at every node.

In Phase 4, we assume that another robot appears on the scene, which does not have any prior knowledge about the environment and the deployed sensor network. The first three phases are running LRV, discussed in the previous chapter, and Phase 4 executes the navigation algorithm described in this chapter which allows the new robot to home to a desired goal location.

### 3.4.1 First Phase

The deployment of the sensor network for this stage of the scenario is presented in Figure 3.7 in sequence.
As shown in Figure 3.7, the robot deployed the network over the whole environment, while at the same time accomplishing coverage. Figure 3.10 plots coverage values over the first three phases of the experiment.

### 3.4.2 Second Phase

As shown in Figure 3.8a, several nodes of the sensor network were removed (nodes in the upper part of the figure are assumed to be malfunctioned or damaged). As seen in the Figure 3.8, the gap in the network has been detected by the robot and repaired. Note that the robot continued coverage of the environment (Figure 3.10) and was not affected by the problems in the network.

### 3.4.3 Third Phase

In this phase of the experiment, we assume that certain perturbations occurred in the environment so that the robot starts with the environment shown in Figure 3.9a. Figure 3.9bc show expansion of the network by deployment of additional nodes into new open space by the robot. Note, that
the problem of coverage was not abandoned by the robot under the circumstances depicted in the last three phases. A unified view of cover time for three phases is shown in Figure 3.10. In addition, the robot injects a *Start Computation* packet into the network and the *navigation field* is computed.

### 3.4.4 Fourth Phase

In the fourth and final phase, the task of the robot is to use the navigation field and to drive to the home area marked H (Figure 3.11a). Figure 3.11b shows the navigation field that was produced by distributed computation of the optimal policy by the deployed network. The path that the robot traverses is shown in Figure 3.11c.
Figure 3.9: Deployment of additional nodes into the discovered open space.

Figure 3.10: Coverage over the first three stages of the experiment. The robot continued coverage of the environment and was not affected by the problems in the network.
3.5 Summary

We have presented an algorithm that allows a robot to navigate precisely and reliably using a deployed sensor network. Our approach differs from systems described in the literature because we assume a map, localization or GPS are not available. Our navigation approach can be used by simple (modest computation, communication and sensing requirements) heterogeneous robots.

We model the navigation problem as a Markov Decision Process. To compute the best action at a given vertex we use a distributed version of the value iteration algorithm on the set of vertices comprising the network. The general idea behind value iteration is to compute the utilities for every state and then pick the actions that yield a path towards the goal with maximum expected utility. The utility is incrementally computed. Given the utilities, an action policy (optimal direction) is computed for every state (node) \( s \). The computed optimal directions of all nodes in
conjunction compose the navigation field which provides the robot with the 'best possible' direction that has to be taken in order to reach the goal. Our algorithm naturally solves the 'kidnapped' robot problem. Extensive experimental results were presented showing the successful application of the algorithm on a real-robot and network testbed.

In Chapter 2 we presented an algorithm for robot-based network deployment and coverage/exploration. In this Chapter we presented an algorithm that allows a robot to navigate precisely and reliably using the deployed network. The presented algorithm, hence, validates our second hypothesis:

**Hypothesis 2 [Network mediated Robot Navigation]:**

*An embedded sensing and communication network of static nodes can be used to guide the robot navigation.*

At this stage, an active infrastructure is physically embedded into the environment. Moreover, we know how to use this infrastructure for robot navigation. In the next Chapter we address the final hypothesis (multi-robot task allocation) required to establish our thesis and, hence, enable high-fidelity spatiotemporal monitoring.
Chapter 4

Network-mediated Multi-Robot Task Allocation

Chapter 1 introduced the problem of designing a system to monitor spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment and stated three hypotheses. In Chapter 2 we demonstrated an algorithm for the deployment and maintenance of a static network by robots. We have also demonstrated algorithms for exploration (Chapter 2) and navigation (Chapter 3) where robots use the deployed network to efficiently explore an unknown environment and navigate to a designated goal. In Chapters 2 and 3 we validated the first two hypotheses. The third, and final, hypothesis that we need to validate is that:

Hypothesis 3 [Network mediated Multi-Robot Task Allocation]:
An embedded sensing and communication network of static nodes can efficiently mediate multi-robot task allocation to enable high fidelity spatiotemporal sampling.

The Multi-Robot Task Allocation (MRTA) problem has been well-studied in the robotics community [24], and is simply stated as the problem of allocating tasks to robots. Of particular interest is the online version of the problem (OMRTA), where tasks in the environment are geographically and temporally spread, and robots need to visit task locations to accomplish task completion. The problem is to assign tasks to robots optimally in an online fashion. Previous
MRTA approaches in the robotics community have focused on performing the task allocation computation on the robots, or at some centralized location external to the robots. All the sensing associated with tasks, and robot localization, is typically performed on the robots themselves.

Our approach to OMRTA is based on the interaction between a sensor network and mobile robots. Tasks, upon arrival, are allocated to robots by a static sensor network. In this Chapter we propose two approaches termed Distributed In-Network Task Allocation (DINTA) and Multi Field DINTA (DINTA-MF). In both cases, we assume the network is pre-deployed (through means outlined in Chapter 2, and robots have to perform spatially and temporally distributed tasks efficiently. We consider the online task assignment problem, in which tasks need to be assigned to robots in real time and the distribution of tasks’ arrival is not known a priori. Our solution is to allow the process of task allocation to occur in the network through distributed computation.

The basic idea of DINTA is that given a set of tasks detected by the network (e.g. nodes detecting change in PAR, humidity, temperature, etc.), every node $k$ in the network computes a suggested motion direction for a robot if it is in the vicinity of $k$. The ensemble of suggested directions computed over all nodes is called a navigation field. An adaptive distributed value iteration algorithm is used to compute the navigation field.

DINTA-MF, is a variant of DINTA, where multiple navigation fields (one for every task) are maintained in the network at a given time. Fields are assigned to robots using a greedy policy. The difference between the two approaches is that in DINTA-MF every network node computes the direction that a robot should follow in its locality for every task in the environment.

We study a particular experimental scenario, event handling, as an experimental substrate. In prior work [47], a similar scenario has been used to study the role of opportunism vs. commitment in MRTA. In our experimental scenario, events in the environment trigger alarms. An alarm is
spatially focused, but has temporal extent (i.e. it remains on until it is turned off by a robot). Alarms are detected by sensor nodes. We abstract the notion of an alarm to a sensory event generated by a node in a network if ’something interesting’ has been detected. For example an event might be generated in case a change in PAR reading is detected, and a sample needs to be taken.

The task of the team of robots is to turn off the alarms by notionally responding to the emergency signaled by each alarm. This is done by a robot navigating to the location of the alarm which causes the alarm to shut off. The goal is to minimize the cumulative alarm On Time across all alarms, over the duration of the entire experiment. Each alarm’s On Time is computed as the difference between the time the alarm was turned off by a robot and the time the alarm was detected by one of the nodes of the network. We make the assumptions that the network is already deployed and the distributions of transition probabilities \( P(s' | s, a) \) are already computed (Section 3.2.1). The appropriate distributions are stored on corresponding nodes. Further, we assume that an alarm requires at least one robot to service it. To turn off an alarm, a robot needs to appear in its vicinity. Thus, the handling of the alarm is purely notional since that is not our focus here. The nodes of the sensor network are time synchronized (high precision is not required). One of several existing techniques may be used for this. As an example see [17]. The robots do not have a pre-decided environment map or access to GPS, and do not perform localization or mapping.

There are many applications of MRTA besides spatiotemporal phenomena monitoring, including security, monitoring, and urban search and rescue (USAR) in the aftermath of a natural or man-made disaster. Further, the ability of a sensor network to assign tasks to different robots, thus serving as a multi-purpose infrastructure, enables solutions to problems requiring heterogeneous groups of robots. Even though we study the task allocation problem in the context of
mobile robots, the proposed system can be applied for the general online task assignment problem where the resources are different from robots (e.g. people trying to get outside of the building would be guided (tasked) to the closest exits by the network).

4.1 Related Work

The problem of multi-robot task allocation (MRTA) has received considerable attention. For an overview and comparison of the key MRTA architectures see [24], which subdivides MRTA architectures into behavior-based and auction-based. For example, ALLIANCE [49] is a behavior-based architecture that considers all tasks for (re)assignment at every iteration based on robots’ utility. Utility is computed by measures of acquiescence and impatience. Broadcast of Local Eligibility [65] is also a behavior-based approach, with fixed-priority tasks. For every task there exists a behavior capable of executing the task and estimating the utility of robot executing the task. Auction-based approaches include the M+ system [8] and Murdoch [25]. Both systems rely on the Contract Net Protocol (CNP) that makes tasks available for auction, and candidate robots make ‘bids’ that are their task-specific utility estimates. The highest bidder (i.e., the best-fit robot) wins a contract for the task and proceeds to execute it.

DINTA and DINTA-MF, differs from the above MRTA approaches in the following ways:

1. Both approaches rely on a static network, and communication, sensing and computation are distributed.

2. The utilities of task assignments are propagated and computed by the network based on purely local communication between the network nodes.
3. The system does not require mobile robots to be within communication range of each other. The network is used for propagating messages between the robots.

4. The system does not place a limitation on the number of robots. There is no computation or communication overhead associated with increasing the number of robots.

5. The system does not require one robot to recognize another robot.

4.2 Task Allocation: Offline vs. Online

The task allocation (TA) problem has two major subdivisions: Offline and Online. Offline TA is the problem of assigning robots to different tasks if the tasks’ information such as arrival time distribution, tasks’ weight or priority, etc.

\(\text{is known} \ a \ priori. \) The assignment process is thus offline. An offline TA problem, in its most general form, is equivalent to the NP-Complete conjunctive planning problem [11].

We focus on the other version of the problem - Online Task Allocation. In online TA, the information about the tasks becomes available only upon arrival and hence has to be computed in real time. Therefore, the task assignment occurs in \textit{decision epochs}.

\textbf{Definition 4.1:} A \textit{decision epoch} is a period of time during which only the alarms which have arrived are considered for assignment.

Using methodology of [23], we consider a special kind of online TA - Single Robot tasks with Single Task robots performing Instantaneous Assignment(SR-ST-IA). In other words, we limit the problem domain to the case where each task requires only one robot to complete it and each robot is competent in performing only one task. In addition, robots are assigned tasks as fast as possible.
(instantaneous assignment). Moreover, online assignment means that tasks are introduced one at a time and robots that have already been assigned cannot be reassigned [30, 23]. SR-ST-IA online assignment is one of the most simple cases, but one of the best studied and mostly used. We note that several real-life applications involving mobile robots in dynamic environments are online problems that fall into SR-ST-IA category.

4.3 Methodology

We adopt the following general online TA methodology. Suppose at a given decision epoch the system maintains a set of resources \( R = \{r_1, \ldots, r_n\} \) (robots) and weighted tasks \( T = \{t_1, \ldots, t_k\} \). Tasks are prioritized based on a criterion \( C \). \( C \) is an application dependent function and can combine such parameters as task arrival time, task importance, etc. A set of assignments \( A = (l = min(n, k) : \{a_1, \ldots, a_l\}) \) is computed as follows.

\[
\forall a \in A: a = \arg \max_j (U(r_j, t))
\]

(4.1)

where \( t \) is the next unassigned task according to \( C \) and \( U(r_j, t) \) is the \( j \)-th resource (robot) utility value for accomplishing \( t \). The assigned robot and corresponding task are removed from \( R \) and \( T \) respectively, before the next assignment. The utility function is chosen to be application and resource dependent. In our model, once assigned, robots cannot be reallocated. After a robot has completed its task it becomes available for a new assignment. In the terminology of [47] we adopt a commitment strategy as opposed to opportunism.

Now we can define the online TA per decision epoch.
**Definition 4.2:** Suppose at a given decision epoch there are \( n \) available robots and \( k \) weighted tasks. Suppose also that every robot has a utility measure of each task. The Online Task Allocation is the problem of assigning robots to tasks within the current decision epoch, maximizing the robots’ net utility.

The weight of a task is an application dependent abstraction and can combine such parameters as task arrival time, task importance, etc. Robot’s utility measure of a task \( t \) is an estimate of the efficiency of robot performing a task \( t \).

**Definition 4.3:** Suppose at a given decision epoch there are \( n \) available robots and a weighted task \( t \). Suppose also that every robot has a utility measure of the task \( t \). The Online Task Allocation is the problem of assigning, within the current decision epoch, a robot with the maximum utility to a task \( t \).

In this case the greedy assignment is obviously optimal.

The network monitors the environment for events of interest (motion, change in light intensity, etc). The problem then is to prioritize the events, and navigate the mobile robot to a node that detected an event. Once the robot arrives, the local phenomenon is measured. In TA terminology, a robot is a resource and a detection by a sensor node of an event requiring perusal by a robot is a task or an event.

### 4.4 DINTA: Distributed In-Network Task Allocation

The basic idea of DINTA is that given a set of alarm-weight pairs \((a_i, w_i)\) detected by the sensor network, every node \( k \) in the network computes a suggested direction that a robot should take if in the vicinity of \( k \). This computation results in a direction which maximizes the net utility of the robot. The weight \( w_i \) is an abstraction, which is a scalar representation for several parameters like priority, magnitude, time (older alarms should be served first), etc. The ensemble of suggested
directions computed over all nodes is called a navigation field. The navigation field is computed as in section 3.2.2.2. An example of a navigation field for one and three alarms is shown in Figure 4.1.

The DINTA approach has two subsystems - Coverage/Exploration and Alarm Response. If no alarms are detected, the system operates in Coverage/Exploration mode. In this mode, the navigation field computed by the network, causes the robots to patrol the environment, using the LRV algorithm (Chapter 2). If, on the other hand, an alarm is detected, the system switches to the Alarm Response mode where the navigation field computed by the network guides the robots to turn off alarms, thereby implicitly solving the MRTA problem. The Coverage/Exploration mode is the same in DINTA and DINTA-MF.

Figure 4.2 shows the data flow in a network node. The Alarm Response mode proceeds as follows. If a node receives an ALARM message with identification \( a \) of the node that detected
the alarm, weight $w$ (estimation of the alarm’s importance) and hop count $h$ (estimation of how far away node $a$ is), the alarm is placed on the list $L$ of currently active alarms according to its utility $V$ (Utility Update block). We define the utility for this approach as the ratio $V = \frac{w}{h}$.

Note that DINTA maintains only one navigation field. Thus the utility is used by each node to determine which alarm it should compute a direction for. Note also that the further away a node is from the node detected the alarm, the smaller the ratio becomes and hence the utility of further node is smaller. For example, in the case of Figure 4.1b, the three alarms have the weights so that $w_1 > w_3 > w_2$, but even alarm node $A^2$ with the smallest weight has its own subfield.

Every node maintains a current alarm variable, which is the element of $L$ with largest utility. If the current alarm changes, the Task Allocation block computes a new local task assignment for a robot (discussed next) and reroutes the alarm message with incremented hop count.
count to neighboring nodes. In this case local task assignment is the same as navigation direction which would yield maximum utility for the robot if it would take the suggested direction at this node. The direction computation proceeds according to a distributed value iteration algorithm described in section 3.2.2.2. In the global perspective, prioritizing between the tasks according to their utility value results in creation of multiple superimposed navigation fields (for example three alarms case of Figure 4.1b). Robots are simply guided by the global navigation field, and hence task allocation is *implicit*. Note that if the Sensor Data block indicates that an alarm is detected by the node itself, then the node initiates a message $ALARM(thisNodeID, w, 0)$. The General Task block represents the Coverage/Exploration subsystem (Chapter 2), in case $L$ is empty.

It is important to note that DINTA does not make explicit assignments of tasks to robot or specific robot subgroups, which may result in suboptimal behavior both in terms of time and wasted resources (several robots might pursue the same task). Consider the case when the robots are cluttered in one region (one subfield) and therefore, can all be attracted towards the same alarm or simply ignore other alarms, depending on the implementation.
4.5 The Multi-Field Approach: DINTA-MF

DINTA-MF is based on maintaining multiple navigation fields, one for every alarm at the same time and assigning those fields to different robots using a greedy policy. In other words, every node in the environment computes the ‘optimal’ direction that the robot should follow (when in the vicinity of that node) for every alarm in the environment. Figure 4.3 shows the data flow on a network node executing DINTA-MF.

The DINTA-MF approach, like DINTA, has two subsystems - Coverage/Exploration and Alarm Response. If no alarms are detected, the system operates in Coverage/Exploration mode. In this mode, the navigation field computed by the network, causes the robots to patrol the environment. If, on the other hand, an alarm is detected, the system switches to the Alarm Response mode where navigation fields (one for each alarm) are computed and explicitly assigned to robots by the network. These fields guide assigned robots to turn off alarms, thereby explicitly solving the MRTA problem.

The following describes behavior of a node in the sensor network when in the Alarm Response mode. If a node receives an ALARM message with identification $a$ of the node that detected the alarm, the system time $t$ when the alarm was detected by node $a$ and hop count $h$ from this node to the node detected the alarm (node $a$), the alarm is placed on the list $L_{UA}$ of currently active unassigned alarms according to its time $t$.

This portion of the system executes in decision epochs. A decision epoch is a period of time during which only recently arrived alarms are considered for assignment. We consider a decision epoch to be equal to one alarm, but easily can be generalized to n alarms or time units. Hence, in the experiments described here, we assign one alarm at a time. For a given decision epoch, if the
list $L_{UA}$ is not empty (i.e. it contains active unassigned alarms) and there are unassigned robots in
the vicinity, the Direction Computation block starts to compute the assignment direction
for an alarm of smallest time $t$ in the list. This computation is outlined in section 3.2.2.2. List
$L_{CD}$ contains direction assignments computed for alarms of list $L_{UA}$.

A node decides which robot should be assigned to a particular alarm based on where robots
are relative to the alarms. In a given decision epoch every node that has unassigned robot(s)
in its vicinity sends out a ROBOT-SYNCH-MESSAGE that contains robot’s id, alarm id (id
of the alarm considered in current decision epoch), and the hop count to the alarm. A node
processes such responses in Robot Synchronization block and passes this information
to the Task Assignment block. This information gives distances of unassigned robots to
the alarm considered in current decision epoch. Given shared information about the alarms and
robots relative positions to these alarms the Task Assignment block of every node assigns
the robot with shortest distance. Ties are broken in favor of robots with smaller IDs. Note also
that in order to guarantee that the system is in the same decision epoch (i.e. computes assignments
for the same tasks) a time-synchronization mechanism is needed. The General Task block
represents the Coverage/Exploration subsystem (Chapter 2), in case $L_{UA}$ is empty.

### 4.6 Simulation Experiments with MRTA

In our experiments we used the Player/Stage [26, 61] simulation engine populated with simulated
Pioneer 2DX mobile robots equipped with $180^\circ$ field-of-view planar laser range finders (used for
obstacle avoidance), wireless communication and a mote base station (to communicate with the
motes, used as network nodes). A network of 25 motes was predeployed in a test environment.
The communication range of motes and robots was set to approximately 4 meters. The task of the team of robots is to serve alarms by navigating towards the point of alarm and minimize the cumulative alarm \textit{On Time}. Each alarm’s \textit{On Time} is computed as the difference between the time the alarm was served by a robot and the time the alarm was detected by one of the nodes of the sensor network. We conducted 3 sets of experiments, measuring the cumulative alarm’s \textit{On Time} for ‘exploration only’ algorithm, DINTA and DINTA-MF. Every experiment was conducted in an environment of $576m^2$ with robot group sizes varying from 1 to 4, 10 trials per group. For experiments the schedule of 10 alarms was drawn (time-wise) from a Poisson distribution, with uniformly distributed nodes that detected the alarm. The parameter of Poisson distribution was set to $\lambda = \frac{1}{60}$, which means that the expected number of alarms is 10 in 600 seconds.
4.6.1 Efficient Exploration vs. DINTA

First we compare the performance of exploration (LRV) algorithm and DINTA (exploration and alarm response). The exploration only algorithm (LRV) consists of only one subsystem - Coverage/Exploration, described in Chapter 2. DINTA, on the other hand, consists of two subsystems - Coverage/Exploration and Alarm Response. Hence this comparison highlights the benefits of switching to a mode with purposeful assignment of alarms to robots, if alarms are present, like in DINTA. Figure 4.4 shows the OnTime comparison for 'exploration only' and DINTA. Clearly, DINTA outperforms the exploration only algorithm even though as the environment becomes saturated with robots, the difference becomes smaller. The difference is statistically significant (the T-test p-value is less than $10^{-4}$ for every pair in data set).

Moreover, the performance of DINTA is stable (small and constant variance) whereas variances produced by the exploration mode change drastically and reduce as the environment becomes saturated with robots.

4.6.2 DINTA vs. DINTA-MF

In this section we compare DINTA and DINTA-MF approaches. Note that DINTA and DINTA-MF consists of exploration subsystem and alarm response. Therefore, this experiment shows the benefit of assigning tasks explicitly (DINTA-MF) rather than implicitly (DINTA). Figure 4.5 shows the On Time comparison for DINTA and DINTA-MF. In order to test the similarity between the pairs of datasets for DINTA and DINTA-MF, we ran the T-Test. The T-Test produced the following $p$ values for corresponding data points: 0.9139 for the first pair of points which means that they are similar, 0.0081, 0.0897, 0.091 for the last three pairs of points, which means that they are different. Note that with one robot the performance of both algorithms is approximately
similar, which is due to the fact that both approaches reduce to the same solution. Although, despite the fact that environment becomes saturated with robots, DINTA-MF outperforms DINTA as the number of robots in the team increases. The reason behind this is that every node in the network computes an assignment for every alarm and hence all unassigned robots are assigned an alarm, which is not necessarily the case with one assignment field. Note also that DINTA-MF does not 'waste' resources (robots), whereas in DINTA, several robots can pursue the same alarm.

The space and time requirements for DINTA-MF are linear in the number of alarms, which makes it realistic for implementation on our target node platform (the mote).
4.7 Optimality Considerations

As discussed in Section 4.2, we consider a special kind of online Multi Robot Task Allocation - Single Robot tasks with Single Task robots performing Instantaneous Assignment (SR-ST-IA). In other words, we limit the problem domain to the case where each task requires only one robot to complete it and each robot is competent in performing only one task. In addition, robots are assigned tasks in an online fashion as fast as possible (instantaneous assignment).

The proposed algorithm, DINTA-MF, falls into a category of online greedy algorithms which assign newly introduced tasks to the currently available robot with highest utility. Such algorithms are also known in the context of network flows as the Farthest Neighbor algorithm (please refer to [23]). Such algorithms are known to be 3-competitive. That is, the quality of the solution is at most 3 times worse than that of an optimal post hoc offline solution. In addition, it has been shown [29, 23] that this performance bound is the best possible for any online assignment algorithm. Therefore, without domain knowledge or model of the tasks to be introduced, and without an option of reassigning the robots that have already been assigned, it is impossible to design a better task allocation algorithm than greedy online algorithms, such as DINTA-MF.

4.8 Explicit Sensor Network Repair and Maintenance

One of the applications of in-network task allocation is explicit sensor network repair and maintenance. Consider an example in which a sensor node detects that it is running out of power and creates an event requesting a robot to recharge or replace it. This event can be injected into the system and treated as any other task requiring the robot’s inspection/assistance. Note that special
care should be taken in assigning the weight to such tasks, since the node might run out of power before it would be inspected by a robot.

In another example a sensor node detects that one of its neighbors has not responded. Like in the first case, an event is generated, the corresponding task is injected into the system. The robot would be assigned to repair the network in this case.

In Section 2.6 we showed that LRV implicitly addresses the problem of sensor network repair and maintenance. Results of the previous section show significant improvement of using task allocation over opportunistic discovery of tasks (LRV).

4.9 Summary

In this Chapter we proposed an algorithm to validate the third, final, hypothesis:

**Hypothesis 3 [Network mediated Multi-Robot Task Allocation]:**

An embedded sensing and communication network of static nodes can efficiently mediate multi-robot task allocation to enable high fidelity spatiotemporal sampling.

We introduced DINTA: Distributed In-Network Task Allocation and DINTA-MF: Multi Field DINTA for solving the Online MRTA (Multi Robot Task Allocation) problem. These approaches allow us to combine the benefits of a sensor network with the mobility and functionality of robots. Both systems compute task assignments distributively in-network, while, at the same time, providing a virtual sensor and communication device that 'extends' throughout the whole environment and has obvious benefits over traditional OMRTA approaches. The fundamental assumption, though, is the existence of the sensor network on which robots can rely. However, Chapter 2 shows that given a set of sensor network nodes large enough, a sensor network can be
deployed into an environment and maintained by the robots. Experimental data show that DINTA outperforms LRV which is an exploration only algorithm.

Next we compared DINTA with DINTA-MF. The difference between these two approaches is that DINTA assigns tasks implicitly, whereas DINTA-MF explicitly determines which robot serves which task and hence performs generally better and does not waste resources. DINTA relies on maintaining a single navigation field consisting of several subfields, one for each alarm. The implicit assignment occurs when a robot is located in one of the computed subfields and simply travels along the path from node to node in that subfield till it reaches the goal node. DINTA-MF, on the other hand, computes a navigation field for every detected alarm and then assigns available robots in a greedy fashion, which is the best possible assignment in the case of the Online task allocation problem.

The experimental data show that DINTA-MF outperforms DINTA. The difference in the On Time metric is not large though. The reason is that the experimental environment is not large and the occurrence of alarms is rather infrequent. Hence the environment and conditions of the experiment are better suited for DINTA. Another advantage of using DINTA-MF is that it does not waste resources, whereas in DINTA, several robots can pursue the same alarm. In general, DINTA-MF proposes a solution which can handle alarms (tasks) of high frequencies, represents a multi purpose distributed manager that can solve a large variety of problems.

In terms of computation and communication overhead, DINTA-MF is comparable to the worst-case scenario of DINTA. As an example consider the case when there are several alarms in the network such that the weights of corresponding alarms drastically differ from each other. In this case in DINTA, the whole field would always consist of only one subfield, and hence a
Table 4.1: Tradeoffs between different approaches.

<table>
<thead>
<tr>
<th>Performance if $\frac{\text{number of robots}}{\text{environment size}}$ small</th>
<th>LRV</th>
<th>DINTA</th>
<th>DINTA-MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation intensity of sensor nodes</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Communication intensity</td>
<td>low</td>
<td>medium - high</td>
<td>high</td>
</tr>
<tr>
<td>Power consumption in SN</td>
<td>low</td>
<td>medium - high</td>
<td>high</td>
</tr>
<tr>
<td>Overall performance (worst case)</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
</tbody>
</table>

separate navigation field would be constructed eventually for every alarm, which is the case in terms of the communication and computation overhead in DINTA-MF.

There are several advantages in using DINTA and DINTA-MF as opposed to other MRTA approaches. The sensor network allows a robot to detect a goal (alarm) even though the alarm is not in the robot’s sensor range. In addition, robots can use the sensor network to relay messages if they are not within communication range of each other. One of the other benefits of using DINTA or DINTA-MF is distributed in-network computation, which 1. avoids redundant computation by updating the state of a node based only on the state of its neighbors and robots in the vicinity (scalability), 2. computes utilities in the network distributively and propagates from the goal state (alarm). Another benefit is the ease of determining relative distance to the goal (for determining utilities) by considering hop counts from the goal node. Note also that robots using proposed approaches can be quite simple - they do not need to localize and map the environment - they can navigate by following to the suggestions from the sensor network. The tradeoffs between these approaches are shown in Table 4.1.

In Chapter 2 we presented an algorithm for robot-based network deployment and coverage/exploration. In Chapter 3 we presented an algorithm that allows a robot to navigate precisely and reliably using the deployed network. The results of these chapters allowed us to physically
embed an active infrastructure into the environment. Moreover, we know how to use this infra-
structure for robot navigation. These results validate the first two hypotheses. In this Chapter we
addressed the final hypothesis needed to enable high-fidelity spatiotemporal monitoring - multi-
robot task allocation. The system presented validates the third, final hypothesis. The next Chapter
gives an example of how methodologies, presented in this Chapter, can be applied to the problem
of spatiotemporal sampling in a real system, deployed in a mountain reserve.
Chapter 5

Task Allocation in a Networked Info-Mechanical System

In this Chapter we present an example of a symbiotic system, which is deployed in a natural habitat and used for monitoring of spatiotemporal phenomena such as PAR.

5.1 Introduction to Networked Info-Mechanical Systems

We have implemented our task allocation methodologies, presented in Chapter 4, on a new kind of system - Networked Info-Mechanical System (NIMS). NIMS is a large interdisciplinary project with many participants and facets (sensor and node design, uncertainty modeling, forest ecology, etc.) which are not discussed here. For details the reader is referred to [45]. Figure 5.1 shows a NIMS system deployed in a forest reserve for continuous operation. This system includes supporting cable infrastructure, a horizontally moving mobile robot (the NIMS node) equipped with image sensing, and a vertically mobile meteorological sensor system carrying water vapor, temperature, and PAR sensing capability.

NIMS architecture incorporates static sensor network which is comprised of vertically suspended static sensors that are referred to as sensor strands. Sensor strands, also exploiting infrastructure, are suspended in the plane parallel to the NIMS sampling transect (Figure 5.2). Every
Figure 5.1: NIMS system deployed in the forest reserve for continuous operation. (Courtesy of the NIMS project [45])
Figure 5.2: NIMS deployed at the James San Jacinto Mountain Reserve (http://www.jamesreserve.edu). a) This image shows the NIMS cable infrastructure, horizontal transport node (carrying an embedded computing platform, image sensor, vertical transport control, and vertically mobile meteorological sensor node. (Courtesy of the NIMS project [45]) b) Schematic representation of the deployed NIMS. Both mobile and static nodes are shown. Static nodes are arranged in strands.
sensor strand includes two sensors separated by 1-2 meters. Data from strand sensor elements is sampled by an embedded sensor node based on the Intel Stargate™ platform. Communication between strand nodes and the mobile, horizontal node occurs over an IEEE 802.11b wireless interface. Figure 5.2 schematically shows the experimental architecture at the James San Jacinto Mountain Reserve. As shown in Figure 5.2, there are three sensor strands with six PAR sensors deployed in this transect. Note that each sensor samples PAR values that are then considered to be representative of the light intensity in the vicinity of this sensor. Hence, the sampling transect is discretized into six cells. The sampling scheduling (determining where and when to sample) is guided by the Task Allocation system hosted on the NIMS node.

The NIMS infrastructure is elevated in the environment and thus lies above environmental obstacles to solar radiation. NIMS systems have exploited this and have been deployed with battery energy sources supplied solely by solar photovoltaic cell energy harvesting. Energy is transported as needed along the NIMS cable infrastructure by an articulated cable system. The NIMS system is deployed in a transect of length 70m and average height of 15m with a total area of over 1000 $m^2$.

The experimental NIMS system operates with a linear speed range for node motion of 0.1 to 1 m/second. Thus, the time required to map an entire 1000 $m^2$ transect with 0.1 $m^2$ resolution *with a simple 'raster' regular scanning schedule* will exceed $10^4$ to $10^5$ seconds. Phenomena that vary at a characteristic rate exceeding this scanning rate may not be accurately mapped. Experimental results with using in-network task allocation methodology described in this thesis show an order of magnitude improvement in phenomenon mapping time. Experiments reported in this Chapter were conducted on a NIMS system deployed at the James San Jacinto Mountain Reserve (http://www.jamesreserve.edu) or on either of the two NIMS test systems. The first test
Figure 5.3: The NIMS Lab System (NIMS-LS). (Courtesy of the NIMS project [45]) a) Schematic of NIMS-LS. b) NIMS-LS main node.

The second test system, NIMS-SIM, operates in a simulation mode that allows to replay the data obtained by the real NIMS, and simulate some or all of the physical devices of NIMS.

### 5.1.1 NIMS Test Systems

The second test system, NIMS-LS, operates in an indoor laboratory-scale facility (hence the name) where all inputs may be controlled. NIMS-LS includes a mobile node system suspended by a cable network (see Figure 5.3a) that is articulated by a stepper motor control system (see Figure 5.3b) controlled by a *Stargate*™ embedded node. The motor control system alternately winds and unwinds cable length from a pair of cable spools thereby causing a horizontal and vertical translation of the mobile node. The actuator system includes acceleration and deceleration control as well as odometer sensing of motion on the NIMS infrastructure cable. Through
proper calibration, this system provides less than 1 cm resolution for localization of the node at any point within the transect plane. While configurable in height and width, the transect used for the experiments reported here was 8m in length and 2.5m in height. Fixed sensor nodes were distributed on the surface of the transect region and also elevated in the transect, much like sensor strands in a real NIMS.

The mobile node is a standard wireless mote sensor system. The MICA-2 mote is used as a standard wireless mote in both the mobile node and in a wireless network with both the Stargate platform as well as the distributed fixed sensors.

5.2 NIMS Task Allocation System

The methodology presented in Chapter 4 is used for the task allocation system on NIMS. The system consists of a mobile node suspended on a cable and a static sensor network. We assume that the network is predeployed where each node knows its location in a global coordinate system. The network monitors the environment for events of interest (motion, change in light intensity, etc). The problem then is to prioritize the events, pick the robot with maximum utility and drive the mobile node to a vantage point from which a particular event is better observed. Once the robot arrives at the event location, the local phenomenon is measured. In TA terminology, a robot is a resource and a detection by a sensor node of an event requiring perusal by a robot is a task.

Figure 5.4 shows two network topologies that we define - positioned on the ground (the 2D-case) and more generally, in the volume surrounding the transect (the 3D-case). In order for the TA system to plan the node’s motion the goal points should lie in the transect plane. Hence, we project the nodes locations onto the transect plane. As a result we get a set of points on a line
Figure 5.4: Different static sensor network topologies and corresponding projections onto the transect.

$l$ (2D-case, Figure 5.4a) or a plane $\Pi_r$ (3D-case, Figure 5.4c), both of which lie in the transect plane. In the 2D-case, $l$ is the line where the transect plane intersects the ground plane. Since, the mobile node cannot move along that line, we translate $l$ to a parallel line $l_r$ on the transect. We define the projection function in the 2D-case $\text{PROJ}_l$ and 3D-case $\text{PROJ}_{\Pi_l}$.

Based on tasks projected locations TA divides the transect into slices (2D-case, Figure 5.4b), or generally cells (3D-case, Figure 5.4d). With every projected node $k$ we associate a cell $C_n$.

Note that a 2D system is sometimes preferred because it is easier to setup in the field and for some applications a 2D perspective is enough. As an example, consider studying sunlight intensity shining through a forest canopy. In this case a sensor network with illumination detectors can be placed on the ground. Suppose node $k$ discovered an interesting reading (say an abnormal light value). The TA system then would guide the most fit robot towards the goal point on $l_r$. 
computed by $\text{PROJ}_l$. The mobile node then can study appropriate slice $C_k$. The general 3D-case system is investigated here.

Note that we consider the problem of assigning tasks one at a time. In this case the greedy assignment is obviously optimal. The NIMS Task Allocation system consists of two algorithms, one running on the static sensor nodes and the other on the robots. The algorithm of static nodes is simple - retrieve data from the sensors, process it, and deliver to the robots via a wireless link.

The algorithm running on the robots is as follows. For simplicity, assume we consider a single robot assignment problem. Suppose a robot receives the sensor data from the static node $i$. This data is analyzed and if there is a difference greater than a threshold in the current sensor data with respect to the previously stored value, a sampling task for the sensor node $i$ is created. The task for the robot is then to travel to the location of the node that generated the task (after that a sampling policy can be applied to the vicinity of the static node, but this is not our focus here). Next, if the task generated by node $i$ is not stored in a set of currently active tasks $T_a$, it is added to this set. If the mobile robot is available for the next task and $T_a \neq 0$, the next task is extracted from $T_a$ according to the criterion $C$. For the single robot case we implemented two policies for the criterion $C$ - time policy (tasks with smaller time stamp get priority) and distance policy (tasks closer to the robot get priority). Note that since the system does not have any apriori knowledge about the spatiotemporal variation of event arrival, simple greedy scheduling is appropriate. Next, based on the task information the mobile node needs to compute a goal point. If the task’s position is $p$ then the goal position will be $\text{PROJ}_l(p)$ in 2D-case and $\text{PROJ}_{l, r}(p)$ in 3D-case (see Figure 5.4). The robot then moves to the computed location of the task. After the robot completes its last task it becomes available for reassignment.
Now, suppose there are multiple robots available for the assignment. In addition to the algorithm above, there is a need for a negotiation to take place among the robots. Available robots exchange information about the tasks in the system, prioritize the tasks according to the criterion \( C \), and exchange the utility values for accomplishing a task of highest priority. For the multi-robot case, we used for the \textit{time} policy as a criterion \( C \). Next, the robot with the highest utility is assigned the task (according to equation 4.3). Note that in this work distance to the task origin is taken as robot’s utility.

5.3 Experimental Results

In our experiments we used a real NIMS system (Section 5.1), deployed at mountain reserve, as well as NIMS test systems (Section 5.1.1). Sections 5.3.1 and 5.3.2, through the experimental and simulation results, establish the benefits of using our Task Allocation technique and its adequacy and effectiveness for spatiotemporal sampling. These experiments are performed with a single robot system and 6 static sensor nodes. Section 5.3.3 considers multi-robot NIMS and shows the change in system performance as the number of robots available for task allocation increases.

5.3.1 Task Allocation vs. Raster Scan

In the following experiments we will compare the cumulative task \textit{OnTime} across all tasks, over the duration of every experiment. Each task’s \textit{OnTime} is computed as the difference between the time the task was serviced by a robot and the time the task was detected by one of the nodes of the network. For the following experiments we used NIMS-LS system presented in Section 5.1.1. As shown on Figure 5.3, a network of 6 Mica2 Motes was predeployed in a test environment
with predetermined coordinates. We use the general 3D topology. Hence, by knowing nodes locations and computing nodes’ projections onto the transect plane, the TA algorithm produces a subdivision of the transect similar to Figure 5.4b.

Experiments were conducted comparing Task Allocation system with conventional Raster Scan method. The Raster Scan samples every point of the transect with a specified resolution. When the Raster Scan reaches the location of an event it clears it by sending a prespecified message. Raster Scan proved to be prohibitively low in performance. In particular, experimental results showed that at the maximum NIMS-LS spatial resolution of 1 cm, with a sampling dwell time of 1 second at each location, OnTime results were dramatically inferior to TA system. Raster Scan was also characterized at reduced spatial resolution of 5cm with a corresponding improvement in response time. This however, is still inferior to TA system described here.

In this experiment an artificial event is first generated on a remote server. Then the server sends an event message to the node designated for task generation and the node proceeds as if this event was detected by the node’s sensor. For this experiment, schedules of 3, 5, 7, 10 and 20 events were drawn (in time) from a Uniform distribution to arrive within 600 seconds, with uniformly distributed nodes that detected the event. Note that for actual applications we do not expect to receive/process more than 1 - 10 events in 10 minutes on average. Hence the case of 20 events shows the behavior of the system at the limit.

Figure 5.5 shows experimental results comparing OnTime performance of TA and Raster Scan. The number of events varies between 3 and 20. Both algorithms were evaluated from 3 different starting positions of the mobile node on the transect (drawn from a Uniform distribution). The results were averaged. As can be seen from the graph, TA performs 9-22 times better on the entire interval of 3-20 events. Note also that TA is stable, as indicated by error bars, and hence
is favored for use in this application since it provides reduced bounds on system run time over Raster Scan.

In addition to response time comparison, it is also important to compare mobility requirements for TA and Raster Scan. Specifically, the use of mobility requires energy. Thus, this can be computed and compared for each method. Now, when the density of the events is low, the TA algorithm enables the mobile node system to remain in a static position for extended periods - “in between events”. This occurs when it has serviced all events that have arrived and is awaiting new events. Raster Scan, however, forces the robot to move constantly. Hence, this method will consume far greater energy and mobility resources than TA. A measure of energy for mobility is determined for the purposes of comparison by computing the total time of mobile node motion. Figure 5.6 shows comparison of energy consumption in units of time-in-motion between TA and
Figure 5.6: Comparison of energy consumption in units of time-in-motion (t.i.m.) between TA and Raster Scan. Number of events varies between 3 and 20.

Raster Scan. As expected, TA outperforms Raster Scan significantly. However as the number of events increases to infinity, the TA should approach Raster Scan energy consumption. Also note, that on interval [5, 20] the slope of the Raster Scan curve is very small and the energy consumption is insensitive to event arrival rate.

5.3.2 Experimental Results for Time and Distance Policies

We performed a set of experiments using our Task Allocation system and compared two policies - *Time* (tasks with smaller time stamp get priority) and *Distance* (tasks closer to the robot get priority).
Figure 5.7: PAR data acquired by the first sensor strand during one of the field experiments. Events generated and serviced are shown for Time and Distance policies. Note that events are rendered time of occurrence vs. the PAR value of the event.
Figure 5.8: OnTime in form of a zero-mean Gaussian distributions for Time and Distance policies. The OnTime of events generated by all sensors is considered. Dotted (blue or lighter) graphs show the distributions at original mean.

Figure 5.9: PAR data acquired by the sensor strands on August 21st 2004, from 10:33 till 20:00.
5.3.2.1 Experiments in the Mountain Reserve

First, a set of experiments was conducted on a NIMS deployed in The James San Jacinto Mountain Reserve (as shown in Figure 5.2). Note that only representative graphs are presented. Figure 5.7 shows the representative PAR data from sensor 1 and 2 collected during the operation of the *Time* policy (Figure 5.7ab) and the *Distance* policy (Figure 5.7cd). Figure 5.7 also shows points in time when events were generated and serviced by both policies for 2 sensors. Note that events are generated in response to fluctuations in PAR. As shown in Figure 5.7, events are generated proportionally to the density of the 'spikes’ in PAR data and cover all significant 'spikes’ of PAR data.

As an evaluation metric of the task allocation policy performance we use the task (event) *On-Time*. Figure 5.8 shows the comparison between the cumulative event *OnTime* of the *Time* policy and the *Distance* policy. For visualization purposes, in Figure 5.8 event’s *OnTime* is presented in a form of a zero-mean Gaussian distribution. It follows that the *Distance* policy has smaller average *OnTime* with smaller deviation.

These experiments show that the presented task allocation system achieves spatiotemporal sampling with both policies. However, in order to compare the performance of the two policies we need to run each on the *same* and longer set of data.

5.3.2.2 TA Policies: Time vs. Distance

In order to compare the performance and characteristics of the two policies we have recorded a set of sensor strand data (approximately 9 hours and 30 minutes) spanning one day. Then we can *replay* the strand data through the same interface as in the field system in our lab testbed system. The testbed system is computationally identical to the NIMS system (at the James Reserve) and
Figure 5.10: Estimation of PAR fluctuations with event generation densities by both policies.

(a) Distance policy. Sensor 5  
(b) Time policy. Sensor 5

Figure 5.11: Event generation and servicing by both policies. Note that in some cases events are generated at different times and the OnTime of some events varies depending on a policy.
Figure 5.12: Comparison of an average OnTime and OnTime in a form of a zero-mean Gaussian distributions for Time and Distance policies (p). Three different thresholds (t) used. The OnTime of events generated by all sensors is considered.

Figure 5.9 shows prerecorded sensor strand PAR measurements for all three strands (six sensors). We use this data for all of the following experiments. We conducted experiments for two policies (Time and Distance), for three different thresholds (10, 25, 50). A threshold is used by our system to determine when to trigger an event (a task) and is a measure in units of PAR.

Figure 5.10 shows data from Sensor 5 (third strand) and generated events by Distance policy (Figure 5.10a) and Time policy (Figure 5.10b). Note that both policies generate events so that the spikes in the PAR data are covered, which in turn means that each of those spikes can be sampled by the system. Figure 5.10 also shows a good spatiotemporal phenomenon 'coverage' by both policies.
Figure 5.11 shows a magnified view of a part of the data from sensor 3 (second strand). Generated and serviced events are drawn on this figure, for Distance policy (Figure 5.11a) and Time policy (Figure 5.11b). Note that at certain points in time, due to inherent differences between the two policies we consider, some events are generated by one policy and not generated by the other. As a result, the OnTime for same generated events is different.

Finally, Figure 5.12 shows a comparison of the performance of both policies as a measure of cumulative event OnTime. Figure 5.12a shows the change in average OnTime for different values of the threshold. It follows that OnTime becomes smaller (the system responds to events faster) with bigger threshold values. This result is expected, however - the smaller the threshold, more events are generated and hence the system spends more time to service all events. Figure 5.12a also shows that there is no significant difference in performance between the two policies. If we consider Figure 5.12a representing the comparison of average values of the policies, or the means, then Figure 5.12b shows the comparison in deviation of both policies from the mean. Figure 5.12b also does not show significant difference between the two policies.

The results of this experiment show that our Task Allocation system, based on a greedy algorithm, is well suited for the task of spatiotemporal sampling. In addition, the difference between the two task prioritization policies is relatively small, which is due to the fact that distance policy can be expressed in terms of the time policy and vice versa.

5.3.3 Multi-Robot System

We conducted experiments with multiple simulated NIMS systems and used a set of sensor strand data (approximately 9 hours and 30 minutes) spanning one day. Then we replayed the strand data
through the same interface as in the field system in simulation, varying the number of robots between 1 and 5.

Figure 5.13a shows the average OnTime as the number of robots increases from 1 to 5. Note that the performance of the TA system improves with increasing number of robots, but approaches a limit at point of 6 robots (not shown). In the case of 6 robots, at some point of time one robot is positioned around each of the static sensor nodes, and hence responds to the event immediately. Hence, there can be no improvement in the performance in systems consisting of \( > 6 \) robots.

Figure 5.13b shows the number of serviced events as the number of robots increases from 1 to 5. Note that as the number of robots increases, the number of events serviced by the system of robots increases as well. However, this increase has a decaying slope. As in the previous case, when there are 6 robots in the system, each of the robot eventually occupies area around each of the sensors. Hence, every event that is generated, can be serviced by the system.
This experiments show a noticeable improvement in performance and increase in the number of serviced events as the number of robots increases in the system. This improvement is bounded, however, by the number of static sensors in the system - 6 in our case.

5.4 Summary

In this chapter we presented Networked Info-Mechanical System (NIMS). NIMS is an example of a symbiotic system, which is deployed in a natural habitat and used for monitoring of spatiotemporal phenomena such as PAR. We have implemented our task allocation system on NIMS. This system includes supporting cable infrastructure, a horizontally moving mobile robot (the NIMS node) equipped with image sensing, and a vertically mobile meteorological sensor system carrying water vapor, temperature, and PAR sensing capability. In addition to the mobile node, NIMS also includes a set of statically positioned sensor nodes, arranged in strands.

The sensor strands discretize the sampling transect into sampling areas, where each static node is responsible for sensing the phenomenon in its local area of regard. If static node’s sensor value reaches a predetermined threshold, a task (or an event) is created, and the system is notified of this event. Next, the system allocates the most fit available robot to service the event. The robot’s task then is to navigate to the location of the static node and sample the area around this node to gather more precise data about the phenomenon.

We conducted a number of experiments both on the real NIMS deployed at the mountain reserve and on NIMS test systems in the lab. First, we compared the performance of our task allocation system versus a traditional Raster Scan method. Raster Scan method patrols every point of the environment in search of the events. Our experiments show that our task allocation
system implemented on NIMS significantly outperforms Raster Scan, that is a method of choice for the problem of spatiotemporal sampling. Moreover, our system also consumes less energy while accomplishing the task as indicated by Figure 5.6.

Next, we performed a set of experiments on the real NIMS and on NIMS test systems with replayed data from the real system. We compared two policies for prioritizing events - time and distance policies. The experiments show that our task allocation system achieves adequate spatiotemporal sampling. In general, the distance policy produced slightly better results than the time policy, however both policies aim at optimizing the response time of the system, and hence the difference is not significant.

Finally, we performed a set of experiments examining the change in performance of our task allocation system as the number of robots in the system varies between 1 and 5. The experiments were performed by replaying the sensor data, obtained from the sensor strands of the real NIMS, in simulation. The results show a clear improvement in performance as the number of robots increases. However, the improvement limits at the point of 6 robots in our setting. This result is expected, because in the case of a 6-robot system, every robot would be positioned in the area of every sensor (6 in our case), and hence, every event is serviced right away. Adding more robots would not improve results because they would not be assigned to any event.

The main result of this chapter is that methodologies proposed in this dissertation can be implemented on real systems and are adequate for the problem of spatiotemporal sampling.
Chapter 6

Conclusion

In this thesis we presented a solution for the problem of monitoring spatiotemporal phenomena in an unknown, unstructured, dynamic environment using mobile robots and a sensor network. Our thesis is that a symbiotic system combining the capabilities of mobile robots and a static sensor network efficiently provides the ability to monitor spatiotemporal phenomena at high fidelity in an unknown, unstructured, dynamic environment.

In order to establish our thesis we validated the three hypotheses:

Hypothesis 1 [Coverage and Exploration through Sensor Network Deployment]:

*There exists an efficient, robust and scalable algorithm to embed an active infrastructure (sensing, communication and computation) into the environment while simultaneously using this infrastructure for coverage and exploration.*

Hypothesis 2 [Network mediated Robot Navigation]:

*An embedded sensing and communication network of static nodes can be used to guide the robot navigation.*
Hypothesis 3 [Network mediated Multi-Robot Task Allocation]:

An embedded sensing and communication network of static nodes can efficiently mediate multi-robot task allocation to enable high fidelity spatiotemporal sampling.

Hence, we proposed solutions to three problems: coverage and exploration through sensor network deployment, network mediated robot navigation and network mediated multi-robot task allocation.

In order to validate the first hypothesis we presented an algorithm (LRV) that solves the dynamic coverage and exploration problem through network deployment and repair. The proposed algorithm is provably complete, efficient, decentralized, scalable, robust, fault tolerant and can be used on simple robots. We verified the performance of the proposed algorithm and its asymptotic behavior in theory and in simulation. There exists a direct correspondence between the results obtained from the theoretical analysis (coverage on the graph) and the data from simulation experiments.

The second hypothesis was established next. Once the network is deployed it can be used for a fundamental capability of mobile robots - navigation. We presented an algorithm that allows the robot to navigate precisely and reliably using a deployed sensor network. Our approach differs from systems described in the literature by assuming that a map, localization or GPS are not available. The navigation approach can be used by simple (modest computation, communication and sensing requirements) heterogeneous robots. The navigation occurs through node-wise motion from node to node on the path from starting node to the goal node. Extensive experimental trials have been conducted to test this algorithm.
The final problem for efficient spatiotemporal phenomena monitoring, as stated in the third hypothesis, is multi-robot task allocation. Our solution is to use the deployed sensor network to detect the phenomenon locally and assign and navigate robots to inspect it. Multi-robot task allocation algorithms can also be used for efficient network repair and maintenance. We introduced DINTA: Distributed In-Network Task Allocation and DINTA-MF: Multi Field DINTA for solving this problem. Both compute task assignments distributively in-network while, at the same time, providing a virtual sensor and communication device that ‘extends’ throughout the whole environment. The difference between approaches is that DINTA assigns tasks implicitly, whereas DINTA-MF explicitly determines which robot serves which task and hence performs generally better and does not waste resources. There are several advantages in using DINTA and DINTA-MF as opposed to other MRTA approaches. The sensor network allows a robot to detect an event even though the event is not in robot’s sensor range. In addition, mobile robots can use the network to relay messages if they are not within communication range of each other. One of the other benefits of using DINTA (or DINTA-MF) is distributed in-network computation, which 1. avoids redundant computation by updating the state of a node based only on the state of its neighbors and robots in the vicinity (scalability), 2. computes utilities in the network distributively and propagates from the goal state. Another benefit is the ease of determining relative distance to the goal (for determining utilities) by considering hop counts from the goal node. Note also that the robots implementing these proposed approaches can be quite simple - they do not need to localize and map the environment - they can navigate by following the suggestions from the sensor network.
We implemented our task allocation methodologies on a new kind of system - Networked Info-Mechanical System (NIMS). This system includes supporting cable infrastructure, a horizontally moving mobile robot (the NIMS node) equipped with image sensing, and a vertically mobile meteorological sensor system carrying water vapor, temperature, and PAR sensing capability. In addition to the mobile node, NIMS also includes a set of statically positioned sensor nodes, arranged in strands.

The sensor strands discretize the sampling transect into sampling areas, where each static node is responsible for sensing the phenomenon in its local area of regard. If static node’s sensor value reaches a predetermined threshold, a task (or an event) is created, and the system is notified of this event. Next, the system allocates the most fit available robot to service the event. The robot’s task then is to navigate to the location of the static node and sample the area around this node to gather more precise data about the phenomenon.

We performed extensive experiments in real environment (the James Mountain Reserve), on a lab-scale platform (NIMS-LS) and in simulations using hardware in the loop and/or data from the real system. We showed that NIMS system, combining mobile and static sensor nodes and following our methodologies, outperforms mobile robot-based (sampling using raster scan) or static network-based (sampling using static network) approaches. Further, experimental results verified that the proposed system can effectively estimate spatiotemporal phenomena and efficiently mediate multi-robot task allocation to enable high fidelity spatiotemporal sampling. As a result, we further validated hypothesis 3 and our thesis in general.

An important quality of the proposed system is its robustness. Note that the system can solve the problems of coverage and exploration, navigation and multi-robot task allocation without global information about the environment (no GPS, map, localization). As a result, the system
is robust against perturbations in the environment as long as sensor network topology remains the same. If network topology changes physically (network can not repair itself), the system can always fall back on the most basic subsystem - coverage and exploration through sensor network deployment (LRV). LRV is provably complete, which guarantees that eventually the robot will visit every node of the network. LRV can also repair the topology of the network as required. Note that in this case we trade off the efficiency of the navigation and task allocation algorithms for robustness of LRV. Hence, the system would be able to complete its task even in the event of severe perturbations. Finally, when LRV restores the topology of the network and accompanying transition probabilities, the other subsystems can start working again (navigation and multi-robot task allocation).

This thesis proposes a novel symbiotic system consisting of mobile robots and a sensor network. We showed that such a system addresses problems (e.g. spatiotemporal monitoring) that are impossible or infeasible to solve by traditional means. We also showed that our system can efficiently, reliably and robustly solve traditional problems in robotics (e.g. coverage and exploration, navigation and multi-robot task allocation) and sensor networks (e.g. deployment, maintenance and repair). We demonstrated that in our system robots rely on a deployed active infrastructure and, thus, do not require global information about the environment (GPS, map, localization, etc). Hence, the robots can be minimalist or can dedicate their resources to other tasks (e.g. data processing, mission planning, etc.).

Note that our system provides an online solution to the problem of spatiotemporal sampling - a solution which assumes that domain information about the phenomenon (e.g. model) is not available. An ability to efficiently characterize spatiotemporal patterns of the phenomenon of interest allows us to study the dynamics of that phenomenon as well. At the same time, further
improvement of the system’s performance and the fidelity of the spatiotemporal characterization of the phenomenon (phenomenon dynamics) is possible with modeling. Phenomenon modeling is a difficult problem in general. However, the proposed system provides spatiotemporal phenomenon data which can be used for creating and updating the model of the phenomenon as well as for tuning the parameters of the model. Hence, the proposed system lays the foundation for approaches that use/learn models. In addition, the proposed system can also be used to characterize the dynamics of the environment. For example, perturbations in the environment topology can be characterized (i.e. mapped or modeled) as well. Studying environment dynamics is important because it 1. can be used to improve system’s performance in general and 2. aids phenomenon modeling. Hence, the proposed system lays the foundation for approaches that study the dynamics of the environment.

This thesis thus describes a new symbiotic system that combines both fixed and mobile sensor nodes to achieve a spatiotemporal environment coverage that is dramatically advanced over that of either system alone. Mobility allows the networked sensor system to always seek the most efficient spatiotemporal sampling distribution to achieve environmental variable reconstruction. Further, mobility also permits the symbiotic system to respond to initially unpredictable and variable environmental evolution.

We have identified modeling of the phenomenon and of the environment dynamics as the two areas for future research that can use the proposed system as a foundation. At the same time we would like to note that several additional problems can be addressed by using proposed system at its current state. For example, we considered a planar 2D environment for spatiotemporal monitoring, however certain real life problems (e.g. monitoring harmful algal blooms in the ocean) require monitoring of phenomena in 3D. The proposed system, however, is easily extensible to
the 3D case. Another example is characterizing spatiotemporal patterns of highly dynamic phenomena. In this case, task allocation overhead during the computation stage might not provide fast enough response time. To address such a problem, LRV can be used to sample areas of the environment (covering the environment), guaranteeing complete coverage and striving to provide linear performance.
Reference List


Appendix A

Publications


