Algorithms for Distributed Sensor Networks

James D. McLurkin
University of California at Berkeley
Berkeley Sensor and Actuator Center

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Kristofer S. J. Pister (Date)
Research Advisor

Professor John F. Canny (Date)
Second Reader

Algorithms for Distributed Sensor Networks

By
James D. McLurkin

Submitted to the Department of Electrical Engineering and Computer Science on
December 16, 1999 in partial fulfillment of the requirements for the degree of
Master of Science
University of California at Berkeley
Berkeley Sensor and Actuator Center

Abstract

A distributed sensor network is many (100-10000) autonomous sensor nodes spread out over a large area. Each node is equipped with a processor, mission-specific sensors, and short-range communications. Local interactions between sensor nodes allow them to reach global conclusions from their data. This work develops algorithms that allow:

- The group to establish robust spatial patterns of messages
- The group to develop a communications network by dividing tasks among themselves
- Each mote to determine its position in physical space based on their location in the network topology
- Each mote to determine if it is on the boundaries of the network by measuring global constants through local interactions
- The group to project the path of a target moving through the network

To verify our algorithms, we have constructed two simulation environments. One is based in software and allows for very rapid proof-of-concept development. The other is a hardware version that still allows rapid development, yet provides all the problems of real hardware for a high fidelity simulation.

To Mom and Dad

For investing so much in my future

Algorithms for Distributed Sensor Networks

A Master's Thesis by James D. McLurkin
With 29 Illustrations
(and 30 Equations!)

Table of Contents

Abstract i
1 Introduction 1
1.1 Algorithms for Sensor Networks 2
1.1.1 Sensing 2
1.1.2 Processing 3
1.1.3 Act 3
1.2 weC: A Software Simulation 4
1.3 Macromotes: A Hardware Simulation 5
2 Background 6
2.1 "Smart Dust" 6
2.2 Applications 6
2.3 Smart dust Engineering Issues 7
2.3.1 Communications 7
2.3.2 Power Management 7
2.3.3 Distribution Techniques 7
2.4 Biological Sensor Networks 7
2.4.1 Ants 8
2.4.2 Bees 8
2.4.3 Coral 8
2.4.4 Sponges 8
3 weC: A Software Simulation 9
3.1 Introduction 9
3.2 Definition of Conventions 9
3.3 Assumptions 10
4 Distributed Algorithms 11
4.1 The Smart Dust Lexicon 11
4.1.1 Pheromone Message 11
4.1.2 Agent Message 12
4.2 Smart Dust Memory Requirements 12
4.3 Pheromone Message Diffusion 13
4.3.1 Graph Theory of Systems of Pheromone Messages 14
4.3.2 Robustness of Systems of Pheromone Messages 15
4.3.3 Algorithm Execution Time 17
4.4 Directed Communication 19
4.5 Heterogeneous Task Allocation 19
4.5.1 Relay Network Task Allocation 20
4.5.2 Finding an Optimal Distribution of RelayMotes 21
4.6 Determining Physical Position from Network Topology 28
4.7 Physical Distributions and Network Topology 32
4.7.1 Neighbor Counts 32
4.7.2 Edge effects 33
4.7.3 Edge Detection 33
4.8 Path Projection 39
4.8.1 Failure Modes 40

4.9 Algorithm Summary 41
4.10 Simulation Operation 41
5 MacroMotes: A Hardware Simulation 44
5.1 Hardware Overview 44
5.2 Algorithms 45
6 Conclusions and Future Work 46
6.1 Design Philosophies 46
6.1.1 Scalability 46
6.1.2 Robustness 46
6.2 Future Work 46
6.3 Conclusion 48
7 Appendices 49
7.1 Appendix A: weC code 49
7.2 Appendix B: Macromote Parts List 49
7.3 Appendix C: Macromote Schematic 50
8 References: 51

Note: This document is much better when read in **color**. Visit www.eecs.berkeley.edu to get your very own .pdf file. Color printer not included.

Table of Figures:

Figure 1: Overview figure of all the algorithms presented in this work..... 1
Figure 2: The weC simulation environment..... 4
Figure 3: A macromote, a cubic-inch sensor node..... 5
Figure 4: System diagram of a cubic millimeter sensor node..... 6
Figure 5: Picture of a micro aerial vehicle that can be used to dispense smart dust sensor nodes..... 7
Figure 6: Pictorial conventions used in this work..... 9
Figure 7: Pheromone message diffusion algorithm explanation..... 13
Figure 8: Pheromone message diffusion generates a tree graph..... 14
Figure 9: Pheromone message diffusion simulation output..... 15
Figure 10: Algorithm execution time explanation..... 18
Figure 11: Relay network task allocation – Heterogeneous division of labor..... 20
Figure 12: Centralized algorithm RelayMote selection output compared to distributed algorithm output..... 22
Figure 13: Relay network formation removes probabilistic links from graph..... 22
Figure 14: Relay network algorithm formation cannot always find one upstream RelayMote in two dimensional networks..... 24
Figure 15: One-dimensional relay network simulation output..... 25
Figure 16: Two-dimensional relay network simulation output..... 26
Figure 17: Basins of attraction for swystems with mulriple ChairMotes..... 27
Figure 18: How physical separation relates to network hops – the line straightness constant..... 28
Figure 19: Position estimation using the maximum distance between the BasisMotes and the MeasureMote - explanation..... 29
Figure 20: Position estimation using the maximum distance between the BasisMotes and the MeasureMote – simulation output..... 30
Figure 21: Position estimation using the average distance between the BasisMotes and the MeasureMote – simulation output..... 32
Figure 22: Edge effects and their effect on local neighbor count..... 33
Figure 23: Edge detection simulation output..... 35
Figure 24: Effects of neighbor count on edge detection..... 37
Figure 25: Measuring the edge detection threshold constant..... 38
Figure 26: Path Projection using initial sightings to estimate a target's path through the network – explanation..... 39
Figure 27: Path projection "Vector Field" and simulation output..... 40
Figure 28: A macromote..... 44
Figure 29: Macromote hardware block diagram..... 44

Acknowledgements

Kris Pister - They simply don't make advisors better than this. Get 'em while their hot!

Shontaye McGriff - My wonderful, supportive, loving, "totally awesome" girlfriend.

James Dwight McLurkin III (AKA Dad) – For all the technical teachings from way back when.

Sandra Lawson (AKA Mom) - For all the teachings on everything else.

Anita Flynn (AKA Lab Mom) – For still being my UROP advisor after all these years.

Lance Doherty - My "Spiritual Math Guide" who demonstrated considerable patience while I whined about how much math I can't do, all while helping me to do more of it.

Shankar Sastry – Thanks to my "Grand Advisor" for wise words and encouragement

John Canny – For being willing to adopt a computer scientist stuck in the body of an electrical engineer.

Soichiro Honda - For building things that go fast and make me happy.

1 Introduction

Distributed sensors networks are comprised of large number of simple autonomous sensor nodes. Each node contains a processor, sensors, short-range communications, and a power source. These nodes are distributed throughout the target area, form a local communications network, and look for the desired targets. This work describes distributed algorithms designed to produce global results from the local interactions of thousands of sensor nodes.

The ability to accomplish your sensing task with large number of simple elements has several advantages:

- The overall system can be more robust to the failure of any one, or any half of the individual elements.

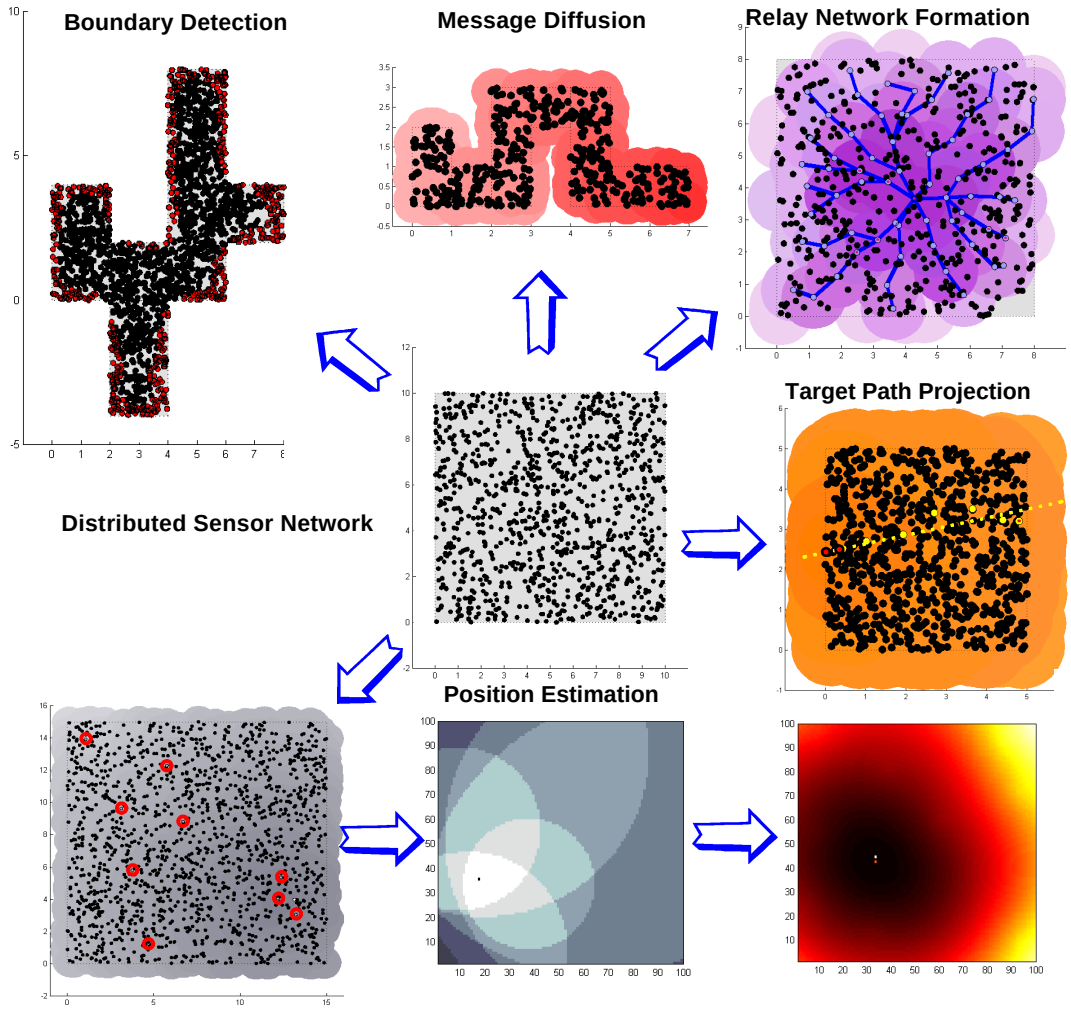


Figure 1: The distributed algorithms developed in this work. Edge Detection allows individual sensor nodes to determine if they are on the edge of the workspace. Message diffusion provides robust, stable, spatially correlated distributions of messages. Relay Network formation allows for heterogeneous task allocation. Position estimation lets an individual sensor node compute its position based on network topology.

- The individual elements can be made very small. This opens up new possibilities for sensors inside small spaces.
- The individual elements can be made very cheaply. 1000 cheap things might be cheaper than 1 expensive thing.
- The total coverage of the network can be spread out across areas much larger that any individual's sensory or communication range.

Programming a large number of computational elements requires different techniques and algorithms. This work describes several algorithms designed for sensor networks:

Pheromone Message Diffusion

Pheromone messages are used to set up and maintain robust, spatially correlated distributions of messages within the network. These messages are initiated from a source node and then relayed from node to node throughout the network. Intelligent software agents with the ability to transfer themselves from sensor node to sensor node can use the data contained in these messages to guide themselves around the network.

Relay Network Formation

In some applications, it is useful to divide the sensor network into functional groups. By monitoring communications usage within the network, individual nodes become communications hubs, leaving the other nodes free to spend all their time sensing. The distribution of tasks is robust to network topology changes.

Position Estimation

The user of the sensor network would often want to know where particular events are occurring. Each sensor node can analyze its connections in the network topology relative to special nodes which know their position. From this topology data, every node can estimate its own position.

Edge Detection

Sensor nodes that are on the boundaries of the workspace are able to detect targets that are new to the network and inform other nodes when old targets leave the network. Algorithms to determine whether a particular node is on the edge of the network are presented.

Path Projection

Sensor nodes tracking targets moving throughout the network can send directed communications to other nodes that lie in the linear projection of the target's path. This can be accomplished in a distributed fashion, without the use of previously mentioned position information.

1.1 Algorithms for Sensor Networks

The mission for a distributed sensor network can be divided into three main tasks: Sensing, Processing, and Acting.

1.1.1 Sensing

The sensory suite for a real-life sensor node would depend on the application it was designed for. Bio-chemical sniffer-nodes would need olfactory input, while rodent-detectors could use thermal, motion, or acoustic cues.

1.1.2 Processing

A system comprised of a large number of computational elements can process information in two distinct ways, explicitly within each node or implicitly through inter-node communication. Traditional algorithms use the explicit computational model. Each sensor node analyzes its sensory inputs and forms a conclusion independently of its neighbors. This approach uses standard algorithms, but does not take advantage of the information sensed by the surrounding nodes or leverage their computational resources.

In a distributed computational model, each node is still responsible for performing the initial processing of its own sensory information, but the output of the algorithm is formed as a result of each node communicating with its neighbors. This allows the group to draw conclusions from the data that a single sensor node could not do. For example, if the same target is detected by several nodes, it can be classified from its sensory signature by each node independently, but by working together, they can estimate its size and heading.

1.1.3 Act

When the processing of sensory data is complete, the network should act. Our network can act by communicating with the user or by reallocating its resources. Efficiency and sensitivity can be tuned to optimize the overall performance of the network. As robotic research progresses, distributed sensor networks will eventually turn into distributed robotic networks [1,2], in which case acting on a stimulus could have many diverse responses.

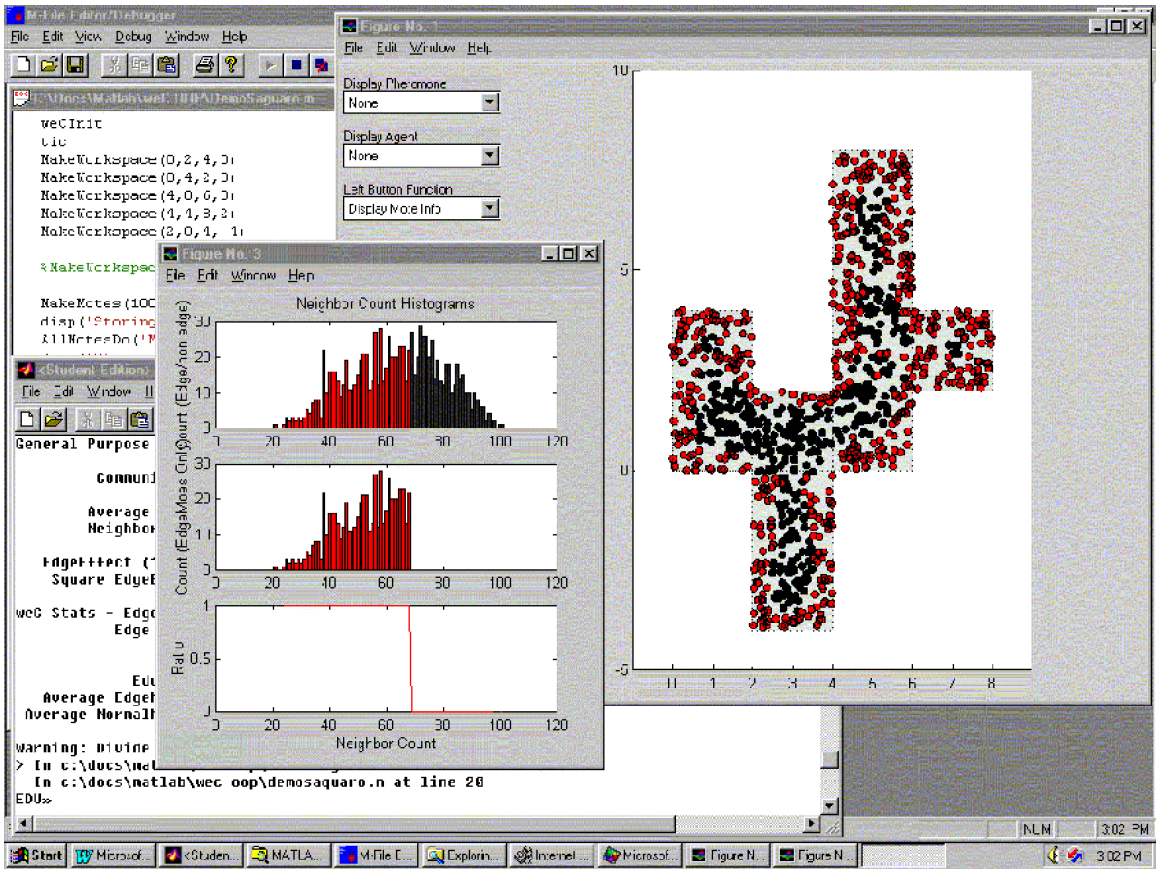


Figure 2: The weC simulation running in MATLAB. The window on the right is displaying the physical positions of the Smart Dust sensor nodes. The different colors indicate what each node is doing. In this example, nodes that are colored red think they are at the perimeter of the workspace The chart in the middle shows a histogram of nodes, colored by job. Some code can be seen in the background. I call this program "DemoSaguaroEdge".

1.2 weC: A Software Simulation

Programming a large community of autonomous agents is a daunting task. The goal of weC (Pronounced We-See or Wee-C) is to provide a layer of abstraction and a toolbox of algorithms for the software design of such a system. The pinnacle of evolution of weC would be a language that would allow the programmer to specify abstract, high-level goals for the whole community, and then the compiler would spit out low-level software for the individual agents. Solving this problem appears to require the solution to a related problem, artificial intelligence. This is hard.

Falling somewhat short of that grand plan, the goal of this work is to identify and test useful techniques for programming networks of distributed sensors. Although these techniques should apply to many distributed systems, our examples are all in the context of a sensor network.

1.3 Macromotes: A Hardware Simulation

While simulations are useful for illuminating specific aspects of a problem and for exploring many different ideas rapidly, they are no substitute for physical hardware. Macromotes are overgrown sensor nodes that let us test our algorithms on a real system. They are about one cubic inch in volume, which makes them useful in many real-world applications in addition to thesis software development.

The macromote designed for this work is shown in Figure 3. It is based on the many designs of our group [19]. Each one contains a light sensor, a temperature sensor, two microcontrollers, a radio transceiver, and a battery. There are three LEDs on top that let the user determine which task that node is performing. The ability to reprogram the entire network remotely is important, so the second processor was included to reset and reprogram the main microcontroller.

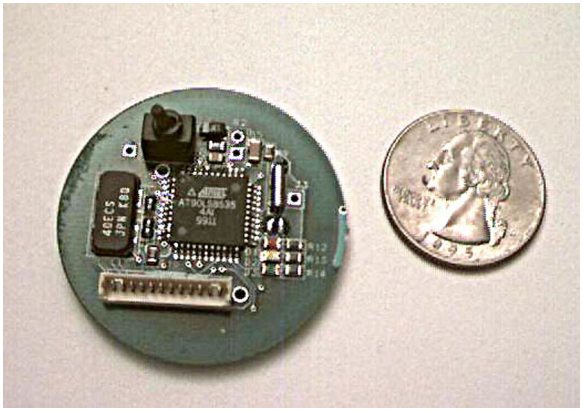


Figure 3: A Macromote. These devices were designed to be a "hardware simulation" of a more practical Smart Dust network. As such, their capabilities are rather limited. This one is equipped with two processors, a temperature sensor, a light sensor, a radio transceiver and a battery. The dual processor architecture allows the user to reprogram the entire group remotely. This will shorten development cycles considerably.

2 Background

With current technology, it is easy to fit a processor, sensors, communications hardware, and a power source into a cubic inch of volume. Cubic centimeter packaging goals are within sight, and cubic millimeter levels of integration will certainly follow. Writing software for a computational system that could be sprinkled across the countryside requires new models and assumptions. This work explores algorithms for forming a sensor network from these tiny pieces of hardware, affectionately known as "Smart dust".

2.1 "Smart Dust"

The goal of Kristofer Pister's Smart Dust Project at U.C. Berkeley is to turn a cubic millimeter of silicon into an autonomous sensor node. Each particle, or mote, of Smart Dust will have a processor, a sensor suite, batteries, and communications as depicted in Figure 4. They may be thought of as sessile robots with local communications, or "insects with their legs pulled off" [3]. Their principal application will be surveillance and other types of information gathering. This application would require distributed sensor arrays of hundreds, or thousands of individual motes. Distributed algorithms and design paradigms for a network of this type is a topic of open research, and the focus of this work.

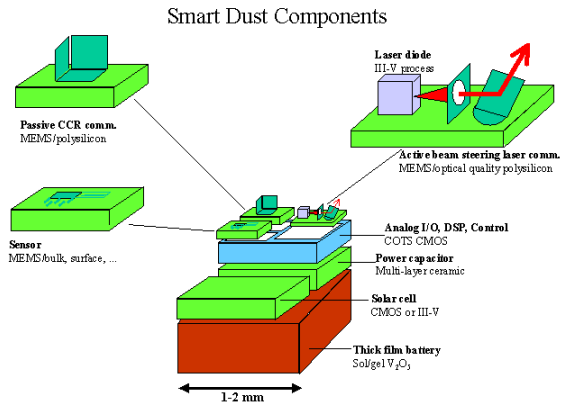


Figure 4: A mote of Smart Dust* This cubic millimeter of silicon comes equipped with a microprocessor, sensors, communications devices and a power supply. They would be cheap enough to allow thousands of them to be distributed throughout an area. Although their individual abilities might be limited, the effectiveness of the group would be much greater that the sum of its parts.

*Not yet available in all areas

2.2 Applications

Smart Dust can do anything! Here is what we have come up with so far. Feel free to make up your own applications as you read along.

Department of Defense:

Battlefield sensor networks
Sensor burrs, maple seeds and dandelion seeds
Enemy traffic pattern observation
Bunker mapping
Biological /Chemical weapons detection and marking

Civilian:

Security systems
Non-invasive surgery
Traffic management/traffic jam avoidance
Tracking professors
KPS: Kid Positioning System for hyper pre-teens and their overprotective parents
Vermin monitoring and extermination

A tactical advantage for hard-core paintball players

Scientific

Wildlife conservation and monitoring
Underground/in nest/in burrow/in body real-time observation
Social insect community monitoring on an insect-by-insect basis

2.3 Smart dust Engineering Issues

The limitations of physical hardware can have a strong influence on algorithm design. Important issues to smart dust networks are communications, power management, and the distribution technique employed.

2.3.1 Communications

Because the communication system is now an integral part of the computational process, details of its implementation are of greater importance. A network with a communication model based on omnidirectional broadcast will spread, and subsequently process, information differently from a network with directed communications. The correlation between communication effectiveness and spatial relation can be very important. For example, this work assumes short-range radio links, which means that if two motes can communicate with each other, they are physically close to each other. This assumption might not be true with a laser-based communication system due to constraints on aiming the emitter.

2.3.2 Power Management

Smart dust motes have a meager power budget. Their energy comes from sources such as solar cells, small batteries, and thermopiles. These are not high power sources, so conservation of energy is important. Well-designed hardware will minimize each individual mote's energy consumption. Distributed algorithms might allow for even more power conservation. For example, by interacting with their neighbors and allocating tasks heterogeneously, individual motes can devote all their energy to one particular task.

2.3.3 Distribution Techniques

The dust motes can be distributed by hand, plane, unmanned micro aerial vehicle (see Figure 5), microrocket, or slingshot. They can range in form from simple cubes of silicon to floating dandelion seeds or autorotating maple seeds.

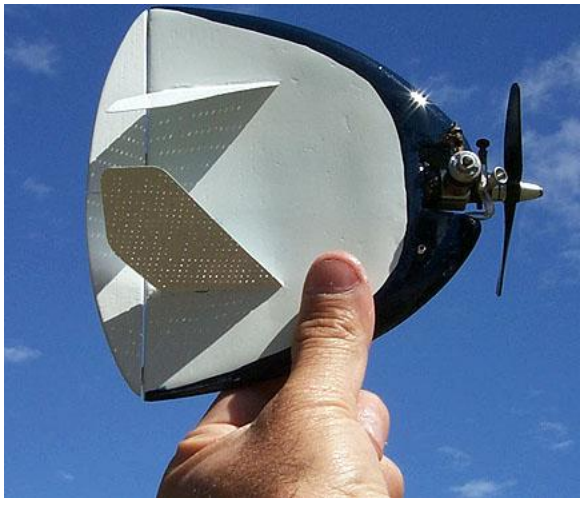


Figure 5: A micro air vehicle (MAV) built by MLB Co. [4]. This six-inch wondertoy comes with a video camera and a dispenser for cubic-centimeter sized sensor nodes. At the time of this writing, it can hit 60 m.p.h. and stay aloft for 18 minutes! Swarms of these tiny craft could rain Smart Dust upon the target area.

2.4 Biological Sensor Networks

We cannot claim rights to the idea of a network made up of distributed sensors. Nature beat us to that by many millions of years. Much of the inspiration for the algorithms presented here came from natural examples.

2.4.1 Ants

Ant navigation by scent trails is well explored in the literature. [5,6,7] Differential reinforcement of competing scent trails will ensure that the colony as a group will always exploit the closest source first. The diffusion algorithm from section 4.3 was modeled after the scent trails worker ants follow.

2.4.2 Bees

Bumble bees and honey bees [8, 9] are more remarkable examples of natural organization. Bees actively monitor resources in their environment, spread information throughout their colonies in distributed fashions, and divide labor between their workers.

2.4.3 Coral

Jellyfish, Corals, and Sea Anemone all have simple nervous systems called nerve nets. The relay network formation algorithm in section 4.5.1 came about from wondering if I could get a set of sensor nodes to differentiate into a nervous system.

2.4.4 Sponges

Sponges are surprisingly complex to be referred to as “simple animals” [10] Among their many different cell types, they have amoebocytes, which roam around the body of the animal sharing food and information. These cells inspired the software agents that roam our distributed sensor network.

Nature is filled with examples of distributed systems, indeed, it seems to be a primary design principle. Examples of local interactions forming global patterns are found everywhere.

3 weC: A Software Simulation

3.1 Introduction

The weC Simulator was designed for rapid development of distributed algorithms. A sample screen shot is shown in Figure 2. MATLAB [11] was chosen as our development environment for two main reasons. It is "the language of technical computing" and as such has many built-in functions and features that ease algorithm design and data analysis. It is also a very popular language among engineers, so the source code is accessible to a wide audience.

The goal of this software was not to make a high fidelity simulation of the complete smart dust environment, but rather to have way to investigate distributed algorithms. Our high fidelity simulation is comprised of a network of macromotes like the one shown in Figure 3. "The world is its own best model" [12] and should be used as such whenever possible.

3.2 Definition of Conventions

The graphical output of the simulator is somewhat meager, consisting of little circles and big circles, just like the ones shown in Figure 6. We compensate for this somewhat by drawing these circles in beautiful 32-bit color, but all that effort was not appreciated by the 1-bit photocopier that probably produced the document that you are now reading. If you want to see the pretty pictures in all their glory, visit http://www.eecs.berkeley.edu/ and download this document. The smart dust motes are represented by the small circles. Their color depends on the job they are performing. The communications range is depicted by a larger circle surrounding each mote. Any other dust motes who lie within the bounds of their neighbor's communications circles will be able to talk to each other. The brightness and color of the communications circles depends of the level and type of *pheromone message* currently displayed. Some simulations show very small red dots inside the motes. These are *agent messages*. The impatient reader can skip ahead to section 4 for a description of these terms.

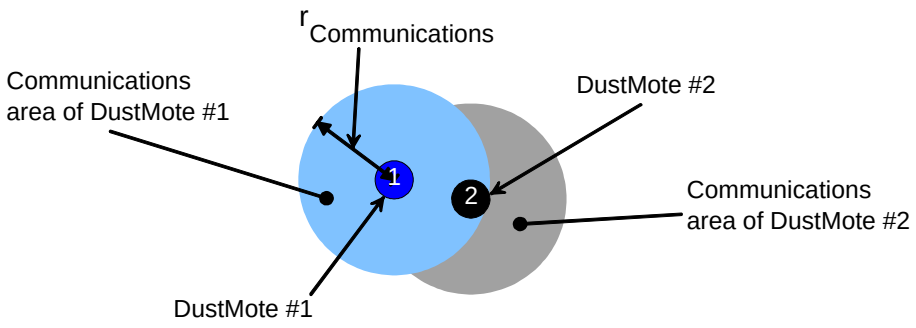


Figure 6: weC Pictorial conventions. The blue circle (Mote #1) is a dust mote that is broadcasting a message. The black circle (Mote #2) is receiving the message. The number in each mote is the unique Mote ID. Assume that each mote has a unique ID, even if they are not drawn. The larger circles around each mote represent its communications range. The color and brightness of each communications circle represent a Pheromone Message name and level, respectively. For example, the mote #1 might be transmitting: "Message Name=Blue Level=50". The next section describes pheromone messages in more detail.

3.3 Assumptions

All good scientific work must begin by simplifying the initially-interesting-but-hopelessly-intractable problem into one that can actually be solved in finite time. As such, an impressive list of assumptions immediately follows:

1. The world is flat. No, Magellan, I'm not asserting this in general, just on the scale of a single dust mote. For example, imagine thousands of motes distributed on 100 m^2 of rugged terrain. As long as the dimension of the ruggedness is small compared to 100 m, the smart dust network will be essentially two-dimensional. Two examples of a truly three-dimensional system where this assumption would not hold is a bunch of dandelion-motes dropped from an airplane, or a school of plankton-motes living in the water.
2. Homogeneous hardware. This allows for any mote to act like any other mote if they like. Phrased differently, the only differences between the motes are position, sensory input, and software state.
3. A communications systems that does not fail all of the time, or have systematic failures. As long as errors are random and packets can get through eventually, the distributed algorithms will still work. The dust motes could use TDMA, CDMA, or random retries. Collisions are allowed, as long as neighbors can hear each other every now and then. A 50% communication error rate should make a well-designed distributed algorithm take about twice as long, not crash and burn.
4. The communication system broadcasts omnidirectionally. This leads to a correlation between receiving a communication signal and your physical location relative to the sender. Furthermore, I assume that there is an average radius of effective communications that is relatively constant among all the motes
5. There is no range or directional information received with any communications signal. This constraint is based on the assumption (and personal experience) that the hardware that works will be very simple.
6. Each mote has a globally unique ID code. There are many algorithms that will work fine without unique IDs but there are some that require them, such as the network formation software from section 4.5. At this time it is not clear what classes of algorithms do not require IDs, but we can be sure that they form a proper subset of the algorithms that do need them. We assume that we have IDs so as not to limit ourselves.
7. For some of the algorithms presented, a uniform spatial distribution of motes is assumed. This is the most unrealistic assumption of the bunch, but it allows for a simplification of some of the mathematics involved. (read: doable vs. non-doable). Ideas for generalization to non-uniform distributions are suggested, but not rigorously considered.
8. The dust motes are stationary, or move slowly relative to the algorithm update time. The constraint "slowly" will require more careful treatment. Section 4.3.3 discusses algorithm execution time and real world events.

4 Distributed Algorithms

4.1 The Smart Dust Lexicon

Dust Motes talk to each other using two different types of communications packets, pheromone messages and agent messages. These are just exciting names given to rather boring streams of bits flowing from mote to mote, so of you prefer to think of them simply as communications packets, go right ahead.

4.1.1 Pheromone Message

Pheromone messages are used to set up and maintain spatially correlated distributions of messages within the network. These messages are initiated from a source mote and then relayed from mote to mote throughout the network as shown in Figure 7. These gradients can be used for computation or to guide agent messages around the network. Pheromone messages keep track of how many times they have been relayed from mote to mote. This allows messages to propagate and "flood" the entire network, or create local "puddles" around the source mote. For example, if one mote detects an interesting sensory stimulus, it could alert its neighbors by transmitting a pheromone that would decay to zero and no longer be relayed after two communication hops. Or, if one mote wants to determine how many hops it is away from another one, it could transmit pheromone messages and wait for an "echo" from the targeted mote. Any mote that broadcasts a pheromone message can then be targeted by the recipients to receive replies. This is useful when a user is communicating with only one mote but wants data from the entire network.

Each dust mote keeps a list of all the pheromone messages it has received, indexed by name. If a message with a duplicate name is received, then the **`ReplacementOperation`** parameter of the new message is used to determine which one is kept. If the level field of any pheromone falls below zero, that message is purged from the mote. The complete message format follows:

4.1.1.1 Pheromone Message Data Fields:

`Name:` The name for this message. This is a text string, i.e. "Funk" from Figure 7. Each dust mote can only have one message with the same name. If it receives a second one, the **`ReplacementOperation`** parameter is used to determine which one the mote will keep.

`OriginatorID:` The unique identifier code for the dust mote that was the source for the pheromone message.

`SenderID:` The ID of the dust mote that most recently relayed the message. If the message has only been relayed once (one hop), then this will be the same as the **`OriginatorID.`**

`Hops:` The number of communication hops from the source mote this message has traveled.

`Level:` The level of this message. Valid levels are positive integers.

`ReplacementOperation:` What to do if the receiver mote already has a pheromone of the same name. The choices are:

- **`KeepMaxLevel`** - Keep the pheromone with the highest level (Default)
- **`KeepMinLevel`** - Keep the pheromone with lowest level
- **`AddLevels`** - Add the two message's levels into one message
- **`KeepNewer`** - Keep the newer pheromone

DiffusionDecayRate: The value by which **Level** is decremented each time the message is relayed from one mote to another.

TemporalDecayRate: The amount by which **Level** is decremented during each decay cycle. Each mote periodically decays all the pheromones it is storing. This eliminates old state and allows the network to be robust to communication and topology changes.

Data: Extra Data. Just in case you actually want to say something to your neighbors.

4.1.2 Agent Message

Agent messages, or simply agents, are designed to transport data throughout the network. In the simulation, they are modeled as independent processes that are able to hop between neighboring motes at will. Agents are able to query the state of their current mote and its neighbors. This ability lets them navigate the network's pheromone landscape and head towards their goals.

4.1.2.1 Agent Message Data Fields:

Name: The name for this message. This is a text string, like the pheromone name. It is allowed for a mote to have multiple agents with the same name, because agents can be further discriminated by their ID.

SenderID: The unique identifier code for source dust mote.

AgentID: The unique identifier code for the agent. This is equal to the number of agents the source dust mote has sent. This number combined with the sender ID create a unique ID for every agent message.

Hops: The number of communication hops from the source this message has traveled.

Data: Data.

Error-prone communications affect how agents hop from mote to mote. To ensure that there is always only one copy of a particular agent, both the sender mote and the receiver mote would have to know that the communication took place correctly. The sender could then delete its agent and the receiver could execute its copy. Error-prone communications makes it impossible for both parties to know if a communication has been successful. [17] We are left with two alternatives:

- The sender transmits and then immediately deletes its copy. Drawback, agents could vanish. This is not as bad as it sounds, because that particular agent might be coming from a periodic source, or it might have been carrying redundant information.
- The sender transmits, but waits for one handshake from the receiver before it erases its copy. If it gets no handshake, it retransmits the agent. Drawback, the retransmission process could create multiple copies of the agent. Because the network topology is affected by communications errors, even a completely deterministic agent might not end up in the same mote that its duplicate traveled through. The solution is that agents, or the software that collects their data, will have to know what to do with multiple copies of the same agent.

4.2 Smart Dust Memory Requirements

The amount of information that needs to be stored in each mote is an important consideration. Careful selection of what to store must be made to ensure that the

algorithms will scale with increasing network size. In this work each mote needs to know:

- Their pheromones: These messages scale with algorithm complexity, not number of motes. For example, to perform position estimation (see section 4.6) with four BasisMotes, each mote will have to store four pheromone messages, regardless of the total number of motes there are in the network.
- Their agents: These programs reside in motes. In this work we assume that each mote can store an arbitrary number of agents. This is fine for simulation, but real systems with finite memory would need to be designed o limit the simultaneous number of agents per mote to a reasonable number. For example, the path projection algorithm in section 4.8 uses a constant number of agents. In contrast, the number of agents that are present in the relay network formation algorithm from section 4.5.1 depends on the total number of motes in the network. This problem is accentuated by the fact tat all these agents are trying to get to the same ChairMote.
- Their neighbors ID's: For certain algorithms, a program executing inside a dust mote must query all of that motes neighbors. That implies that each mote needs to have enough memory to at least remember its neighbors ID's. This does scale with network density, but not with network size. During the algorithm design phase, an optimal density can be decided upon and individual memory sized accordingly.
- More of their neighbor's state: This can increase the efficiency of calculations, but it would need to be synchronized somehow.

4.3 Pheromone Message Diffusion

Once a source mote initiates a pheromone message, it is relayed from mote to mote in a process we call message diffusion. Each mote periodically broadcasts all of its messages to its neighbors. Periodic retransmission helps to minimize the effects of random communications failures between motes.

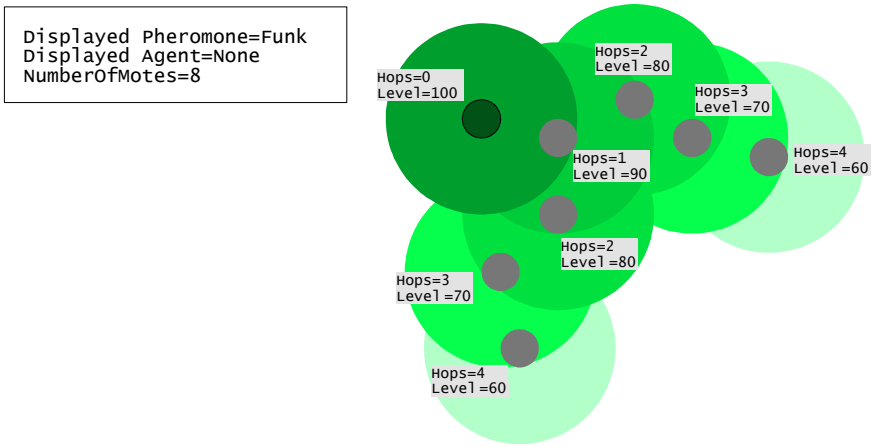


Figure 7: Pheromone message diffusion. The pheromone message named "Funk" starts at the dark green mote in the upper left with a `Level` of 100 and a `DiffusionDecayRate` of 10. Every mote periodically retransmits all the pheromones it is storing, which causes "Funk" to propagate from mote to mote. Every time the message is relayed, its level is decreased by 10. If a mote receives multiple copies of the same message, it keeps the one with the highest level. The net global result is a pheromone gradient with increasing levels directed towards the source mote. The numbers in the white boxes indicate how many hops each mote is from the message source and the corresponding level of "Funk".

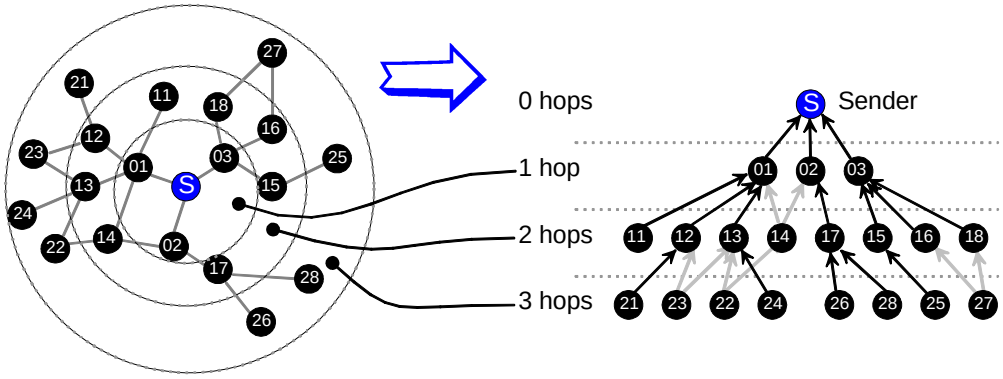


Figure 8: The pheromone diffusion algorithm generates a tree of motes if the `KeepMaxPheromoneLevel` replacement operation is used. The blue mote generates the source pheromone message with an initial level. It then transmits copies of this message to its neighbors with a reduced level. The motes within communications range (one hop) receive the transmission, reduce the level again, and then relay it to their neighbors. This receive-decrement-retransmit process is repeated by all the motes simultaneously and asynchronously. Since the `KeepMaxPheromoneLevel` operation discards all the messages except for the one with the highest level, this ensures each mote will have a route back to the sender along a path with minimal hop count. See text for details.

To prepare for transmission, each mote makes temporary copies of all its pheromone messages. The hop count of the temporary copies is incremented, and the level is decremented by the **`DiffusionDecayRate`**. Pheromones with a level that falls below zero are culled from this temporary list. This list is then sent to the communication system for transmission.

It can be seen that a pheromone with a **`DiffusionDecayRate`** of zero will not decay with each retransmission step. Instead, it will diffuse throughout the entire network in a process we will refer to as "flooding". If the message decays after a small number of hops, then it will affect only the motes in the vicinity of the source mote. This process is called "puddling".

When a set of pheromone messages is received from a neighboring mote, their names are compared with the list of pheromones that the mote already has. Duplicate messages are resolved by using the **`ReplacementOperation`** parameter from the new message. This parameter has the biggest impact on the resulting global distribution of pheromones throughout the network.

4.3.1 Graph Theory of Systems of Pheromone Messages

Assume a particular mote transmits a pheromone message to its neighbors with a replacement operation of **`KeepMaxLevel`** a nonzero **`DiffusionDecayRate`** and a zero **`TemporalDecayRate`**. We will refer to this mote as the source mote. As the message is relayed from mote to mote throughout the network, some dust motes will receive multiple copies from their neighbors. Since the motes will only keep the copy with the highest level, i.e. the message that had to make the fewest number of communications hops to reach it, each mote will keep message that came from a neighbor that is closer to the source mote. Figure 7 shows a dark green source mote in the upper left and a green pheromone message.

You can convert this diffusion into a tree by using the motes as vertices and the pheromone messages as edges. Each edge points from the mote that has the pheromone message towards the mote that sent it. This tree is not unique because of the situation where a mote receives more that one message with the same maximal level. Randomness in the timing of the asynchronous communications between motes

will influence which one of these edges (pheromone messages) will be kept. The graph shown in Figure 8 shows the edges that are deterministic in black and the edges that are probabilistic in gray. Even though each mote only keeps one pheromone message from the source, it still has the ability to query its neighbors about their pheromone messages. That allows the network to operate on the directed acyclic graph with all of the probabilistic edges. This is used in section 4.5 to pick the best mote from the set of neighbors to communicate with.

The important property of this tree is that every mote that is able to receive a relayed message has at least one neighbor that is one hop closer to the source. Because of the way the algorithm works, no mote can have a neighbor that is more that one hop closer. Essentially, this algorithm is a distributed implementation of a breadth first search [13].

4.3.2 Robustness of Systems of Pheromone Messages

Under the correct conditions, a network of dust motes communicating with pheromone messages can produce a very robust distribution of levels. More specifically, every system of network connectivity, source motes, and pheromone message type has a unique stable state. A network that is disturbed from this state will return to this stable configuration over time, as explained below.

We define the source dust mote as the one that initiates the pheromone message. This message must have a nonzero **`DiffusionDecayRate`** and use **`KeepMaxLevel`** as the **`ReplacementOperation`**. These conditions ensure that our diffusion algorithm will not produce any self-excitatory loops of pheromone levels. A self-excitatory loop occurs when a pheromone message can bounce from mote to mote and eventually return to the source mote with a level greater that or equal to the

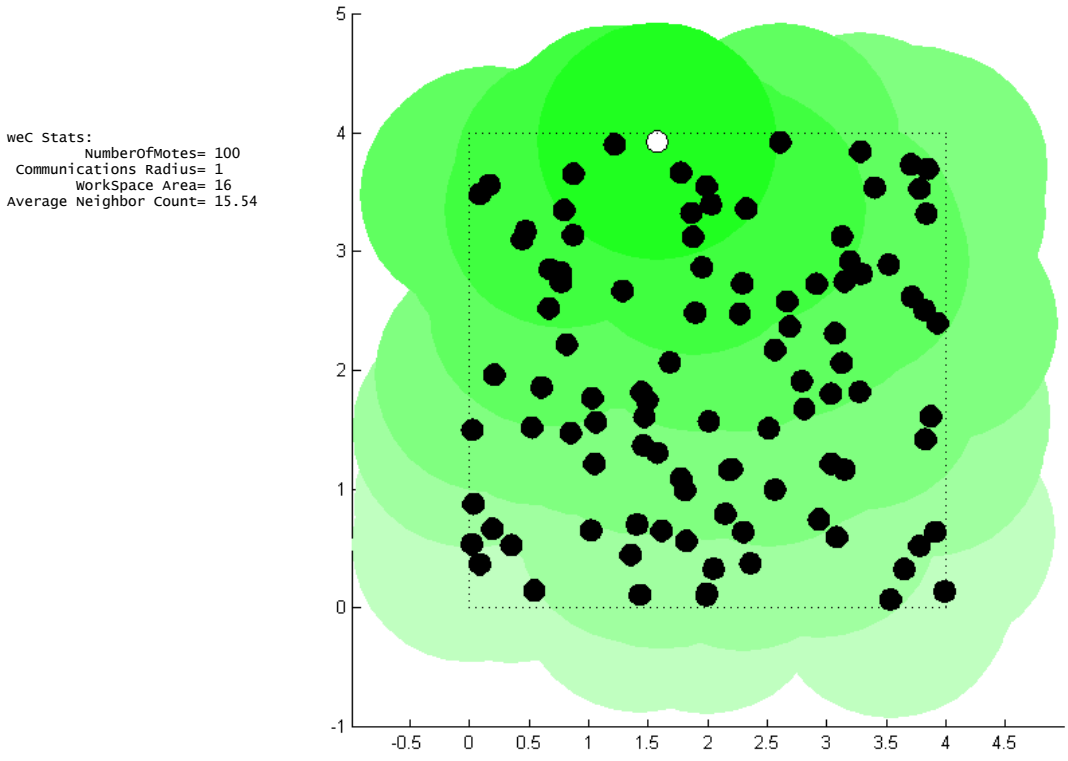


Figure 9: Pheromone message diffusion in action. The source mote is colored white. This message uses the `KeepMaxPheromoneLevel` replacement operation. The different values of pheromone level can be observed by the shades of green in the circles.

original level. A message such as this will never terminate and will bounce around the network forever.

These conditions will produce a unique distribution of pheromone message levels. The level of pheromone P in mote M will be given by:

$$\text{Level}_{PM} = \text{SourceLevel} - \text{DiffusionDecayRate}_P \cdot \text{Hops}_P \quad (1)$$

However, this distribution is not robust to pheromone level perturbations or network changes. For example, if the source mote stops transmitting, all the other motes will be left with a pheromone message forever. A solution is to add a nonzero value to the `TemporalDecayRate` field of each transmitted pheromone. This will ensure that the level of each message will eventually reach zero if the source stops transmitting. Therefore, an undisturbed Smart Dust network with no sources will eventually settle onto a stable, zero pheromone level state.

A combination of `DiffusionDecayRate` and `TemporalDecayRate` will produce a composite level distribution. The steady-state level of pheromone P in a given mote M (Level_{PMSS}) will be a function of:

SourceLevel = Initial level of pheromone P from the source Mote

Hops_M = The number of communications hops mote M is from the source mote

TimePerHop = Average amount of time it takes for one message to be transmitted and received. Communications errors and collisions should be included in the computation or measurement of this value. This makes it dependent on low-level communications system implementations, random errors, communication collisions, etc. The average number of collisions will be a strong function of the average neighbor count of the network.

$\text{DiffusionDecayRate}_P$, $\text{TemporalDecayRate}_P$ = Pheromone Parameters

$$\text{Level}_{PMSS} = \text{SourceLevel} - \left(\text{DiffuionDecayRate}_P + \text{TemporalDecayRate}_P \cdot \text{TimePerHop}\right) \cdot \text{Hops}_M \quad (2)$$

This steady-state value will only be approximate since there exists some randomness in the TimePerHop parameter.

Since the diffusion algorithm keeps the message with the highest level, a mote with an initial level lower that the stable value will immediately increase to the steady-state level defined above when it received the first message transmission. A mote that starts higher will lose pheromone level at the TemporalDecayRate until it gets to the steady-state level. At that point, the message it is receiving will have a higher level that the message it currently has, and will replace it.

Since the only topographical input to this equation is Hops_M, adding or removing motes will change the steady-state level of mote M only if their presence or absence affect mote M's hop count. For a sufficiently dense smart dust network the redundancy of network communications path should render individual hop counts insensitive to small disturbances in mote population. However, under severe network topology changes, the hop counts of some or most of the dust motes could be affected. The time an entire network takes to settle into the steady-state level distribution for a given topology can be determined by searching all the pheromone messages in all the motes to find the one with the maximum settling time:

$\text{Level}_{PMInitial}$ =The initial level of pheromone P in mote M

Level_{PMSS} =The steady-state level of pheromone P in mote M

$\text{TemporalDecayRate}_{PM}$ =The **`TemporalDecayRate`** of pheromone P in mote M

$$T_{SS} = \frac{Level_{PMInitial} - Level_{PMSS}}{TemporalDecayRate_{PM}} \quad (3)$$

This time quantity is only valid if the initial level of pheromone in this mote is greater that the steady-state level. If the initial level is lower, then that pheromone level is less than the steady state value and will be updated as the messages from the source mote re-flood the rearranged network. If all the T_{ss} values are less than zero, the then the maximum time for the network to converge would be the time for the pheromone message from the source mote to propagate through the entire network. This time is a characteristic of the network and will examined more carefully in the next section

4.3.3 Algorithm Execution Time

With the assumption that a communication between any two motes will, on average, take a certain (finite) amount of time, the total time for a n-hop puddle can be estimated with the equation:

$$Time_{puddle} = NumberOfHops \times TimePerHop \quad (4)$$

For an entire network of dust motes, the largest amount of time for a pheromone message to flood the entire network can be used as an upper bound for algorithm execution time. First, we need to know the maximum number of hops possible in any given network. The longest distance can be measured with a ruler and some knowledge of the obstacles and network boundaries. In most cases, an estimate of the diameter should suffice.

$$MaxNumberOfHops = \frac{LongestDistance}{CommunicationsRadius} \quad (5)$$

The equation for maximum flooding time is similar to the puddling equation:

$$Time_{Flood} = MaxNumberOfHops \times TimePerHop \quad (6)$$

The last step is to determine how many dependent floods your algorithm needs to terminate. A dependent flood is caused by a pheromone message that initiates a new pheromone flood when it is received by a dust mote.

$$\max(\text{Time}_{\text{Execution}}) = \text{NumberOfDependentFloods} \times \text{MaxNumberOfHops} \times \text{TimePerHop} \quad (7)$$

Knowing the time it takes for pheromone floods to propagate through our network allow us to define our "moving slowly" assumption from section 3.3. The motion of the motes only affects our software if it changes the network topology, so we can immediately rule out small spatial perturbations or anything else that would be equivalent to random communications errors. We can divide the remaining spatial disturbances into three classes, based on how they affect the network topology:

- The topology does not change for all time.
- The topology changes, but does so much slower than pheromone floods distribute throughout the network. Algorithms that depend on pheromone message floods will be unaffected, as each motes pheromone information will stay updated. There will be intermittent errors if the topology changes in the middle of a flood, but the next flood will correct them.
- The topology changes as fast or faster than your communications. If your network topology changes while pheromones are diffusing through the network, they will not be able to reach the correct steady-stade values determined by the topological positions.

Algorithms using agents are more difficult to characterize because an agent message can remain active in the network for an indeterminate amount of time. Some algorithms, like relay network formation from section 4.5 use messages that hop from one mote and then terminate at another following a minimal hop length path and

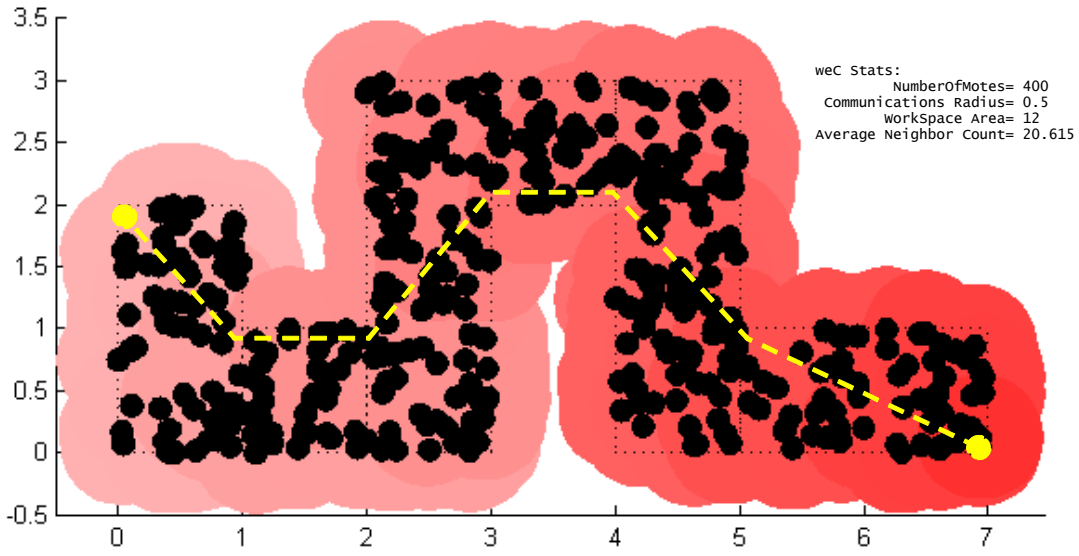


Figure 10: The upper bound on the time it takes for a distributed algorithm to execute can be estimated as follows: The yellow line in the above figure is the shortest network path between the two motes that are the farthest apart in physical space. This is the longest physical distance a pheromone message could travel, assuming it propagates through the network in a relatively straight line. The number of communication hops this actually takes is a function of distance and density (see section 4.6.1.2 for more details). Knowing the number of hops, the average time per hop, and the number of sequential global floods your algorithm requires enables you to estimate the algorithm execution time.

without stopping. These agents will take as long to propagate through the network as one flood, which let us use the above scenarios to analyze their behavior.

Knowing the physical environment's effects on the algorithms is important factor for the communications system design. For example, motes scattered on a field of daisies could use slower communications than motes spread on the leaves of a tree, and all could be downright lethargic compared to motes attached to working honey bees!

4.4 Directed Communication

Message diffusion makes it possible for any mote to talk to any other mote. Assuming the ID of the receiver mote is known, diffusion can be used to establish communications to that mote. The sender broadcasts a pheromone throughout the network that has the ID of the receiver mote stored in the data field. When the receiver mote receives the pheromone, it can then transmit an agent message that hops towards higher levels of pheromone, which will lead it back to the sender. Our diffusion algorithm ensures that every mote will have a pheromone message that came from a mote that is one hop closer to the sender. This allows our agent to propagate back to the sender along a path of minimal hop count.

If there is a need for continuous communication, the agent can build a routing table of the mote ID's it encounters as it heads back to the sender. Once the sender receives the reply, it stops transmitting the initial message. The next time the sender needs to talk to that particular receiver, it can send an agent message that follows this routing table back to the receiver. This eliminates the need for a second pheromone flood from the sender, but makes the communications link sensitive to network topology changes. If the initial pheromone broadcast has a non-zero **`TemporalDecayRate`** parameter, it will eventually decay, which conserves memory for the rest of the network.

For most of the algorithms in this work, there is no need for any two specific motes that are not neighbors to ever talk to each other. A more common use of directed communications would be to talk to a human user of the sensor network. The user would approach a convenient mote and query it for information from the rest of the network. This anointed mote, called the ChairMote, would then broadcast ChairMotePheromone throughout the network. This forms our tree from section 4.3.1 and gives all the other motes the ability to direct their information back to the ChairMote and subsequently out to the user. Section 4.5.1 presents a method to optimize the resulting routing network.

4.5 Heterogeneous Task Allocation

In the preceding chapters, all of the motes except for the one transmitting pheromone were doing the same "job", or running the same program. In some applications, it is useful to divide the sensor network into functional as well as spatial groups. The motivation for our example grouping is power savings and network efficiency, but this list is not exhaustive by any means. Each different application will have its own set of constraints to work with.

4.5.1 Relay Network Task Allocation

In section 4.3.1 we produced a tree by relaying pheromone messages throughout the network using pheromone diffusion. This enabled agents to direct themselves to the mote that was the source of the pheromone message, but required all if the motes to participate in the agent relaying process. We can improve the performance of this solution by dividing tasks among motes, making some motes exclusively responsible for relaying agent messages on their way to the source mote, and others responsible for receiving sensory data. In this example, we will imagine that the source mote, also called the ChairMote, was selected by the user to report information collected by the network.

This functional split is motivated by power constraints. Each mote has a finite amount of energy and time that it is able to use for its activities. If some motes focus their available resources for relaying agent messages to the ChairMote, that would allow nearby motes to maximize their expenditure on sensing. However, directing an arbitrary number of agent messages towards the ChairMote could overrun the communications abilities of that mote and its neighbors. This problm is compunded by the fact that the expect4d numebr of packets We will ignore this problem in the development of this algorithm, but a real system (or better simulation) would need to address this issue.

We will refer to the motes that relay messages as RelayMotes. We will define a mote to be "upstream" of another mote if it is fewer hops from the ChairMote. Motes that can communicate with an upstream RelayMote and thus spend all their time sensing are called normal motes. Figure 11 shows simulation output using the following color conventions:

- ChairMote -Dark Blue
- RelayMotes - Light Blue
- Normal Motes - Black
- Normal motes that cannot communicate to a RelayMote – Brown
- Motes that cannot communicate with the ChairMote (No pheromone=no

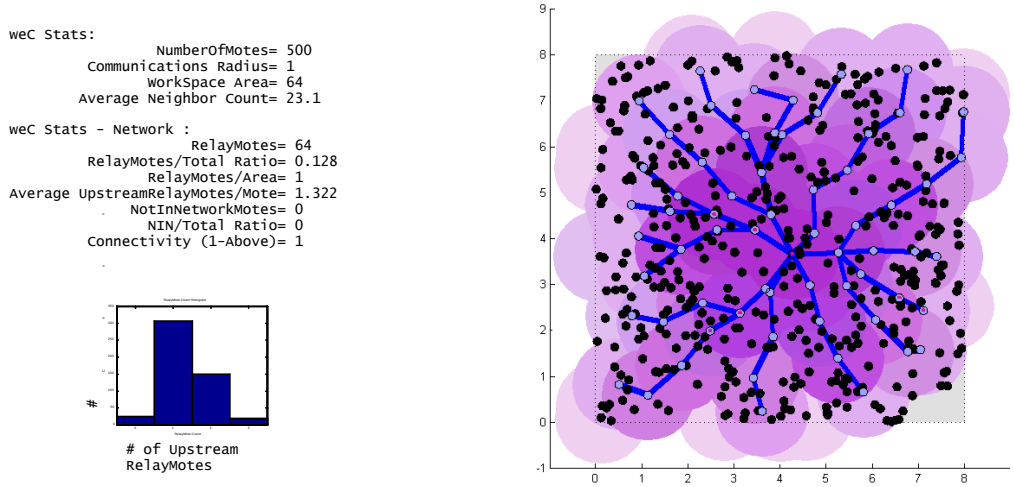


Figure 11: Relay network task allocation. The goal is to ensure that every mote is within communications range of a light blue RelayMote. The blue lines graphically depict the relay network formed from mote to mote. The dark blue mote in the center is the ChairMote. This is the mote that the user is interacting with. The RelayMotes are responsible for transferring information around the network. The black motes are normal motes, who devote all their time to sensing. The brown motes are also normal motes, but that cannot see an upstream relay mote and are not part of the relay network.

route back) – Gray

4.5.2 Finding an Optimal Distribution of RelayMotes

In order for any mote to send an agent message to the ChairMote, it first has to transmit this message to an upstream RelayMote, which then relays all of the agents it receives to another upstream RelayMote and so on. We will assume that each mote is able to relay a message to the ChairMote, i.e. we have a fully connected spatial distribution of motes.

In an optimal configuration, each RelayMote would be positioned to communicate exclusively with a maximal number of normal motes. This implies that the physical distance between any two neighboring RelayMotes, including neighbors upstream and downstream, should be as large as possible to maximize their net total area covered and minimize their communications range overlap. In other words, we are looking for the lowest density of RelayMotes that can cover the entire distribution. Since the total area to be covered is set by the network distribution, the lowest density will occur when we have the smallest number of RelayMotes. The number of RelayMotes does not always specify which ones need to be selected, i.e. the set of RelayMotes that provides total coverage with the smallest number of elements is not unique.

4.5.2.1 Finding an Optimal Distribution Using a Centralized Algorithm

The graph produced from this optimal arrangement is the tree shown in Figure 13, which is the tree from Figure 8 with the probabilistic edges converted into deterministic edges. The edges are picked to minimize the number of RelayMotes. Since this is a tree, removing any RelayMote, and its associated edges, would break the connectivity to the ChairMote through the RelayMotes until the network can re-configure itself to accommodate.

We start by letting the ChairMote pheromone message diffuse through the network. We can now group the motes based on how many hops they are from the ChairMote. We define **HopLevel** as the number of hops a mote is from the ChairMote. We then search all the combinations of RelayMotes at each **HopLevel** for the smallest number that allows the motes at **HopLevel**+1 to be connected.

```
for HopLevel=1 to MaximumNumberOfHopsInNetwork-1
    R=The Set of Motes at level HopLevel
    N=The set of Motes at level HopLevel+1
    for NumOfRelayMotes=1 to length(R)
        Is there a combination Q, of size NumOfRelayMotes, made from the
            elements of R that has a set of downstream neighbors equal to N?
        Yes:
            stop searching at this HopLevel
            make all the motes in Q RelayMotes
            break out of the for loop
        No:
            Increment NumOfRelayMotes and try again
    endfor
endfor
```

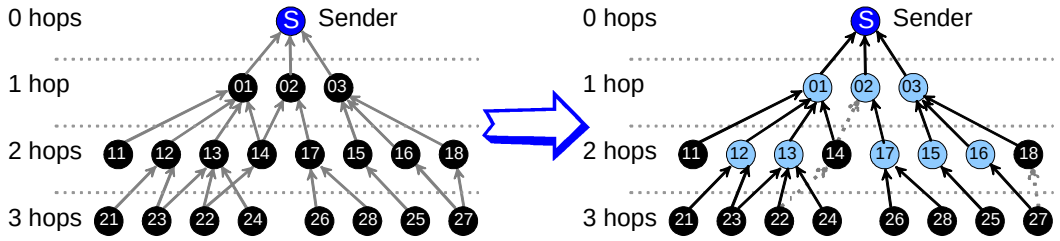


Figure 13: With careful selection of the edges in the diffusion graph from Figure 8, a network communications tree can be formed. Our diffusion results show that any mote can already communicate with the sender with the shortest possible number of hops. Now, we place the responsibility of relaying communications on only a few motes and leave the rest free for other tasks.

At each level of the tree, this algorithm finds the smallest set of RelayMotes to connect to the next level. The sum of the RelayMotes picked at each hop level is the global minimum number of RelayMotes. Figure 12 shows the output of the centralized algorithm and a comparison with the distributed algorithm presented next.

4.5.2.2 Distributed Relay Network Formation Algorithm

In order to achieve the same results with a distributed algorithm we will take advantage of the fact that an optimal solution has the lowest density of RelayMotes. We will attempt to achieve this by selecting RelayMotes that can communicate with large numbers of downstream motes.

The procedure begins with the ChairMote being selected by the user and flooding the entire network with ChairMotePheromone. Once each mote knows how many hops it is from the ChairMote, it can query its neighbors to produce a set of

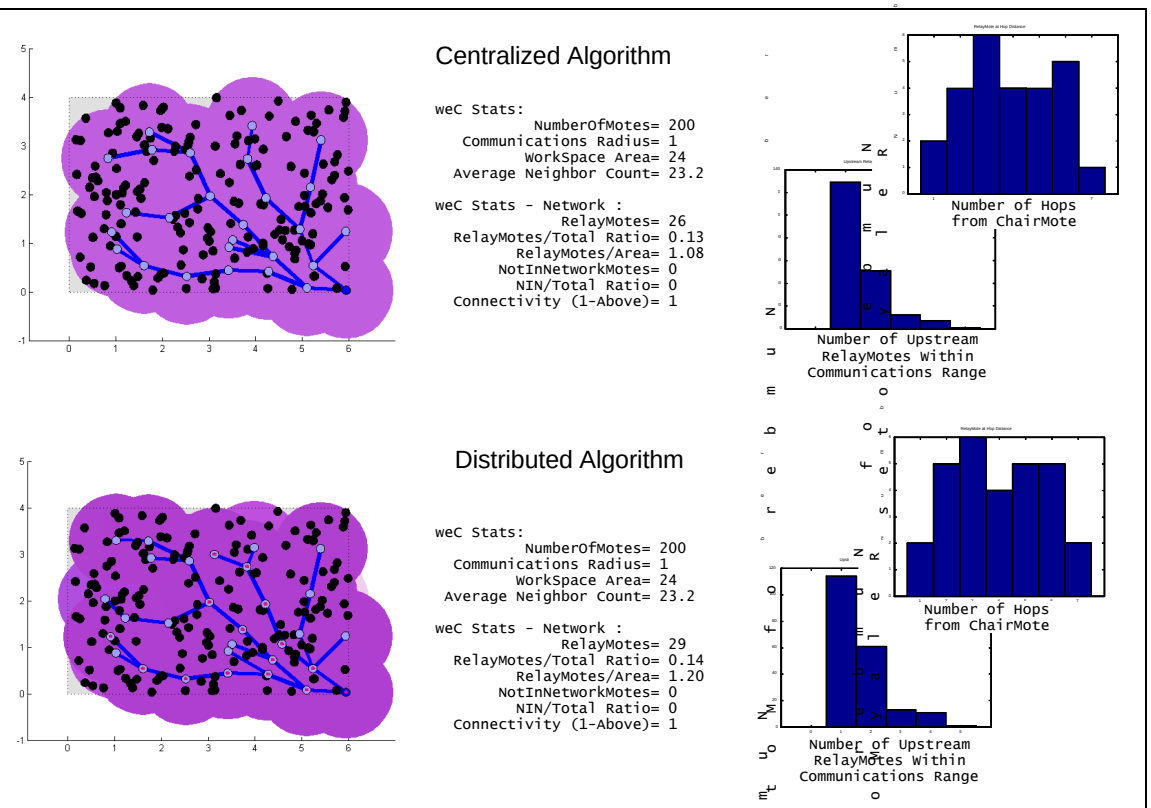


Figure 12: Centralized algorithm selection of relay motes compared to the output of the distributed algorithm.

upstream motes, thus converting our diffusion tree into a directed acyclic graph from which the down stream motes can choose their upstream RelayMote.

Our approach is to have each mote keep track of how many messages it has relayed recently. It stores this count in the level parameter of a pheromone message called RelayPheromone. This kind of differential trail reinforcement is very similar to the way ants reinforce scent rails to select the best routes to and from their food sources. The more individual workers use a particular scent trail, the stronger it gets. Essentially, the group as a whole makes a collective decision to use one particular trail, but the individual work ants are not aware of this level of reasoning, they are simply heading towards the biggest stink.

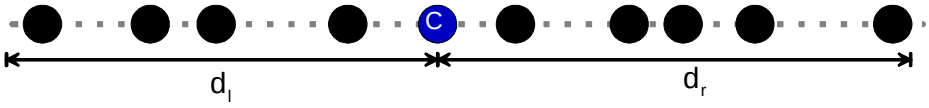
We use agent messages as our worker ants. They move from mote to mote using the following algorithm:

```
if UpstreamMotes==ChairMote
  RecipientMote=ChairMote
else
  MaxLevel=0
  RecipientMote=nil
  for CheckMote=1 length(UpstreamMotes)
    if the level of RelayPheromone in CheckMote > MaxLevel
        RecipientMote = CheckMote
    endif
  endfor
endif
HopToMote(RecipientMote)
AddPheromone(ReceipentMote,RelayPheromone)
```

In the above pseudocode, **UpstreamMotes** is the set of motes upstream of the current mote, **RelayMotes** is the set of all RelayMotes, and **ChairMote** is the set of ChairMotes (there is usually only one). HopToMote is a function that transfers the agent from one mote to another. AddPheromone sends a RelayPheromone message to the current mote. This message has a **ReplacementOperation** of **AddLevels**, so it increases the level of the current RelayPheromone message, or creates one if absent. When the level of this pheromone gets above a constant, then that mote is selected as a RelayMote. The level of this pheromone in each mote decays over time, which provides the same kind of robustness afforded to the diffusion network from section 4.3.2. In particular, if there are changes, no matter how drastic, to the network topology, the system will eventually reorganize to accommodate the new network.

In the simulation, agent messages are produced at random intervals by any mote that is not in communication with a RelayMote. In an actual application, messages would only be produced when a mote or group of motes has pertinent information for the ChairMote, but for the sake of temporally efficient simulation runs, we will assume that everyone has something important to say. Output from simulation runs using this algorithm are shown in Figure 11, Figure 15, and Figure 16. The dark blue mote is the ChairMote, the light blue motes are RelayMotes, the black motes are sensor motes, and the brown motes are sensor motes that cannot see a RelayMote. The level of RelayMotePheromone for each mote is indicated by the purple circles, with a higher level corresponding to a purpler circle. The little red dots in some of the motes are agent messages moving towards the ChairMote.

One-dimensional arrangements of motes always converge to the global minimal number of RelayMotes as shown in Figure 15. The number of RelayMotes required is given by:



$$n_r = \text{ceil}\left(\frac{d_r}{r_{Comm}}\right) - 1 \quad (8)$$

$$n_l = \text{ceil}\left(\frac{d_l}{r_{Comm}}\right) - 1 \quad (9)$$

$$n_{total} = n_r + n_l \quad (10)$$

The function ceil returns its argument rounded towards infinity.

A carefully constructed two-dimensional arrangement will also converge to the optimum, but this is not true in general, as shown in Figure 14. In this situation, the algorithm still performs well, as shown by the histograms in Figure 16.

It is interesting to watch the network of motes evolve over time. As stated previously, the lowest number of RelayMotes occurs when they are at the maximum extent of each other's range. However, as relay motes are selected, the most optimal selections are not always made right away. Eventually, the RelayMotes spread out to the optimal positions. A good way to think about this is to imagine what happens close to the ChairMote. Assume that the first RelayMote selected is located a distance of ½ the communications range from the ChairMote, but there is another mote that is at the maximum extent of the ChairMote's range. On average, this second mote has a higher probability of receiving a transmission from a mote with a higher hop count, because of the fact that more of its communications range overlaps those of motes with higher hop counts. As a result, it will conduct more network traffic than the mote that was picked first, and it will eventually be selected to be a relay mote too. Eventually, the level of RelayMotePheromone in the first mote will decay below the threshold, and it will turn back into a normal mote. This process can be likened to "smoothing out the wrinkles" in our network, because it happens in much the same way as you would smooth out the puckers in a bed sheet while making a bed.[i]

In our assumptions section, we gave ourselves globally unique IDs and a uniform distribution of motes. Both of these constraints can be relaxed for this algorithm. Each mote only needs to know how to target its neighbors individually, one of which will be its upstream relay mote. Therefore we could use a system of locally unique IDs [14] instead of our global IDs. In addition, our distribution does not need to be uniformly random as long as every mote is connected in some way to the ChairMote.

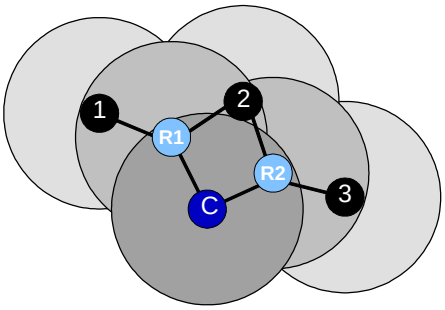


Figure 14: In a two-dimensional network it is not always possible to find a solution where there is only one upstream RelayMote for each mote. In this example, mote #1 needs RelayMote #R1 and mote #3 needs RelayMote #R2. Mote #2 will always be able to see two upstream RelayMotes.

[i] Contrary to popular belief, I DO know how to make a bed!

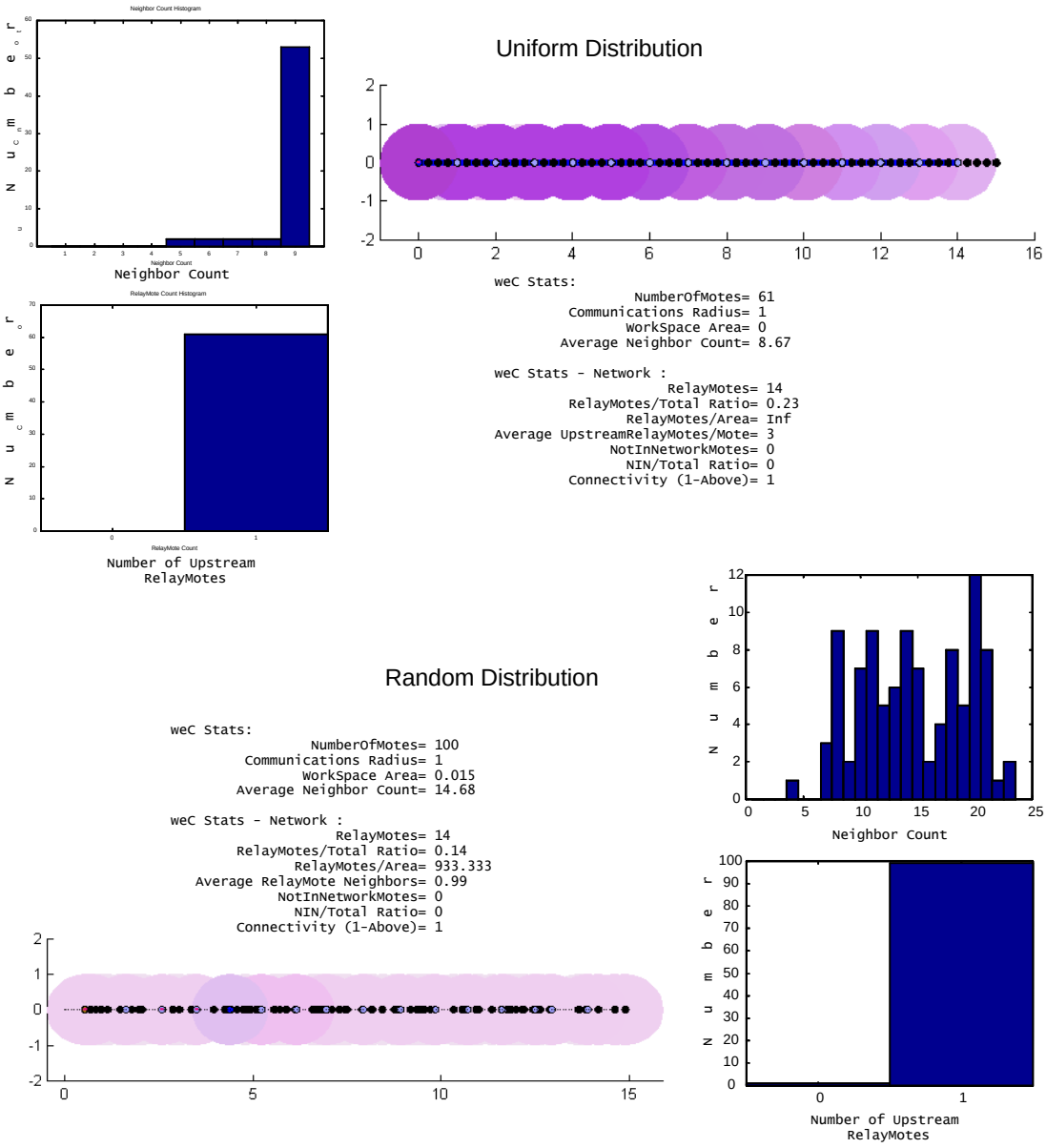


Figure 15: One-dimensional relay network task allocation. The top figure displays motes that are uniformly positioned, while the bottom figure shows motes that are randomly positioned. When the motes are in a one-dimensional network, the algorithm in section 4.5.2.2 always produces the minimal number of RelayMotes.

4.5.2.3 Failure Modes

As agents move through the network, they reinforce whichever path they travel. Since the optimal choices of relay motes are not made initially, suboptimal selections can be initially reinforced and become permanent, forcing the global solution to become stuck in a local minima. This problem can be rectified some what by having the agents make their movement decisions at random until they move to a mote that can communicate with an upstream RelayMote. This allows the network to collect more information about which motes are the best RelayMotes before making a decision. The price to pay for this extra information is longer convergence time.

4.5.2.4 Robustness of Relay Networks

The structure of the network is based on the underlying diffusion of ChairMote Pheromone. If there are topology changes, this underlying structure will change as well. Since agent motes only look upstream for RelayMotes, each agent will still move towards the ChairMote in the shortest number of hops. However, as in the section above, RelayMotes that have been reinforced might still remain active, forcing the global solution into a global minima.

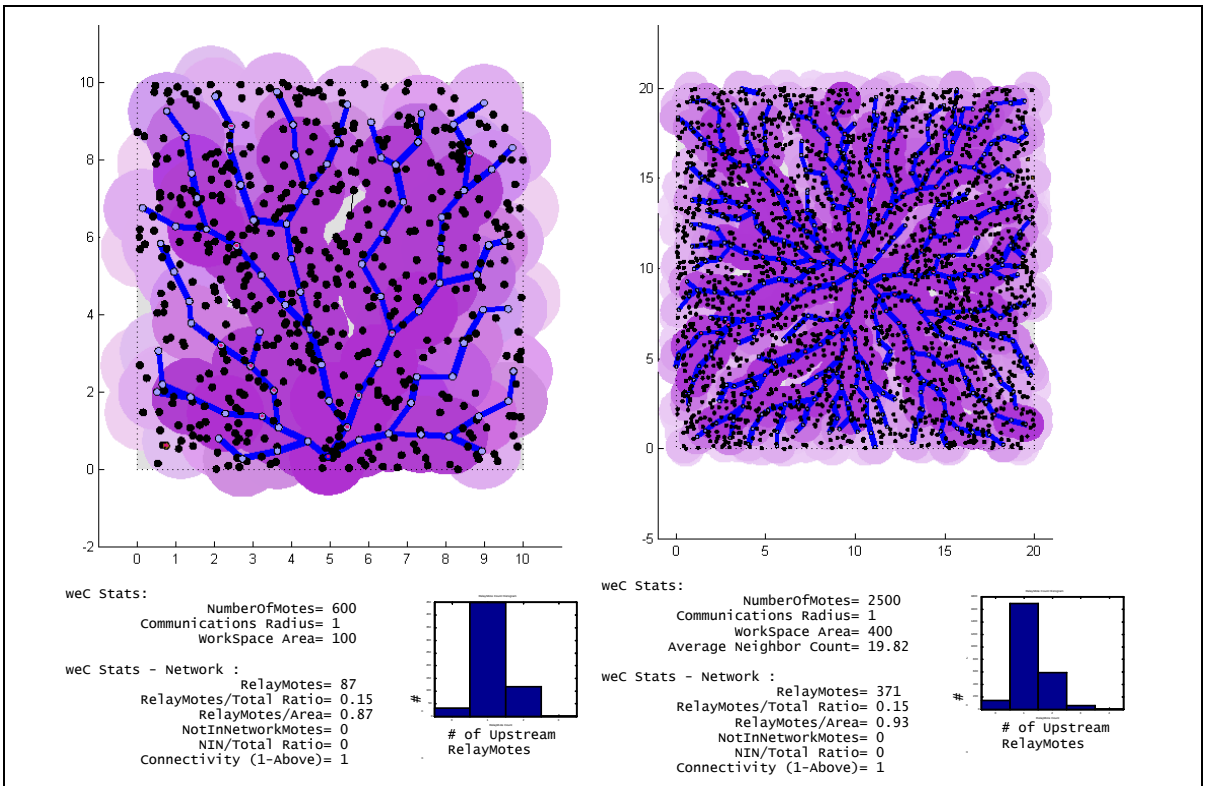


Figure 16: Two-dimensional relay network formation with large groups of dust motes. Our trail reinforcement algorithm cannot always converge to the optimal solution, but it still performs well, with most motes only having one upstream RelayMote. (See the bottom histograms for each data set) Sometimes the algorithm fails by getting stuck in local minima and by physical constraints as shown in Figure 14.

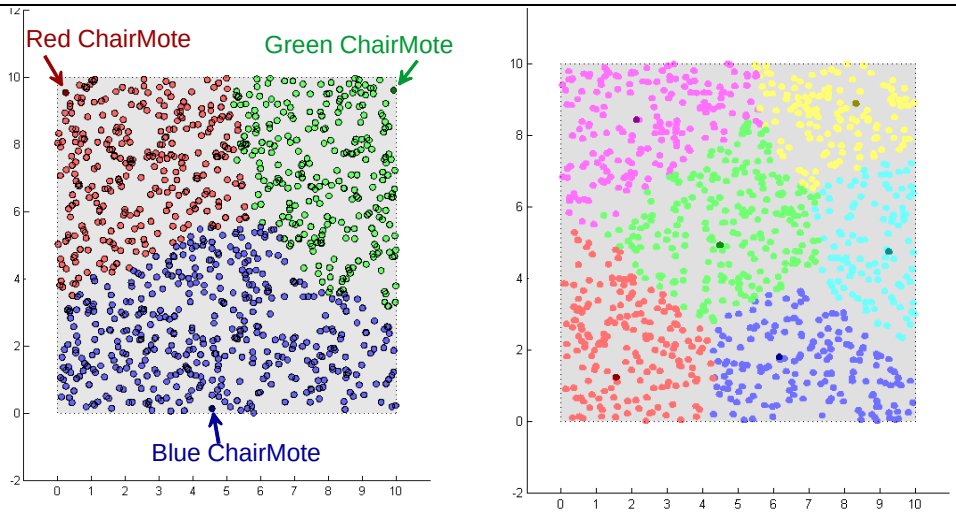


Figure 17: Basins of attraction for multiple ChairMotes. The different colors show to which ChairMote a agent will head based on its starting location. Since pheromone diffusion always finds the shortest path back to the source, the entire region is divided into Voronoi polygons, one for each ChairMote. In these simulations, we have given each ChairMote a different pheromones for illustrative purposes only. The basins will still form if the same pheromone is used for all the ChairMotes.

4.5.2.5 Multiple ChairMotes

In the preceding section, we have assumed that there was only one ChairMote in the network. However, since pheromones that are superimposed keep the maximum value, we could have several ChairMotes spread throughout the network. The diffusion algorithm will still guarantee that the agent messages will head upstream. Additionally, they will automatically head towards the closest ChairMote. The basins of attraction between multiple ChairMotes approximate the Voronoi diagram created from their positions.

4.6 Determining Physical Position from Network Topology

The dust motes themselves could care less where they are physically, but the humans using the sensor network might want to know where all the interesting things are happening. To ease human-mote interactions, it would be nice if an arbitrary individual mote could determine where it is. To facilitate this, a few of the motes could be told exactly where they were. In an outdoor environment, these motes would have GPS receivers. In an indoor situation, they would be given their coordinates on a floor plan of the building. We will call these special motes BasisMotes, borrowing a bit of linear algebra nomenclature. In the diagrams these motes and their pheromones will be colored gray. The mote whose position we are trying to estimate will be referred to as the "MeasureMote" and will be colored white in the pictures.

Each of these BasisMotes would then diffuse their position information throughout the network with their own individual pheromone message, i.e. "BasisMote1", "BasisMote2", etc. Since multiple pheromones can flow through the network at once, this information will propagate through the network in the time it takes for one communications flood.

The time required for multiple pheromone transmission does not scale with network size. It is a function of neighbor count and the number of different pheromone messages that need to be propagated through the network. The number of pheromone messages that each mote needs to retransmit during each periodic transmission cycle is equal to the number of BasisMotes. With a simple time division multiplexing scheme, the pheromone message

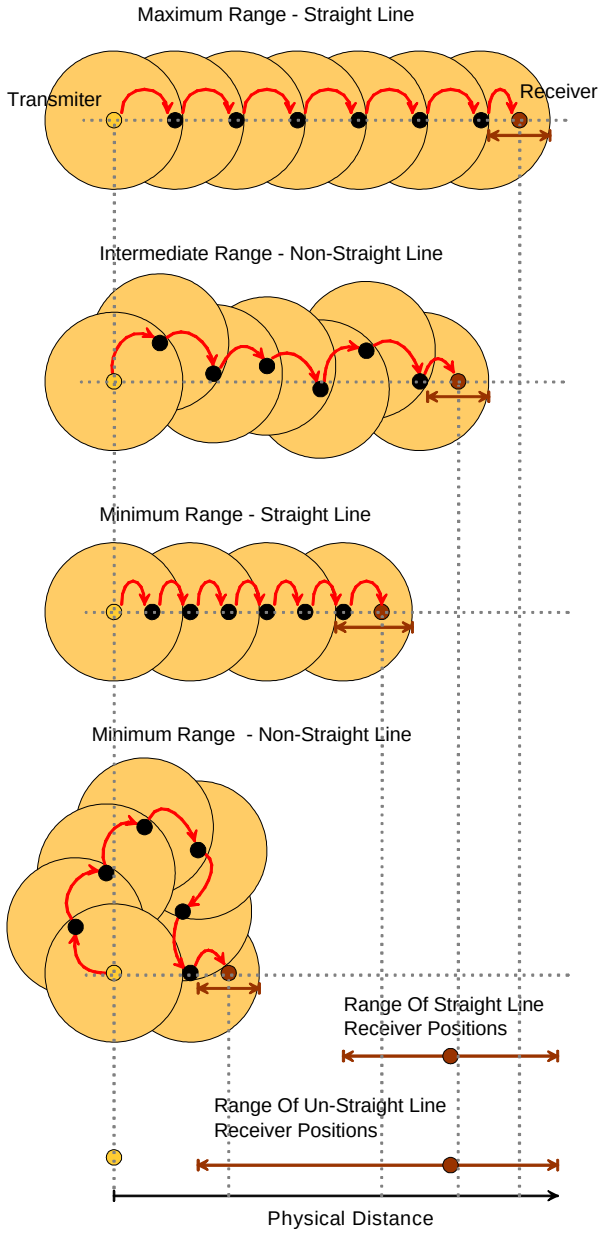


Figure 18: Even with a constant number of communications hops between any two motes, their relative physical positions can vary greatly. These four pictures depict the upper and lower bounds of the distance between two motes that lie in a straight line and are seven hops apart. The top figure shows the maximum possible seperation. The second figure shows a more normal path through a fairly dense network. The third picture shows the minimum distance possible if all the intermediate motes are constrained to lie in a straight line. The bottom picture shows the minimum possible distance, which is always equal to one communications range, regardless of the number of hops..

periodic transmission cycle would need to be larger that the time it takes to transmit each set of pheromone messages times the maximum number of neighbors that any mote is expected to have.

Once all the pheromones have settled, each mote will know how many hops it is away from each of the BasisMotes. The next step is to convert these lists of hops, or "Mote Coordinates", into something more useful, like Cartesian coordinates.

4.6.1.1 Maximum Distance Position Estimation

Given the number of communications hops and the average communications radius, an upper bound on the physical distance between motes can be determined by counting the number of hops a pheromone message takes to get from one to the other. The set of diagrams in Figure 18 shows the maximum and minimum physical distance between two motes that are seven hops from each other. These maximum distance bounds define circles of acceptable positions around each BasisMote, as shown in Figure 19. The intersection of these circles defines a region where the MeasureMote must lie. More formally:

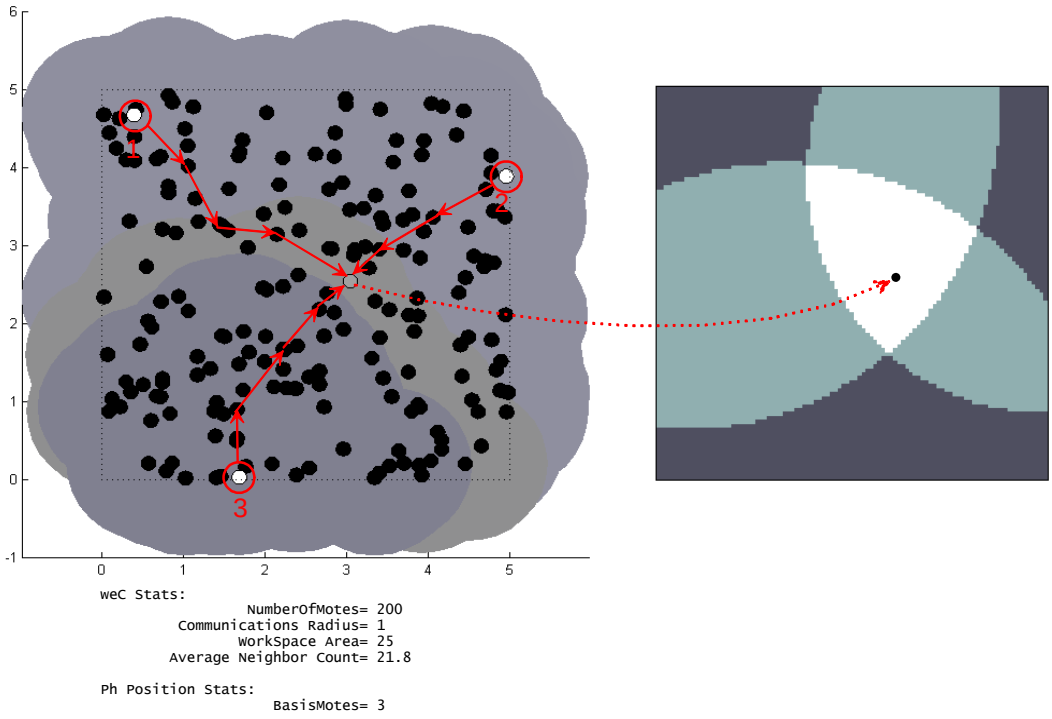


Figure 19: Maximum distance position estimation. The circled motes in the left hand picture are called BasisMotes because they know their physical position. The white mote in the middle is the MeasureMote and would like to determine where it is. It can measure the number of hops between it and the BasisMotes. With the number of hops and the communications range, it can compute the maximum possible distance from each BasisMote. The overlaping of these maximum positions produces the white area in the right hand picture. The MeasureMote can conclude that it it somewhere within this area. For comparison, the black dot in the right hand figure is its actual position.

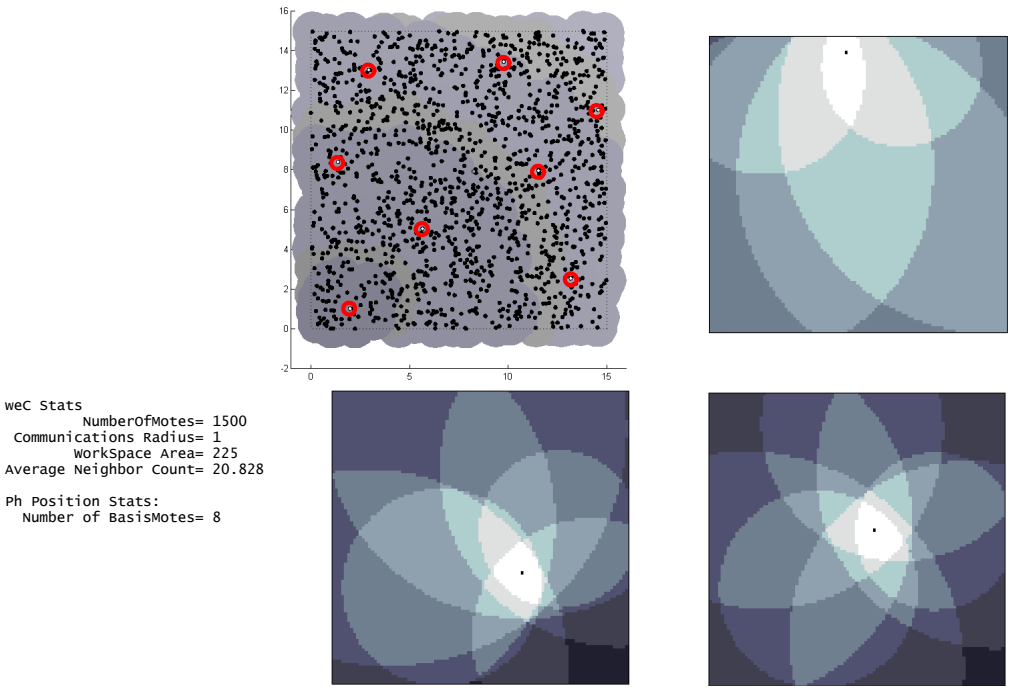


Figure 20: More maximum distance position estimation plots. The eight BasisMotes are circled in the upper left hand picture. The actual and estimated positon of the MeasureMote are shown by the white region and black dot in the other three diagrams.

Let R_{comm} be the average communication radius

Let $Hops_i$ be the measured number of hops from the MeasureMote to the i^{th} BasisMote

The maximum distance from the MeasureMote to the i^{th} BasisMote is given by:

$$d\max_i = R_{comm} \times Hops_i \qquad (11)$$

4.6.1.2 Average Distance Position Estimation

While the above technique is simple, the resulting region can be very large and difficult to define. A stronger relation between the number of hops and physical distance would give us a more precise estimate of position. However, this distance will be a function of the number of hops and the density of dust motes in the surrounding area. A greater density of motes will lead to a higher probability that the motes that relay the message will be arranged in a straight line. The two non-straight paths in Figure 18 demonstrate how this "line straightness" parameter can have a profound effect on distance measurements. We will define a "line straightness" constant, C, as the percentage of the maximum distance a communication with a given number of hops will travel.[15] In other words:

$$\text{ActualDistance} = C \cdot \text{MaxDistance}(\text{hops}) \qquad (12)$$

One approach would be to measure C for many different mote densities. A slightly more elegant solution (involving, of course, more assumptions) is to assume that C is a global constant for a given network of motes. This assumption allows us to make a more formal development of position estimation.

Let n=the number of BasisMotes. Assume $n \geq 3$.

Let B_i be the two dimensional position vector of the i^{th} BasisMote. Assume we can measure position to infinite precision. (we have very good GPS receivers)
Let $Hops_i$ be the measured number of hops from the MeasureMote to the i^{th} BasisMote
Let X be the two dimensional position vector of the MeasureMote.
Let C be the straightness constant as defined above
Let R_{comm} be the average communication radius

The actual distance from the MeasureMote to the i^{th} BasisMote is given by:

$$d_i = |X - B_i| \tag{13}$$

As before, the maximum distance from the MeasureMote to the i^{th} BasisMote is given by:

$$dmax_i = R_{comm} \cdot Hops_i \tag{14}$$

We can define an error between this distance and our measured distance, shortened by our line straightness constant:

$$e_i = d_i - C \cdot dmax_i \tag{15}$$

We would like to minimize this error for all n BasisMotes. This total error is given by:

$$TotalError = \sum_{i=1}^{n} (e_i)^2 = \sum_{i=1}^{n} \left(|X - B_i| - C \cdot dmax_i \right)^2 \tag{16}$$

Our problem has now been reduced to minimizing a function of two variables: X, the vector (x,y) position of the MeasureMote, and C, our global line straightness constant. There are many ways to do this, but our desire to optimize algorithm development efficiency coupled with an abundance of computational resources encouraged us to implement a brute-force search of this entire three-dimensional space. The results were then plotted using MATLAB's exhilarating "hot" color scheme to produce the pictures shown in Figure 21. The estimate is often quite good. The errors that occur might be due to the fact that C is not globally constant. Most likely this "constant" varies with local density and the number of hops the MeasureMote is from each BasisMote.

4.6.1.3 Position Estimation Errors

There are a multitude of ways both of these algorithms can fail. Any obstruction in between a BasisMote and the MeasureMote will cause an increase in hop count because the pheromones will have to diffuse around the obstacle instead of in a straight line. This will make that BasisMote seem further away than it actually is. In addition, the positions of the BasisMotes relative to the MeasureMote will affect the accuracy of the output.

In the average positioning algorithm, the line straightness constant C might not be the same for all BasisMotes. Indeed, it might not even be constant over the path between the MeasureMote and one BasisMote. If the errors are random, they would cancel out. Systematic errors, like a network where the right hand side is denser than the left hand side, would cause errors in the reported position. One solution for future work would be to measure C directly by computing the distances and hops between BasisMotes. Each BasisMote pair could then broadcast its value for C. An individual mote could take the average of these values, of use the value from the pair that it is the fewest hops away.

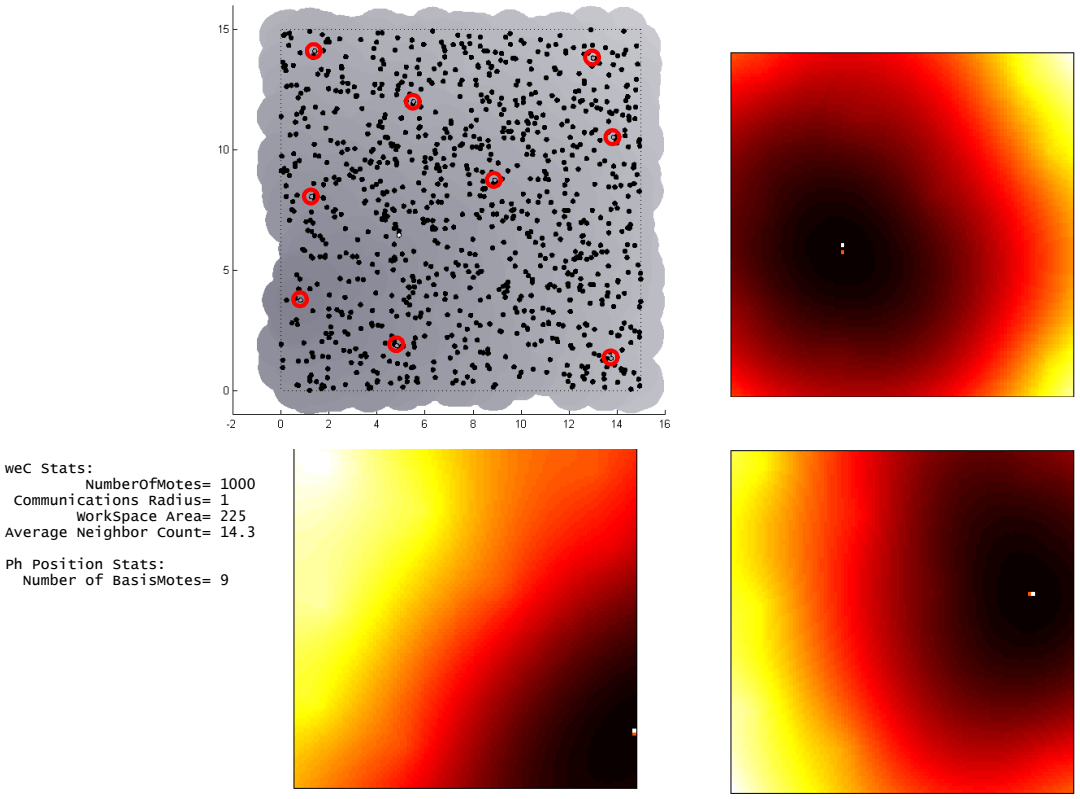


Figure 21: Average distance position estimation. As before, the MeasureMote counts the number of hops it is from each BasisMote, which gives it the maximum spacing between the two. Then it computes an error term given by:

$$\text{error} = \text{ActualDistanceToBasisMote} - C \cdot \text{MaximumDistance}$$

from each BasisMote. The ActualDistanceToBasisMote term and C are unknowns. The last step is to find the best combination of X-Y position and C that produce the smallest total error between the MeasureMote and all the BasisMotes. The results are shown above. The eight BasisMotes are circled in the upper left hand picture. The actual and estimated position of the MeasureMote are shown as white and orange dots in the other three pictures. If you cannot see two dots, thst's because they are very close together!

4.7 Physical Distributions and Network Topology

4.7.1 Neighbor Counts

Our algorithms depend on a network formed by local communications between motes. The topology of resulting network is affected by many factors, including the workspace area, the communications range, the total number of dust motes in this area, and how big they are. An individual dust mote cannot measure any of these quantities independently. The only information that can be sensed about the state of the network is the number of neighbors within communications range. We will refer to this parameter as the *neighbor count* and use it help parameterize the network.

$$\text{NeighborCount} = \pi \cdot \text{CommunicationRadius}^2 \cdot \text{GlobalDensity} \qquad (17)$$

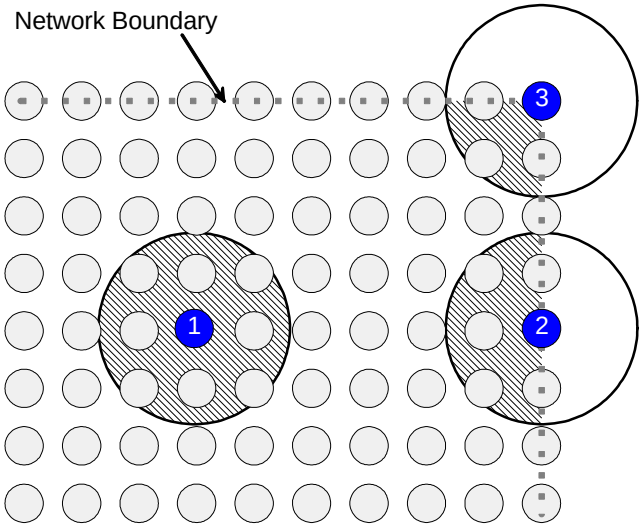


Figure 22: Edges affect the local neighbor count measured by individual motes. Although this is a uniform distribution, because of the network boundaries Mote #2 can communicate with only about 1/2 the number of neighbors as Mote #1. Mote #3 in the corner has only 1/4 of the neighbors as the interior mote. This effect ruins the uniformity of our neighbor count measurements, but can be used to detect motes on the edge of a network. See section 0 for this application.

Since GlobalDensity cannot be measured directly, a more pragmatic approach is to simply count your neighbors.

4.7.2 Edge effects

Edge effects occur at the boundaries of the dust mote network. Figure 22 shows a uniform distribution of dust motes spread out over an area that is much larger than the communications range. On average, the motes in the middle will be within communications range with more motes than the motes on the edges, because of the fact that a mote on the edge has no neighbors in a particular direction. You would expect for motes on a flat edge to see half of the average neighbor count and motes on a 90 degree corner to only see one quarter of the global average, assuming the global average is large enough to allow an individual mote to discern fractions of it reliably.

The perimeter/area ratio of the distribution determines how "edgy" a network is. The smallest possible value for edginess occurs in a uniform circular distribution and is given by:

$$\text{Edgyness} = \frac{2 \cdot \pi \cdot r}{\pi \cdot r^2} = \frac{2}{r} \tag{18}$$

Where r is the radius of the distribution.

4.7.3 Edge Detection

While edge effects ruin the uniformity of local neighbor counts, they can be used to allow individual motes to determine whether they are on the boundary of a distribution. Knowing that you are on the perimeter can be useful. For example, if a mote on the edge, called an EdgeMote, detects a new sensory stimulus that it has not been informed about through pheromone messages, then this stimulus must be a new target. This mote could alert nearby motes to the presence of a new target and update the "distributed database" of targets in the network. The opposite sequence of events would occur if the EdgeMote loses track of a target that its neighbors can also no longer detect. This can let the network keep track of when targets enter and leave.

Because of concavities on a real distribution, the edge of the network cannot be defined using a convex hull algorithm. [16] For a uniform distribution with a sufficiently high global average neighbor count, having a low local neighbor count is correlated to being on the boundary of a distribution. We developed two algorithms for finding EdgeMotes in a distributed fashion.

4.7.3.1 "Charge" Conservation Algorithm

In order to use the edge effect to deduce your position in the network, you have to know what the average neighbor count is for the rest of the group. One way to do this is to set a variable to be initially equal to your own neighbor count, then average this quantity among your neighbors. An analogy to a physical system would be to charge up a capacitor to a level equivalent to that mote's neighbor count. One terminal of each capacitor is then tied to ground. Next, connect all the positive terminals of the capacitors together. After the charge redistributes, all the caps will have the same amount. This value will be the global average amount of charge. In our analogy, charge was initially equivalent to neighbor count, so this quantity will be equal to the global average of neighbor count. An algorithm that does this in a distributed fashion follows:

```
if t=0
  Charge=NeighborCount
else
  DidSomething=1
  While DidSomething==1
    DidSomething=0
    for count=1 to NeighborCount
        MyNewCharge = (MyCharge + MyNeighbor'sCharge(count)) / 2
        MyNeighbor'sNewCharge(count) = MyNewCharge
        if MyNewCharge ≠ MyCharge
            MyCharge=MyNewCharge
            DidSomething=1
        end
    end
  end
end
```

Each mote queries each of its neighbors in succession and equalizes both of their values of charge. The algorithm terminates when each mote has the same amount of charge as each of its neighbors. The net result is that eventually all the motes have the same value in their charge register. This value is the average charge of the entire network. Since the initial charge was set to local neighbor count, the average charge is the global average of neighbor count. To assure this we must impose the additional constraint that we only average charge with one neighbor at a time.

$$\sum_{M=1}^{NumOfMotes} Charge_M = TotalCharge \tag{19}$$

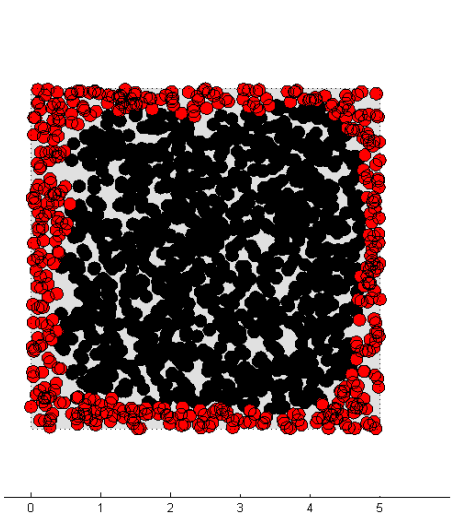
$$GlobalAverage = \frac{TotalCharge}{NumOfMotes} \tag{20}$$

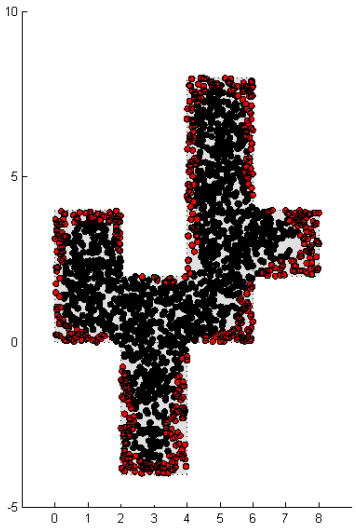
If each inter-neighbor averaging step uses the function:

$$Charge_{Mote1} = Charge_{Mote2} = \frac{Charge_{Mote1} + Charge_{Mote2}}{2} \tag{21}$$

Then total charge is conserved locally. Since each mote can only share charge with one other mote at a time, global charge must also be conserved. The algorithm terminates when all the motes have the same charge. This gives us:

$$TotalCharge = NumberOfMotes \cdot Charge_{MteN} \tag{22}$$






```
:C Stats:
                NumberOfMotes= 1600
        Communications Radius= 1
                WorkSpace Area= 25
       Average Neighbor Count= 172.1

:C Stats - Edge:
 CEdge (Neighbor Count Ratio)= 0.5
                    EdgeMotes= 383
         EdgeMotes/Total Ratio= 0.28
  Average EdgeMote Neighbor Count= 113.9
\verage NormalMote Neighbor Count= 191.7
```

```
weC Stats:
                NumberOfMotes= 2000
        Communications Radius= 1
                WorkSpace Area= 44
       Average Neighbor Count= 123.077

weC Stats - Edge:
        CEdge (Neighbor Count Ratio)   = 0.5
                    EdgeMotes= 607
         EdgeMotes/Total Ratio= 0.30
  Average EdgeMote Neighbor Count= 87.0972
Average NormalMote Neighbor Count= 140.191
```

Figure 23: A example of edge detection. First, the global average or the global maximum and minimum values of neighbor count are computed in a distributed fashion. Then the motes with a local neighbor count that is less than a constant times this average turn red. These motes are called EdgeMotes. The success of this algorithm depends heavily on the average neighbor count of the network. Networks with low neighbor counts do not perform well, as shown in Figure 24.

$$GobalAverage = \frac{TotalCharge}{NumOfMotes} = \frac{NumOfMotes \cdot Charge_{MoteN}}{NumOfMotes} = Charge_{MoteN} \quad (23)$$

There are three drawbacks to this approach. The first is the need for some type of synchronization. The motes need to know when it is time to count their neighbors and when it is time to share charge. Knowing when to stop sharing is comparatively easy – you look at how much your charge changes from step to step and stop when this change is small.

The second problem is more difficult to fix. In order for the algorithm to work, the total amount of charge in the entire network needs to be conserved. This means that for each communication from mote to mote, neither of them should modify their charge until they are sure that the other has received the message and will modify their charge. This is the classic Byzantine consensus problem and is provably intractable. [17] The best we can do is handshake back and forth until the probability of multiple message errors are smaller than we are concerned about.

The final concern is robustness to network topology changes. If the network is drastically altered, new edges could be created and global averages could change. The motes would need to be able to detect this and then reset their charge and start sharing again. It would be difficult for an individual mote to be able to sense when such a drastic alteration has occurred.

4.7.3.2 Min/Max Pheromone Messages Algorithm

A second approach to edge detection involves using pheromone messages to propagate maximum and minimum neighbor counts throughout the network. Once each mote knows the global maximum and minimum values of neighbor count it can then compare its own neighbor count to these values.

Section 4.1.1 described the effects of different values of the **`ReplacementOperation`** field in the pheromone message. These different operations can be used to propagate minimum and maximum values throughout the network. A **MaxNeighborPheromone** message would use the **`KeepMaxLevel`** replacement operation, while a **MinMeighborPheromone** would use the **`KeepMinLevel`** operation. Both messages would need a **DiffusionDecayRate** equal to zero to ensure that they spread throughout the entire network. A non-zero **TemporalDecayRate** value would ensure that old messages eventually fade to allow for network changes

In contrast to the charge conservation algorithm, there is no need for any synchronization or communications handshaking. Each mote becomes a source for the two pheromone messages described above, **MaxNeighborPheromone** and **MinNeighborPheromone**. The levels of both of these messages are set to that mote's own neighbor count. As messages arrive from neighboring motes, the respective **ReplacementOperations** of the different messages will replace the source mote's own message. For example, if our mote has a neighbor count of 10 and receives a **MinNeighborPheromone** message from its neighbor who has a neighbor count of 5, it would keep its neighbor's message and discard its own. When it transmits its pheromone messages, it would send a **MinNeighborPheromone** message with a level of 5.

Since this mote is a source for a **MinNeighborPheromone** message with a level of 10, it will transmit this message whenever it is not receiving a message with the same name and lower value. Essentially, whichever mote has the lowest value of neighbor count acts like a global source for the **MinNeighborPheromone**. The same situation happens in reverse for the highest value if neighbor count and

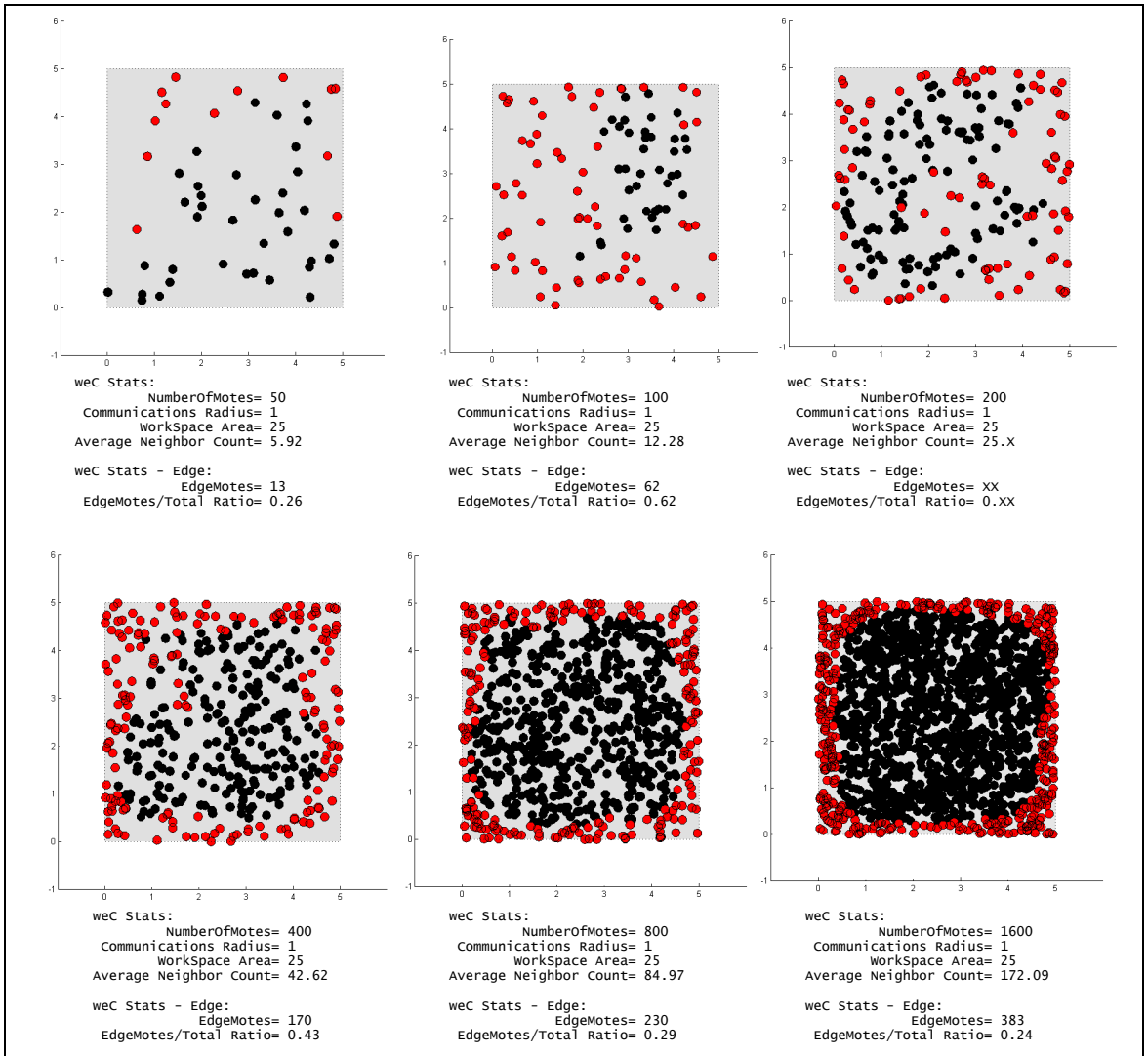


Figure 24: Effects of Neighbor Count on Edge Detection. The ability of the network to find edges varies with neighbor count if a constant local neighbor count edge threshold is used.

MaxNeighborPheromone.

In order to make an edge decision, all the motes compare their own personal neighbor count with the minimum and maximum pheromone messages they are storing. If their personal count is less than…

$$\text{CalculatedThreshold} = \frac{\text{MinNeighborCount} + \text{MaxNeighborCount}}{2} \qquad (24)$$

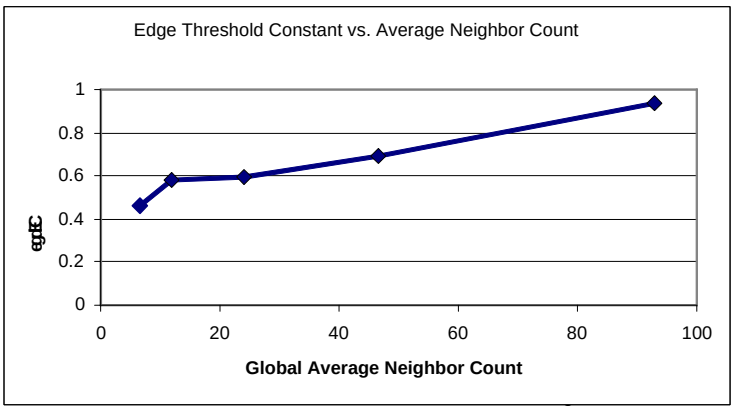
…then they become an EdgeMote. This quantity was not determined by any rigorous technique. The actual threshold varies as a function of neighbor count.

4.7.3.3 Effects of Neighbor Count of Edge Detection

The threshold value from the above equation is not independent of the global average neighbor count. Figure 24 shows how edge detection varied with neighbor count. One solution is to make our threshold value a function of neighbor count. Our threshold equation becomes:

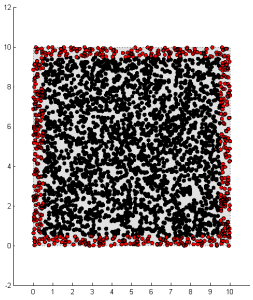
$$\text{Threshold} = C_{\text{Edge}}\left(\overline{\text{GlobalNeighborCount}}\right)\frac{\text{MinNeighborCount} + \text{MaxNeighborCount}}{2} \qquad (25)$$

We conducted several simulation runs where we defined which mote was an EdgeMote based on their X-Y positions, not their average neighbor count. From a histogram of the output, the neighbor count value where half of the motes were EdgeMotes and the other half were normal motes was used as our threshold neighbor




```
weC Stats:
                NumberOfMotes= 3200
         Communications Radius= 1
                WorkSpace Area= 100
       Average Neighbor Count= 92.9281

Position Stats:
           Edge Calculation Method= Zone
                        Zone Ratio= 0.05
                         EdgeMotes= 603
              EdgeMotes/Total Ratio= 0.19
  Average EdgeMote Neighbor Count= 65.53
Average NormalMote Neighbor Count= 100.53
```



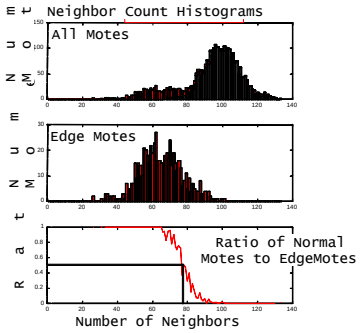


Figure 25: The graph on the top shows how C_{edge} varies as a function of global average neighbor count. The figures on the bottom are an example of the trials use to produce the graph. The two histograms on the right hand side show the number of motes (Y-Axis) that can communicate with a given number of neighbors (X axis). The graph at the bottom right shows the ratio of EdgeMotes to normal motes at all values of neighbor count. The black line is at the 50% mark, meaning that there are more EdgeMotes than normal motes to the left, and vise versa to the right. This critical neighbor count is where the threshold should be for this particular network. We can then compute C_{Edge} as the ratio of the calculated value and the measured value.

count. Knowing this threshold neighbor count and the min and max of the neighbor count distribution we can calculate C_{edge} for that global neighbor count.

$$C_{Edge} = \frac{ActualNeighborCountThreshold}{CalculatedNeighborCountThreshold} \quad (26)$$

However, this relation is of dubious utility for two reasons. First, measuring the average global neighbor count cannot be done effectively in a distributed fashion. Secondly, we only tested square networks with a constant edginess as defined at the beginning of this section, and threshold is probably also a function of the edginess of network.

4.8 Path Projection

Our previous algorithms allow our smart dust network to determine when targets are entering and leaving its coverage area, locate these targets in physical space, and communicate all this information to the user in a efficient manner. The next ability to add to our repertoire is to be able to predict where detected targets are heading.

Knowing what direction targets are heading it can be used to provide distant motes with warning, so that they can be start looking for the new target. This information can also be relayed to the user, although the more accurate position

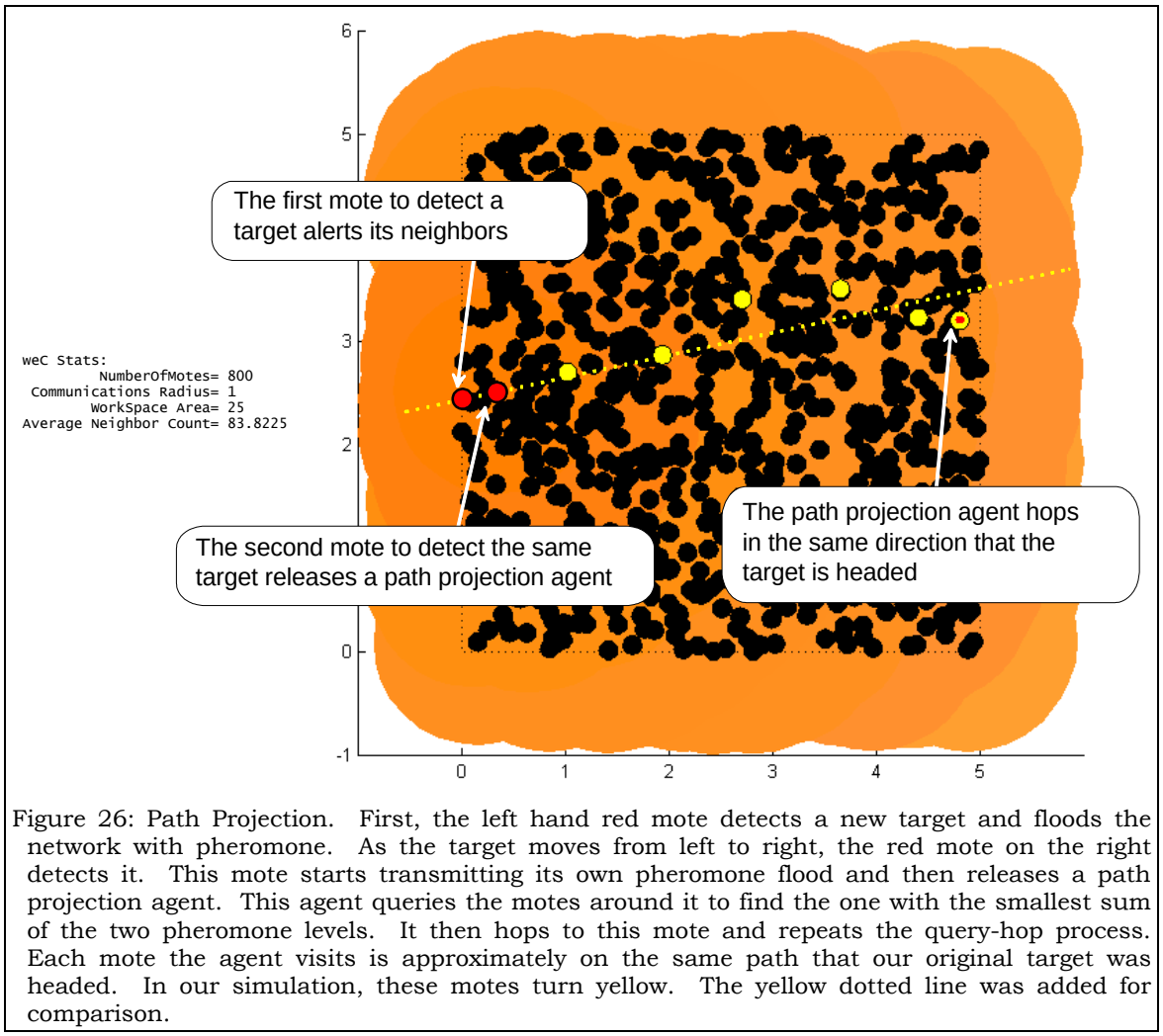


Figure 26: Path Projection. First, the left hand red mote detects a new target and floods the network with pheromone. As the target moves from left to right, the red mote on the right detects it. This mote starts transmitting its own pheromone flood and then releases a path projection agent. This agent queries the motes around it to find the one with the smallest sum of the two pheromone levels. It then hops to this mote and repeats the query-hop process. Each mote the agent visits is approximately on the same path that our original target was headed. In our simulation, these motes turn yellow. The yellow dotted line was added for comparison.

estimation data taken over time would probably be preferred.

The technique for path projection is essentially the same as the one described by Coore [18] for ray generation. The first mote to detect a new target floods the network with a PathPheromone message with its unique ID. We will assume that messages propagate through our network faster than the target is moving. The second mote to detect the same target floods the network with its own unique PathPheromone message, then releases a path agent. The agent movement algorithm samples all the neighboring motes and sums all of their PathPheromone levels. It then picks the mote with the lowest level and hops to that one. It repeats this Query-Hop process for as long at the path needs to be projected, as shown in Figure 26.

The pheromone diffusion from the first mote sets up a radially divergent "vector field" throughout the network, which can be seen in Figure 27. This straightness of the lines in this field is a strong function of average neighbor count. Denser networks will have straighter lines. Since the Path Agent will always hop towards lower levels of PathPheromone, it will move away from the first mote in approximately a straight line. Since the mote that actually releases the agent sits on one of these flow lines, once the agent starts hopping, it will continue to follow that line.

4.8.1 Failure Modes

The distance between the two motes is important, as it the neighbor count of the network. Also, we need to have two motes that can identify and classify a target well enough to realize that it is the same target. In addition, if this target excites multiple motes, how will they nominate one to do the diffusion and another to release the path agent?

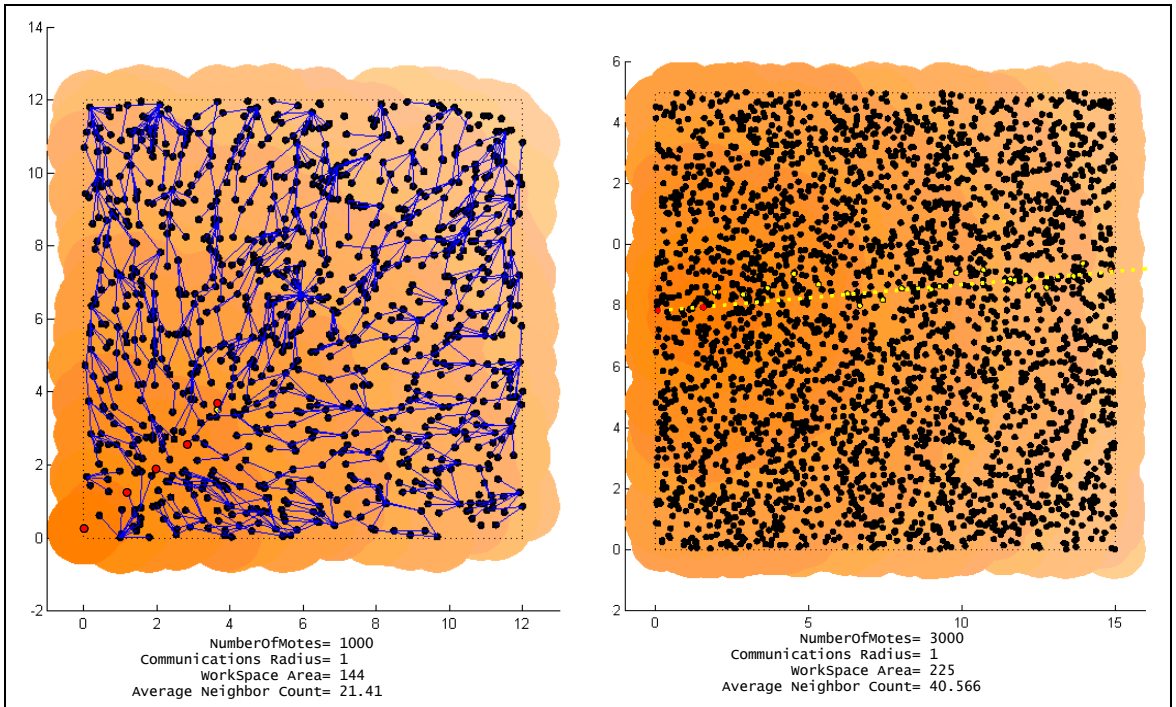


Figure 27: The left-hand figure is the path projection "vector field". The blue lines show which mote an agent message will hop to from any other mote. All the hops will be away from the red motes in the lower left that received the sensory data. The field is similar to that of a point charge, except squashed" in the direction of the projected line. The fact hat all the lines head away from the source makes the choice of mote where the agent is released important. The figure on the right is another path projection run with a large number of motes.

4.9 Algorithm Summary

In this chapter, we have presented several distributed algorithms that would be of use to a sensor network. Each technique places different constraints and assumptions on the distribution of the sensor node. To summarize:

Algorithm	Need Uniform Distribution?	Average Neighbor Count Needed
Message Diffusion	No	Doesn't Matter
Directed Communication	No	Doesn't Matter
Relay Network Division of Labor	No	Doesn't Matter
Position Estimation	Yes	Medium-High
Edge Detection	Yes	High
Path Projection	Yes	High

Table 1: Summary of spatial distribution constraints of different algorithms.

All the algorithms that make spatial inferences need a relatively high neighbor count to smooth out the randomness of individual positions. The robustness of pheromone diffusion coupled with messages that decay over time provides output that is insensitive to disturbances.

4.10 Simulation Operation

The simulation can run in two different modes. In one mode, it functions with a command line interface, with commands being sent to and processed by all the motes in a sequential fashion. This centralized interface is useful for setting up pheromone distributions and specifying initial conditions for simulation runs. For example, here is the code for the maximum position estimation demo:

```
weCInit
MakeWorkspace(0,5,5,0)
MakeMotes(200,1);

disp('Pick some Basis motes, then press return');
SetClickMoteMode('Select BasisMotes');

pause
SetClickMoteMode('Display Mote Info');

disp('Diffusing All pheromones');
DiffuseUntilStable('All');

SetClickMoteMode('Compute Ph Position');
disp('Pick the mote who''s position you want to measure, then press
return');
```

This results in a static diffusion that you can then operate on by clicking different motes. Another example is the relay network division of labor program:

```
% Lots of initialization stuff goes here
Done=10;
while Done>0
   acount=0;
   RandomOrder=Randomize([1:length(Motes)]);
   for count=1:length(Motes)
      mote=Motes{RandomOrder(count)};
      if acount<=NumOfAgentsInBrownMotes &
            strcmp(MoteGet(mote,'NetworkState'),'CantSeeRelay')==1
         agent('Secret Agent',mote,[226 12 128]/255,{'Hi Mom'});
         acount=acount+1;
      elseif acount>NumOfAgentsInBrownMotes
         agent('Secret Agent',mote,[226 12 128]/255,{'Hi Mom'});
         acount=acount+1;
      end
      if acount>=NumOfAgentsInAllMotes
         break
      end
   end
   AllMotesDo('MoteDecayPheromone','RelayMote');
   NetworkCantSeeRelayNum=0;
   AllMotesDo('UpdateNetwork');
   AllAgentsDo('MoveUpNetwork','ChairMote','RelayMote');
   AllAgentsDo('NominateRelayMote',RelayMotePh);
   UpdateDisplay;
end
```

Most of the details here are not important. The underlined while-end statements control program execution. This program is a loop that will run forever. The commands with the gray highlights are processed by all the motes and agents sequentially.

Motes in real sensor networks will be processing and communicating asynchronously with respect to each other. In order to more accurately simulate their interactions, some of the programs were re-written from a more mote-centric point of view. For example, here is the demo for finding edges:

```
weCInit
%Square workspace, Area=25
MakeWorkspace(0,5,5,0)
MakeMotes(1600,1);
MoteCode='FindEdges';
weCRun
```

The last command transfers control to the `weCRun` function which evaluates the program referred to in **`MoteCode`**, in this case, **`'FindEdges'`** The code for `weCRun` follows:

```
function weCRun()
global TimeStep

TimeStep=0;
while 1==1
   for CurrentMote = 1:length(Motes)
      feval(MoteCode,Motes{CurrentMote});
   end
   for CurrentAgent = 1:length(Agents)
      feval(AgentCode,Agents{CurrentAgent});
   end
   UpdateDisplay;
   TimeStep=TimeStep+1
end
```

Essentially, this is just a loop that runs the method in the global variable **MoteCode** for each dustmote object, and then does the same for the agent objects. In this example **MoteCode** contains **'FindEdges'**. The find edges program passes messages between motes to determine the global minimum and maximum, then changes the color of the mote if its neighbor count is below threshold.

The major shortcoming of this evaluation technique is that it is still not asynchronous. Control is passed from one mote to the next. The next step in simulation fidelity would be to use a discrete event simulator package to handle message passing and algorithm execution. This step was not taken initially in an attempt to keep the simulation as simple as possible.

5 MacroMotes: A Hardware Simulation

Simulations are a useful design tool, but it is important to realize their limitations. The algorithms presented in this work were designed to function on distributed smart dust motes that live in the real world of radio interference, dead batteries, and flaky connectors. The macromotes are designed to provide a physical simulation environment that allows for rapid algorithm development, yet still incorporates all the wonderful problems of real hardware.

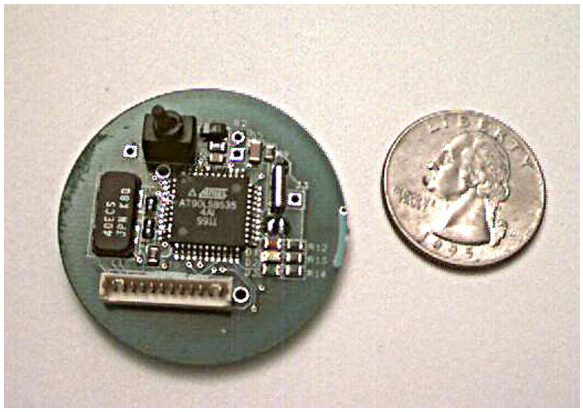


Figure 28: The Macromote designed for this work.

5.1 Hardware Overview

The macromote designed for this work is a variant of other designs from our group.[19] A block diagram of the hardware is shown in Figure 29. The full schematic can be found in the appendix, section 7.3.

The main processor is an Atmel AT90S8535. This chip has 8k of built-in EEPROM program memory, low power consumption, and many I/O pins. The drawback is that its memory architecture does not allow it to reprogram itself. To provide this ability, we added a programming processor, an Atmel AT90LS2343, and a shared EEPROM chip, a Microchip 24LC256. A RF Monolithics TR1000 900mhz transceiver enables nearby motes to communicate with each other. Our algorithms assume a circular communications area, so the antenna is designed to provide a RF field with the nulls pointed up and down, not side to side.

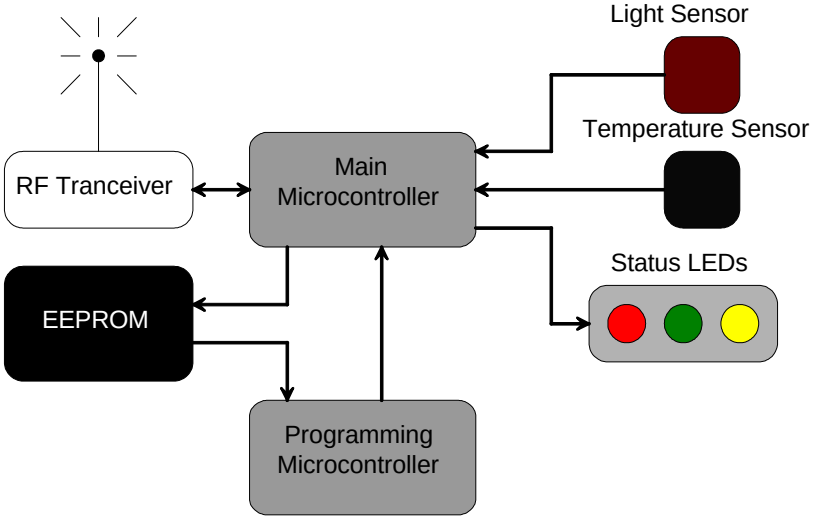


Figure 29: Block diagram of the macromote hardware. The main microcontroller is responsible for almost all the computational duties. To download new software, the user needs to download new software. This processor then puts the new code into the EEPROM, programs the main processor, and resets the system. This enables remote

The motes are equipped with a light sensor and a temperature sensor. The light sensor enables the user to stimulate/select individual motes with a flashlight, which is essential for debugging purposes. The temperature sensors allow us to measure a quantity that varies as a function of position in most physical environments. The LEDs allow the user to quickly determine what the network as a whole is doing.

5.2 Algorithms

Assuming a population macro motes spread out over an area larger that their communications range. If each mote transmits fixed-length packets periodically, the probability of a transmission collision with one neighbor is given by:

$$P = \frac{2 \cdot \text{PacketTime}}{\text{Re transmitPeriod}} \tag{27}$$

with multiple neighbors, the worst-case probability is:

$$P \leq \frac{2 \cdot \text{NumberOfNeighbors} \cdot \text{PacketTime}}{\text{Re transmitPeriod}} \tag{28}$$

Some more relations…

$$\text{PacketTime} = \frac{\text{BitsPerPacket}}{\text{BPS}} \tag{29}$$

$$\text{BitsPerPacket} = \text{NumOfMessages} \cdot \text{BitsPerMessge} \tag{30}$$

…and estimated numbers…

Bits per Second	19200
Message Length (Bits)	100
NumOfMessages	4
RetransmitPeriod (seconds)	2
NumberOfNeighbors	5

…gives us a worst-case collision probability of about 10%. This simple retransmit routine should be fine for initial experiments, but will need to be replaced with more sophisticated bandwidth sharing approaches as the average neighbor count increases. Once a communications infrastructure is implemented, the algorithms in this work can be ported to their new home and tested.

6 Conclusions and Future Work

6.1 Design Philosophies

You can develop a lot of insight[i] after watching your umpteenth simulation generate output at 3 a.m. In this section, I will do my best to articulate design philosophies and techniques that I found useful.

6.1.1 Scalability

The algorithm should be invariant to the number of motes. A system of 10 motes will not have the same properties that a system of 10,000 motes will have, but basic algorithms like diffusion, relay network formation, and position estimation still work fine. In contrast, our simple directed communication algorithm from section 4.3.3 does not scale with increasing network size because the list of motes that the message must keep track of will grow larger as the transmitter and receiver mote get farther apart. When I think about algorithms, I always imagine what the limit would be it there were Avogadro's Number of dust motes, i.e. how does your algorithm scale to the continuous distribution case?

All manner of random errors should be tolerated. Generally, I imagine how the algorithm would perform if 50% of the population suddenly stopped working. As long as there are no systematic failures and the hardware is homogenous, then motes are completely interchangeable. Lessons learned from natural systems also apply here. Often "good enough", not "perfect", seems to be the goal.

6.1.2 Robustness

Pheromone decay gives you stability – you always end up in the zero state. This gives great robustness to pheromone level and topology changes.

6.2 Future Work

This work touched on many related disciplines including communications, networking, computational geometry, graph artificial intelligence, and robotics.

Sensing

I have made grand assumptions that the motes will be able to determine when they have detected something, what they have detected. In reality, this is a very hard problem. The ability to share information with your neighbors opens up the possibility for more reliable sensory data, but brings with it the added complexity of distributed information management.

Learning: The "Distributed Database"

There are two extremes to storing information in a network like this. At one limit everybody only knows their own state. At the other extreme, everybody knows their state and has a copy of everyone else's state. Both are impractical, but it remains to be seen what the best way is to distribute information. The best solution is one that delivers effective local sharing of information to maximize "intelligence" and minimize network traffic. To talk about recognizing targets implies that you are also talking about recognizing the absence of one. The group would need to "habituate" to nominal sensory data in order to avoid false targets.

[i] Not to be confused with sleep deprivation induced hallucinations.

Because information could be spread all over the network, search agents that could seek out information sources, collect the necessary data. then head back to the mote that sent them would be very useful.

The diffusion algorithm is essentially a way of spreading routing information throughout the network. Each mote that has a pheromone knows who it originally came from, the next step in the chain to get a message back to that sender, but not the entire set of hops to move a message to the source.

Spatial Division of Labor

Coore's conclusion [18] says that we can divide our network into any spatial patterns we desire. This is a powerful technique for determining which motes will perform what tasks

General-Purpose Messages.

Pheromone messages and agent messages might prove too limiting for more sophisticated algorithms. A better solution is to communicate using small programs instead of data.

Statistics of Real Distributions

The assumption of uniform distributions was necessary for some of our algorithms, but probably does not accurately reflect real-world distributions. There are three different classes of distributions:

Uniform Density

Our assumption for this work.

Quasi-Uniform

Local density is a function of position, but not a strong one. The algorithms will work within error bounds defined by the variation in the density over the area they are operating.

Non-Uniform

Local density is unpredictable, or systematically flawed. This is a difficult problem and will probably need to be dealt with on a algorithm by algorithm basis.

Better Position Estimation

Average distance position estimation copmutes the line straightness constant, C, as a side effect of computing the position. However, C can be measured directly by computing the distances and hops between BasisMotes. Each BasisMote pair could then broadcast its value for C. An individual mote could take the average of these values, of use the value from the BasisMote pair that it is the fewest hops away.

Our position estimation algorithms only use information from a few BasisMotes and the number of hops of separation. It seems that estimates could be made in a more distributed fashion by relating your position to your neighbors. This would use more of the topology information available and might provide a better estimate.

Better edge detection

Edge detection becomes a trivial problem if you can determine direction information from your communications signals. Without a sensor upgrade, a more through understanding of the statistical distribution might enhance edge detection.

Using Min/Max information is not as direct as using a global average. Communication errors prevent that technique from working reliably. A clever solution would be to devise a system such that communications errors result in a 50/50 chance of gaining or losing charge [3]. That way, random errors would average out to zero and not affect the global amount of charge.

Estimating the Size of a Target

If a target excites several nearby motes, which all release a unique pheromone, its size can be estimated as follows:

$$\rho_{Local} = \frac{NeighborCount}{\pi \cdot r_{Communication}^{2}}$$

p=the number of different pheromones about this target that have been received, which is equal to the number of motes that have detected the same target

$$TargetArea = \frac{P}{\rho} = \frac{P \cdot \pi \cdot r_{Comm}^{2}}{NeighborCount}$$

As with path prediction, the difficulty will be in identifying the target.

Better simulation

While it is practical to build physical systems composed of dozens of macromotes, constructing systems of thousands is not. Therefore, more realistic simulations of sensor network operations will be needed to explore more of problems and solutions carefully. A discrete event simulator that does a better job of capturing the asynchronous operation of the motes is the logical next step.

Robotics

Depending on your personal biases, a community of Smart Dust Motes can be interpreted as a group of very slowly moving robots. Although some of the sensor network algorithms developed in this work may not lend themselves directly to robotics applications, the thought process certainty does.

6.3 Conclusion

Algorithms for distributed sensor networks allow many elements to perform tasks that are greater than the sum of their parts. Every step taken towards understanding these synergistic interactions further unravels the mystery between local behaviors and global results. The potential applications for this knowledge spread across all of science and technology. The ultimate goal of being able to define global goals and use these to specify local interactions is moving closer every day.

7 Appendices

7.1 Appendix A: weC code

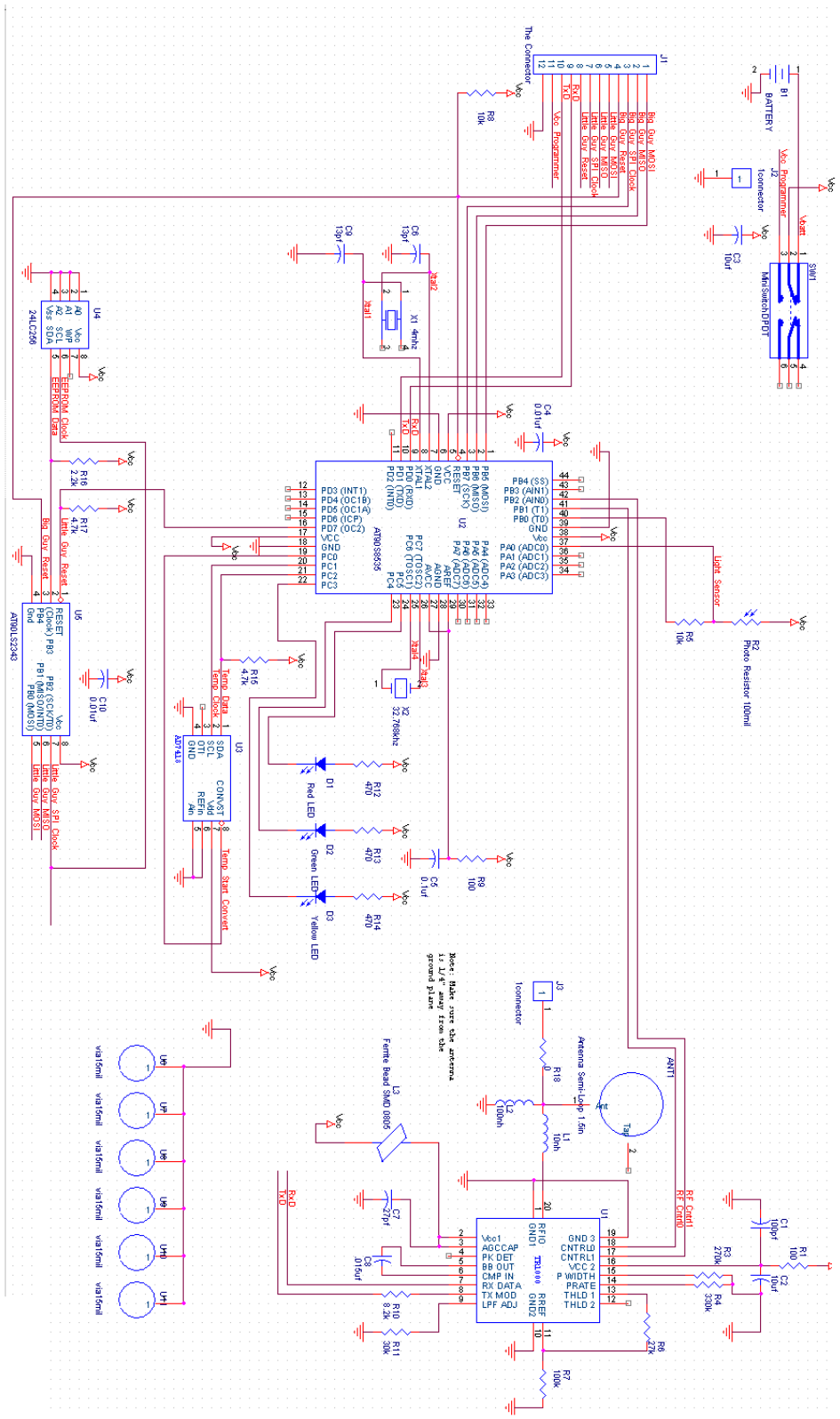
See http://www.eecs.berkeley.edu/ for more information

7.2 Appendix B: Macromote Parts List

Ref. Des.	Description	Manufacturer	Part Number
U1	Radio Transceiver	RF Monolithics	TR1000
U2	Microcontroller	Atmel	AT90LS8535
U3	Temperature Sensor	Analog Devices	AD7418
U4	EEPROM	Microchip	24LC256
U5	Microcontroller	Atmel	AT90LS2343
SW1	DPDT Micro switch	NKK	
J1	12 position micro connector	Walcom	
J2	Ground Clip		
J3	Whip antenna socket		
X1	4mhz Crystal		
X2	32.768khz Crystal		
D1	Red SMD LED		
D2	Green SMD LED		
D3	Yellow SMD LED		
C1	100pf cap		
C2,C3	10uf cap		
C4,C10	0.01uf cap		
C5	0.1uf cap		
C6,C9	13pf cap		
C7	27pf cap		
C8	0.015uf cap		
R1,R9	100ohm		
R2	Photoresistor		
R3	270k ohm		
R4	330k ohm		
R5,R8	10k ohm		
R6	27k ohm		
R7	100k ohm		
R10,R16	2.2k ohm		
R11	30k ohm		
R12,R13,R14	470 ohm		
R15,R17	4.7k ohm		
L1	10nh		
L2	100nh		

All discrete components in 0805 SMD form factor

7.3 Appendix C: Macromote Schematic



8 References:

1 Yeh, R., Kruglick, E., and Pister, K.S.J., "Towards an Articulated Silicon Microrobot", Proc. ASME Winter Annual Meeting, Dynamic Systems and Control, Vol. 2, Chicago, Nov. 6-11, 1994

2 McLurkin, James, "The Ants: A Community of Microrobots", Bachlor's Thesis Unpublished, MIT, 1995

3 Pister, Kristofer SJ, U.C. Berkeley Department of Electrical Engineering and Computer Sciences, Personal Communication

4 Morris, Steven, MLB Co., 1047 Amarillo Ave. Palo Alto CA, 94303 http://www.sirius.com/~mlbco/

5 Holldobler, Bert and Edward O. Wilson, "The Ants", The Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1990

6 Holldobler, Bert and Edward O. Wilson, "Journey to the Ants", The Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1994

7 Grossman, Patricia, "Very First Things to Know About Ants", Workman Publishing, New York, 1997

8 Heinrich, Bernd, "Bumblebee Economics", Harvard University Press, Cambridge, Massachusetts, 1981

9 Seeley, Thomas D. "The Wisdom of the Hive : The Social Physiology of Honey Bee Colonies", Belknap Press, 1996

10 Purves, William K. Gordan H. Orians, H. Craig Heller, "Life: The Science of Biology", Sinauer Associates, Sunderland, Massachusetts, 1992

11 MATLAB, the Math Works, Natick MA

12 Brooks, Rodney, "Elephants Don't Play Chess", Robotics and Autonomous Systems 6, 1990, pp3-15

13 Cormen, Thomas H., Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", The MIT Press, Cambridge, Massachusetts, 1990.

14 Locally unique IDs for network formation – call deborah estrin

15 Aldus, David J., U.C. Berkeley Department of Statistics, Personal Communication

16 Preparata, Franco P. and Michael I. Shamos, "Computational Geometry", Springer-Verlag, New York, 1985

17 Lynch, Nancy A., "Distributed Algorithms", Morgan Kaufman Publishers, San Francisco, CA, 1997

18 Coore, Daniel N., "Botanical Computing" Ph.D. Dissertation, MIT, 1999

19 Hollar, Seth, MacroMote Designs, http://www-bsac.EECS.Berkeley.EDU/~shollar/macro_motes/macromotes.html