# Enabling Path Planning and Threat Avoidance With Wireless Sensor Networks

by

Adam McLendon Eames

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005



Author .....
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by .....
Jonathan Bachrach
Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

**Enabling Path Planning and Threat Avoidance With Wireless Sensor Networks**

by

Adam McLendon Eames

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Wireless sensor networks can provide real time navigation instructions to robots or people attempting to travel in hazardous environments. This thesis presents the design, analysis, and implementation of a distributed system providing path planning and threat avoidance capability to mobile users. Contributions of the system include a unique framework for modeling for the effects of threats as well as original algorithms for discovering the safest path between any two points in the network. The prototype implementation was built using the Cricket v2 / TinyOS platform, and the results from tests of the implementation are presented.

Thesis Supervisor: Jonathan Bachrach
Title: Research Scientist

# Acknowledgments

I would like to thank my supervisor, Dr. Jonathan Bachrach, for his support and guidance. I would also like to offer thanks to Dr. Howard Shrobe for his help in defining the project and clarifying my ideas. This research is supported by DARPA under contract number F33615-01-C-1896.

Michel Goraczko of the Networks and Mobile Systems Group graciously lended the Crickets for development and testing. The work of Prof. Daniella Rus inspired my project–I would like to thank her for taking the time to talk with me about my ideas.

Chris Taylor, Buddika Kottahachchi, and Andy Perelson helped the days go faster in lab. I would especially like to thank Chris for his assistance with the multilateration algorithm.

Finally, I would like to acknowledge the support of my family and friends these past 5 years–I couldn't have done it without them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This thesis describes the design, implementation, and analysis of a distributed system providing threat avoidance capability to mobile users. The system utilizes recent advances in wireless sensor networks to identify and localize threats, determine a safest path between two locations, and guide a robot or person along the path. Unreliable communication, limited energy, and other limitations of sensor networks present challenges in the design process and offer opportunities for optimization in different scenarios.

This document is structured as follows: the introduction continues with background information detailing recent advances in the field of wireless sensor networks and related applications. Motivation for this project and its specific contributions are offered, and previous work in the field is discussed. The system's scope and design is explained within the second chapter, and analysis of its algorithms follows in the third. The final chapter details the functionality, architecture, and performance of an implementation of the system using the TinyOS / Cricket v2 platform.

## 1.1 Background

The development of relatively inexpensive hardware for wireless sensor networks has enabled many new distributed applications in recent years. Environmental monitoring, battlefield awareness, and industrial control are some of areas that are already

benefiting from advances in this new technological field. A typical node in a wireless sensor network is equipped with a battery, a microcontroller, a low power radio, a small amount of random access memory, and a collection of inexpensive sensors. Each node can communicate with its neighbors, and typically forms an ad hoc mesh network by acting as a router for the data of other nodes. Within the academic community, the mote platform developed at the University of California, Berkeley, has become the standard in the academic community for prototyping and deploying new sensor network applications. The motes run TinyOS [4], a lightweight, component-based operating system that allows applications to be constructed from a combination of modules. Using future generations of motes, researchers foresee the ability to deploy networks containing thousands of nodes throughout buildings and cities and integrate them with our preexisting communication infrastructure. Once in place, these networks could aid in disaster recovery, decrease energy use, and provide new insights into our world by monitoring environmental conditions. As the cost of building and deploying motes declines, we can expect to witness exciting new applications of wireless sensor networks.

Despite their promise, sensor networks suffer from several technological limitations and challenges. Radio-frequency (RF) communication is inherently unreliable, and inconsistency over time ensures that applications will face a varying set of neighbors they can communicate with. The reliance on battery power ensures we must design applications that conserve energy and tolerate failures of random nodes. We must also be aware of the limited processing power available at each node; in many cases, analysis of data gathered by a network must be performed later on a powerful, centralized computer. Communication bandwidth is also in short supply–nodes in geographic proximity must share the same channel over time, regardless of whether their broadcasts are intended for each other. Some of these issues may be ameliorated in the future with technological advances, but most will likely remain as constraints to be faced by application designers.

Looking forward, the increased feasibility of real-world sensor network deployments promises to allow a broader range of practical applications. In military set-

tings, temporary networks will likely provide intelligence on battlefields of the future, and office buildings may benefit from improved security and more sensitive HVAC systems. Environment researchers hope to develop less invasive observation techniques with sensor networks, and some success has already been had in this area [8]. Proponents of ubiquitous computing envision a world in which networked sensors surround and improve our daily life–the prescience of this vision may depend more on commercial limitations than technological ones. Regardless, it is likely that the power conservation, localization, and routing algorithms developed in recent years will be fundamental elements of future applications .

## 1.2 Motivation

The introduction of sensor networks to motion planning and threat avoidance scenarios is a natural extension of previous work. The possibility of distributed sensor nodes providing intelligence to a mobile agent has applications in many scenarios. For instance, a network that can identify heat in a burning building can be programmed to quickly find and disseminate the best escape routes to its occupants. Similarly, a network capable of recognizing enemy movement and determining relatively safe locations and routes in an urban warfare setting could provide real-time feedback to soldiers. Robots exploring treacherous natural environments could be aided by an ad hoc network, even if deployed haphazardly. Beyond these scenarios, improved algorithms for motion planning and path discovery have the potential to find widespread use as sensor networks become more cost effective and feasible for new applications.

## 1.3 Contributions

The primary contributions of this work lie in the techniques used to model threats, the algorithms for discovering safest paths, an analysis of the system's performance, and a prototype implementation. Briefly, paths are evaluated with a unique "survival probability" metric that is influenced by the severity of the threats as well as the

length of the path they affect. Each threat contributes to a model that approximates the likelihood of survival when a mobile user is traveling through the space covered by the network. The system takes advantage of the unique capabilities of sensor networks by performing local computation and optimizing partial paths in the background. A unique method for generating a discrete set of points to cover the space ensures that we search in regions with the lowest threat intensity. The path search technique is a back propagation algorithm that uses dynamic programming to discover an optimal path.

## 1.4 Related Work

Several techniques have been developed to allow robots to translate qualitative instructions into specific movements, a challenge generally referred to as the motion planning problem. It is described in more depth in [6]. Recent advances in navigation using sensor networks draw on one or more of the following techniques. Roadmap methods attempt to simplify the search space into a connectivity graph that can then be searched using traditional algorithms such as A$\star$. Of course, the size of the paths in the graph must be large enough to accommodate the robot. Planning methods utilizing exact cell decomposition attempt to subdivide the space into adjacent, non-overlapping cells that can be combined to construct paths. Approximate cell decomposition methods are similar, except they only provide an approximation of the search space and are more prevalent in practice because they are easier to implement and more tolerant of geometric errors. Finally, potential field methods take an iterative approach and typically rely on the robot sensing danger or obstacles in real-time. After calculating a potential field according to the sum of attractive or repulsive forces, the robot moves some incremental distance in the best direction. Rather than guaranteeing success in reaching a long distance goal, potential field methods are best suited for local movements where efficiency is important.

Recent work in robot navigation has focused on several challenges. In monitoring and security applications, mobile robots may be asked to cover a large, unfamiliar

environment as efficiently as possible. While some coverage applications require a static pattern of movement, others demand that that robots react dynamically to changes in the environment. For instance, the topology may be changing rapidly due to an emergency situation and robots must cover the entire network and track those changes. As a precursor to the coverage problem, robots may also be charged with deploying a sensor network and must be able to intelligently place sensors as they travel. In all cases, robots must be able to discern between previously explored and new regions, a process aided by deploying a series of markers or sensors.

Once a sensor network has been deployed, it can be used as a communication medium for robots and people in its vicinity. Nodes can collaborate to help a robot navigate its surroundings and travel between points in the network. Early work allowed robots to lay trails of sensors behind them that could be followed later [10]. Later research allowed sensor networks to determine the best series of nodes to follow in order to travel between locations and then advise a mobile robot where to move in real-time [2]. Such navigation can be done probabilistically by choosing the intermediate sensors with the best chance of leading to the goal using a technique known as Value Iteration. The utility derived from transitions between nodes forms a basis for computing and then maximizing the expected gain in utility for any movement. In this model, the navigation strategy can be viewed as a Markov decision process where the robot moves from one sensor to another [1]. In order to develop a thorough understanding of the spatial relationships between different sensor nodes, the robot must traverse the network several times before navigation can begin.

Strategies similar to the work presented in this thesis for utilizing a sensor network to plan routes for a user in a dangerous environment have also been developed. To tackle this problem, the network must be capable of identifying and localizing threats. Researchers have advocated using a potential field method where a measurement of danger can be propagated throughout the network and used to plan routes [7]. As the user travels through the network, virtual attractive and repulsive forces ensure that the safest path is followed in real-time. In a different scenario, the potential field can be used to plan a safest route using dynamic programming. The network responds

to a request for a path by building progressively longer partial paths from the source to the goal. In this scenario, when a sensor node receives a partial path it adds the danger level at its location and forwards a partial path assuming it has not already forwarded a path with a lower aggregated danger level.

Similar work has investigated using a sensor network to guide an autonomous model helicopter [3]. The authors took a path-oriented approach, where the helicopter is viewed as a mobile node that can request a route to a goal location. Using a control algorithm, localized sensor nodes establish whether they lie along the path according to their location and the path width, and then they guide the helicopter through their potential field as it travels along the route. The fidelity of the approach is limited by the distribution of the nodes, because paths must be constructed using node locations as intermediate points.

## 1.5 Comparative Analysis

The primary advantages of the approach presented in this thesis lie in the sophistication of the survival probability metric and the search technique. Unlike a technique of summing a measure of the threat, the survival probability metric offers a more accurate evaluation of the fitness of a path. We are able to concatenate paths by multiplying their survival probabilities, a process used often in the back propagation algorithm. Mathematically, the multiplication is equivalent to evaluating the probability a user survives a journey along the second path, given that he or she has successfully completed a journey along the first path. An ideal path search algorithm would have minimal latency and reliably find the path with the highest possible survival probability through the continuous space. Given the communication, sensing, and computational limitations of a sensor network, we must make sacrifices in these metrics. As shown in Chapter 4, the latency and reliability of our approach compares favorably with other strategies. The set of discrete points we will search from are chosen in a distributed manner, with each node evaluating the best position for points in its local space. Furthermore, survival probabilities for small paths between

these points are pre-computed, allowing the back propagation algorithm to efficiently combine these paths from node to node across the network. The local computation takes advantage of the distributed processing power available in a sensor network and enables a finer grained search than other methods allow. In scenarios where several threats are present, iterative approaches to motion planning may recommend paths that become "trapped" between several threats, as will be illustrated later. The back propagation algorithm avoids this problem by searching the entire space before recommending a path.

# Chapter 2

# System Design

The architecture of the threat avoidance system and its component algorithms is described in the subsequent sections.

## 2.1 Overview

We assume a two-dimensional field covered by a collection of connected sensor nodes arranged in random locations. Each node can communicate unreliably via packet radio with its neighbors, depending on the density and arrangement of the nodes. Our goal is to enable the sensor nodes to offer real time navigation instructions and path planning to a mobile robot or person traveling in the field. At any given time, the sensor field may contain one or more threats. We assume the sensors on each node can reliably detect these threats within some radius. In order to provide navigation instructions and guide mobile users, we must develop algorithms to solve the following problems:

- Model the affects of the threats on users traveling in the space covered by the network
- Detect and localize threats and ensure awareness throughout the network
- Discover the safest route for a user to follow in response to a request

- Guide the user along the path in real time

### 2.1.1 Assumptions

A shared coordinate system on the sensor nodes is an important pre-condition for the threat avoidance system. Nearly all algorithms discussed later make use of this foundation. For instance, nodes that detect a threat must be able to collaborate with their neighbors to estimate its location and then distribute that information to different regions of the network. Localization in sensor networks has been studied extensively in recent years, and techniques exist for developing a shared coordinate system. In some instances, a subset of the sensors are equipped with GPS or seeded with static coordinates, while other strategies rely on inter-sensor ranging data and multilateration algorithms. Inter-sensor ranging capability requires specialized hardware, such as the Cricket platform, to perform time distance of arrival (TDOA) measurements.

## 2.2 Threat Model

The sensor nodes making up our network store a distributed model of the threats they are capable of detecting. This model describes how threats affect users traveling through the field covered by the network, and therefore allows sensor nodes to evaluate the relative safety of paths. We base the threat model on a survival probability metric– for any path through the space covered by the network we calculate the likelihood that a mobile user following it will complete the journey successfully. The model is supported by a collection of functions that address the following challenges:

- Evaluate the survival probability when traveling a unit distance at some radius, $r$, from a threat
- Aggregate this calculation to support situations when multiple threats are present
- Allow travel distances other than the unit distance
- Support the concatenation and extension of paths

We begin by modeling the survival probability, $s$, when traveling a unit distance at a radius $r$ away from a threat by the following relationship:

$$s = 1 - b \cdot e^{-d \cdot r} \tag{2.1}$$

where $b$ and $d$ are constants dependent on the specific threat. The graph in Figure 2-1 demonstrates how this relationship is realized in practice.

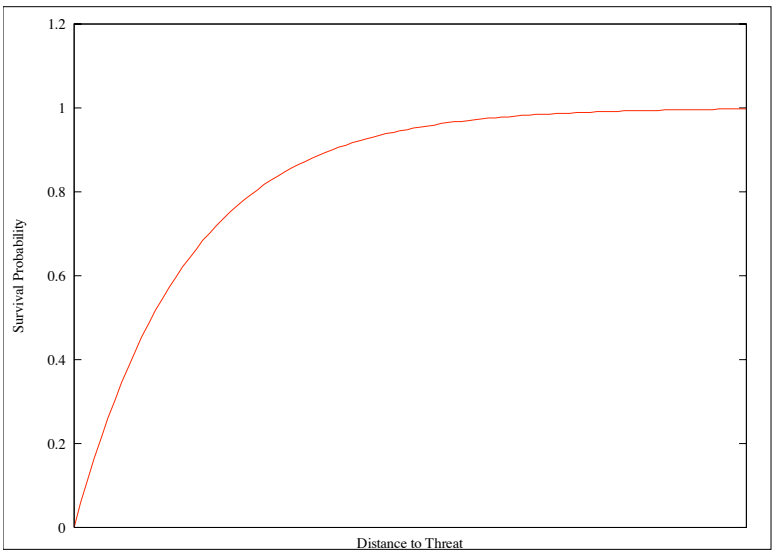


Figure 2-1: The relationship between survival probability and distance in the model.

When multiple threats affect a region of the network, we aggregate their effects by multiplying the survival probabilities calculated from their respective distances to the region. Each successive multiplication step is equivalent to calculating the probability a user will survive the effects of that threat, conditioned on surviving the threats already incorporated in the calculation. More formally, the calculation for multiple threats can be expressed as

$$s = \prod_{i \in threats} 1 - b_i \cdot e^{d_i \cdot r_i} \tag{2.2}$$

To allow for travel distances other the unit distance, we can convert the survival probability using an exponent. For example, if $l_s$ is the length specified by the original

relationship and $l_a$ is the actual length needed in practice, the survival probability is

$$s = (\prod_{i \in threats} 1 - b_i \cdot e^{d_i \cdot r_i})^{\frac{l_a}{l_s}} \tag{2.3}$$

Intuitively, this calculation is the equivalent of conditioning the survival of each additional unit distance of travel on the successful completion of the previous portion of the journey.

Assuming a sensor node has been informed of the threats detected by the network, it can perform the aggregation itself and calculate survival probabilities of short paths through the area near its location. To extend the calculation to a longer path, we multiply the probabilities of its component segments. As with the modification from the unit distance, this computation can be viewed as the conditional likelihood of a user surviving an additional distance, given that it has traveled successfully so far. If we were to view the calculation in the limit as the length of the unit distance approaches zero (and the constants in the survival probability function are modified), we are effectively calculating a product integral of survival probabilities along the path.

The ability to aggregate the effects of multiple threats, calculate a survival probability for a path of arbitrary length, and combine the survival probabilities of two or more paths is used extensively within the path discovery algorithms.

## 2.3 Threat Detection and Awareness

The ability to enable complex distributed applications using relatively unsophisticated hardware is one of the advantages of wireless sensor networks. We use a binary detection model and estimate the position of the threat by calculating the centroid of locations of the sensors that have detected it. This strategy generalizes to different kinds of threats, and fits well with inexpensive sensors. Furthermore, it produces more accurate results as the number of sensors detecting the threat increases. Sensor failures due to the threat may be likely in some cases, but nodes along the edge of

the afflicted region will continue with detection.

In practice, the centroid calculation requires collaboration between all of the sensors detecting the threat. We assume that the maximum sensor detection range is substantially smaller than the radio communication range. Leader election algorithms for tracking applications in sensor networks are well suited for this task, and the one developed by Nagpal and Coore is appropriate [9]. Briefly, when a node detects a threat, it waits a random period of time before assigning the threat a unique identification number and then broadcasts a group invitation message to its neighbors. Each of them that has also detected the threat responds with a update message. Pseudo code for the process once a detection is made is shown in Figure 2-2.

```
if status = NO_GROUP then
  if heard a leader broadcast recently then
    status ← FOLLOWER
    broadcast follower update message
  else
    start leader election countdown timer
  end if
else if status = LEADER then
  broadcast leader update message
else if status = FOLLOWER then
  broadcast follower update message
end if
```

Figure 2-2: Pseudo code for the response to a threat detection. The *status* variable is initially *NO_GROUP*.

If the leader election countdown timer expires, the node declares itself a leader and sends an update message. When other nodes receive this message, they note that they have heard a leader broadcast recently and stop their leader election countdown timer if it is running. In the case where the leader bails out, the group dissolves and each member begins a random countdown before broadcasting an invitation to join a new group. In this scenario, a node that is beyond radio range of the group leader will attempt to initiate another group. We minimize communication by not allowing any node to be a member of more than one group simultaneously. This decision ensures that a threat detected over a wide area of the network will be reported in multiple

locations. Pseudo code for the response to no detection is shown in Figure 2-3.

```
if status = LEADER then
   broadcast leader bailout message
   status ← NO_GROUP
else if status = FOLLOWER then
   broadcast follower bailout message
   status ← NO_GROUP
end if
```

Figure 2-3: Pseudo code for the response to no threat detection.

Periodically, the group leader computes the threat location by calculating the centroid of the locations of all the nodes in the group. After each calculation, the leader broadcasts its own identification number, the location of the threat, and a sequence number so neighbors can distinguish between repeated broadcasts. Neighbors rebroadcast threat awareness messages if the survival probability of a user traveling at their location could be impacted by the threat. In this way, we achieve a controlled, local flood without impacting areas of the network unaffected by the threat. The leader identification number allows remote nodes to avoid duplicating threats and remove stale threats after a period of time.

## 2.4 Search Space

Discovering the safest path through the field covered by the network requires a distributed search algorithm. Naively, this is a search through a continuous plane that is intractable given the computational limits of a sensor network. In order to decrease the computation involved, we must choose a subset of the points in the field as our search space. Paths will be constructed incrementally using these points as intermediate locations. Rather than select static sensor node locations as these points, we can achieve better results with a dynamic set of points that shift in reaction to threats. By locating individual points in areas of lower threat intensity, we increase the chance that the best path we find will have a higher survival probability.

### 2.4.1 Midpoint Shifting

The improvement realized by allowing the set of search points to shift is dependent on many factors. A higher spatial density of sensors increases the likelihood that their locations will fall in areas where travel is safer, but a higher density of threats increases the need for a more sophisticated placement of points. The classic scenario we wish to avoid is one where sensor locations clustered around threats prevent us from discovering an optimal path that bisects the threats. More generally, shifted points will improve the survival probability of a path and allow us to approach the optimal solution that would be found if the continuous space were searched. Figure 2-4 demonstrates the benefits that can be realized from using shifted midpoints rather than node locations as steps along the path.

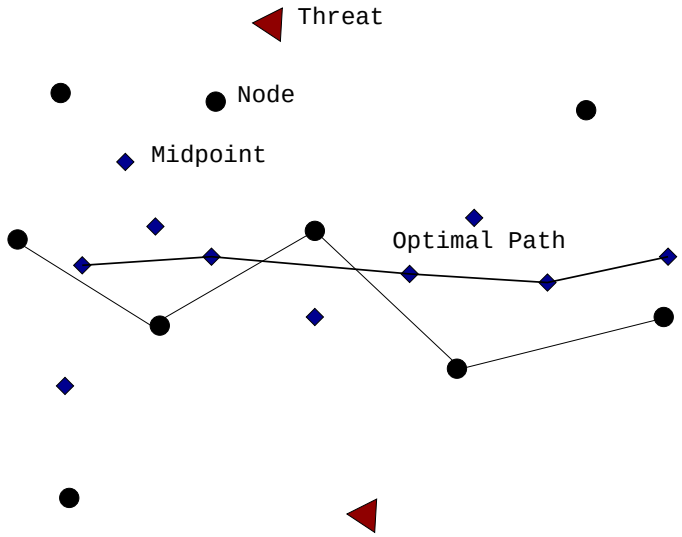


Figure 2-4: Benefits in coverage using midpoint shifting. Note that the optimal path lies roughly along the perpendicular bisector of the line connecting the locations of the threats.

Our strategy will be to incorporate one search point for each pair of neighboring sensor nodes. Each node will be jointly responsible for the location of a point between itself and each of its neighbors. The default location for each point is the geometric midpoint between the two sensor locations. Ideally, we would like to shift each midpoint so that it is located at the point with the lowest threat intensity in the

local area. In practice, however, we only see substantial benefits near regions which roughly bisect the two threat locations–we would prefer midpoints to lie as close to the perpendicular bisector of the line connecting the threat locations as possible. Each node that is aware of at least two threats shifts its midpoints to the perpendicular bisector, but only if the shift is a less than a specified distance away from the default location. To account for variations in node density, we specify this maximum shift distance as a quarter of the distance between the neighbors. Figure 2-5 illustrates an example shift, and Figure 2-6 contains pseudo code for the shifting process.

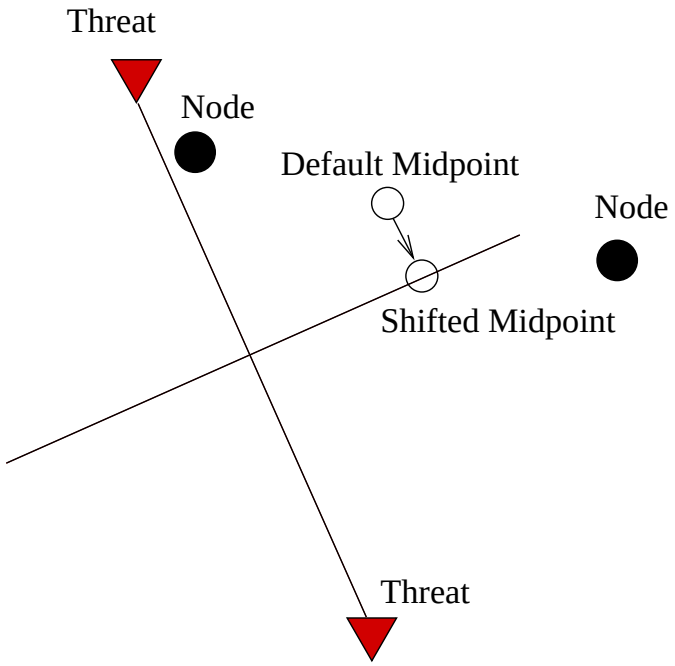


Figure 2-5: An illustration of a midpoint shift. The shifted midpoint lies in a relatively safer position, and will create a path of higher survival probability than if it were in the default location. Note that the midpoint is shifted in the direction matching the slope of the line connecting the threats.

### 2.4.2 Survival Probability Maintenance

Several unique properties of sensor networks influence our search strategy. The computational power of sensor nodes is plentiful relative to their communication bandwidth. Computation also consumes far less energy and can proceed in the background

$(dx, dy) \leftarrow$ geometric midpoint of your location and neighbor $i$
$(tx, ty) \leftarrow$ geometric midpoint of two nearest threat locations
$s \leftarrow$ slope of line connecting two nearest threats
$mx \leftarrow (tx + s * ty + s^2 * dx - s * dy)/(s^2 + 1)$
$my \leftarrow s * (mx - dx) + dy$
$distance \leftarrow$ distance from $(dx, dy)$ to $(mx, my)$
$maxDistance \leftarrow$ (distance from your location to neighbor $i$) / 4
**if** $distance \leq maxDistance$ **then**
  midpoint for neighbor $i \leftarrow (mx, my)$
**else**
  midpoint for neighbor $i \leftarrow (dx, dy)$
**end if**

Figure 2-6: Pseudo code for calculating and storing the midpoint location for neighbor *i*. (*dx, dy*) is the default midpoint location and (*mx, my*) is the proposed shifted midpoint, calculated by finding the intersection of the perpendicular bisector and a line through the default midpoint with the same slope as the line connecting the two nearest threats.

without interfering with other responsibilities of the threat avoidance application. For this reason, each node maintains a calculation of the survival probability associated with a path between each pair of its midpoints. Each of these segments corresponds with a connection between a pair of neighbors, and will be used to build longer, complete paths from neighbor to neighbor at the user's request. Each node is able to periodically recompute the survival probabilities between each pair of points without additional communication with its neighbors. Among other benefits, this strategy allows paths to be assembled more quickly because the necessary computation has already been performed. Each node is responsible only for its local area, and we are able to harness the distributed computational power of the network. As discussed earlier, path aggregation is straightforward–the probability that a user will survive a journey along a longer path can be quickly calculated by multiplying the survival probabilities of its segments.

Building this functionality requires several functions. **spAtPoint**, shown in Figure 2-7, is an implementation of Equation 2.2 and calculates the survival probability for a user traveling a unit distance at a location.

In order to calculate the survival probability of a path between two locations, we

**spAtPoint**($x$, $y$)
$sp \leftarrow 1$
**for** $i = 0$ to $numberOfThreats - 1$ **do**
  $distance \leftarrow$ distance from $(x, y)$ to threat $i$'s location
  $sp \leftarrow sp * (1 - b \cdot e^{d \cdot distance})$
**end for**
**return** $sp$

Figure 2-7: Pseudo code for calculating the survival probability of a path of unit distance at a point. The constants $b$ and $d$ are dependent on the specific threat.

average the survival probabilities at the locations using **spAtPoint**, and then adjust for the distance between the points as specified by Equation 2.3. The pseudo code for this operation is shown in Figure 2-8.

**spBetweenPoints**(*x1, y1, x2, y2*)
$sp1 \leftarrow$ **spAtPoint**$(x1, y1)$
$sp2 \leftarrow$ **spAtPoint**$(x2, y2)$
$sp \leftarrow (sp1 + sp2)/2$
$distance \leftarrow$ distance from $(x1, y1)$ to $(x2, y2)$
$distanceRatio \leftarrow distance/UNIT\_DISTANCE$
**return** $sp^{distanceRatio}$

Figure 2-8: Pseudo code for calculating the survival probability of a path between (*x1, y1*) and (*x2, y2*).

Finally, we must build a table of records for each pair of midpoints, as shown in Figure 2-9.

**for** $i = 0$ to $numberOfNeighbors - 1$ **do**
  **for** $j = i$ to $numberOfNeighbors - 1$ **do**
    **insert**(id of neighbor $i$, id of neighbor $j$, **spBetweenPoints**$(x_i, y_i, x_j, y_j)$)
  **end for**
**end for**

Figure 2-9: Pseudo code to store survival probability calculations between each pair of midpoints. $(x_i, y_i)$ is the midpoint with neighbor $i$, and $(x_j, y_j)$ is the midpoint with neighbor $j$.

## 2.5 Path Construction

Making use of the pre-computed segment survival probabilities stored in *records*, the network can discover the optimal path between any two arbitrary points covered by the sensor field using a back propagation algorithm. Initially, the network routes a path request originated by the user to the node nearest the destination. This node initiates the back propagation by calculating and sending the survival probability of segments connecting the destination with each of its midpoints. Each segment is the beginning of a possible path back to the origin. When each neighbor receives the path reply containing a set of advertisements, it first selects the partial path to its midpoint. To create the reply it will send, it builds a new set of advertisements by adding segments from the midpoint to each of its other midpoints. The survival probabilities of these segments have been pre-calculated, and can be appended quickly to form advertisements for each of its neighbors. Pseudo code for this process is shown in 2-10. Figure 2-11 illustrates this step in the reply process, as the current node extends the path from the midpoint of one neighbor to the midpoints of its other neighbors.

$sp \leftarrow$ survival probability of path * survival probability in advertisement
store ids of nodes along path so far and append your own
**for all** $i$ in set of neighbor ids such that $i \neq$ id of sender **do**
  lookup survival probability and store as advertisement for neighbor $i$
**end for**

Figure 2-10: Pseudo code for building a set of advertisements to neighbors in a reply message. The node must concatenate the survival probability of the short path to its midpoint with the rest of the path built so far before creating the new reply message.

We judge the fitness of a reply based on highest survival probability, followed by lowest hop count in the event of a tie. The decision of whether to forward a reply message involves several issues. If the node initiated a request that resulted in the reply, there is no need to forward–the reply can be stored if it is optimal. Naturally, a node does not forward a reply message if it has previously forwarded a better one. Pseudo code for the process that ensues when a reply message is received is shown in Figure 2-12.

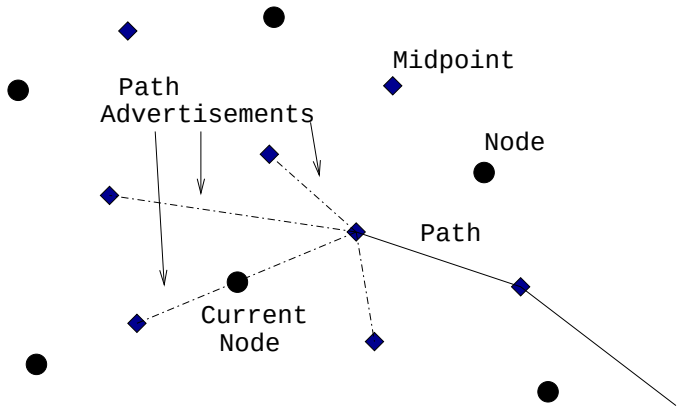


Figure 2-11: Path extension within the back propagation algorithm. The current node includes the survival probability of each extended path in its message.

$sp \leftarrow$ survival probability of path * survival probability in advertisement
**if** *sp* is the optimal reply to a request made by this node **then**
  store *sp* and the ids of the nodes along the path
**else if** *sp* is the optimal reply this node has received for this request **then**
  build advertisements for this reply and broadcast the message
**end if**

Figure 2-12: Pseudo code for the path advertisement forwarding process.

Due to the nature of the construction algorithm, the user may receive multiple paths, each with higher survival probability than the last. Because of this, the user should wait a period of time after a path arrives before declaring it optimal, in case a better path is received soon after. Once the period of time expires, the user broadcasts a path selection message which neighbors continue to forward if they have previously forwarded a path advertisement. This message serves the dual purpose of instructing nodes to stop forwarding path advertisements, and notifying those nodes whose midpoints make up the path that they will be responsible for guiding the user.

The construction algorithm will fail if communication failures prevent any paths from being extended back to the user. With multiple potential paths being extended simultaneously, this situation is unlikely. Regardless, we can ensure that it does not occur by having nodes rebroadcast their path advertisements if they do not overhear any of their neighbors forwarding on the advertisement. Although we cannot guarantee the optimal path will be found in all circumstances, this strategy will mitigate

the result when messages are dropped.

## 2.6 User Guidance

After the user initiates an activation message, the nodes along the path form an active group responsible for helping the user navigate between midpoints. In practice, the user will be equipped with a specialized node capable of ranging to other nodes in the network. As the user travels through the space covered by the network, the specialized node emits coordinated acoustic and RF signals that nearby nodes can use to calculate their distance to the user. These distances, as well as the locations of the nodes and their midpoints, are returned to the specialized node in a radio packet. At any given time, the user can estimate its own location using a collection of recent distances and locations. The estimation is done using a multilateration algorithm that refines an initial estimate through several iterations. The initial estimate used in this case is the centroid of the node locations, and pseudo code for the algorithm is shown in Figure 2-13. In the case of a human user, an interactive application displays the locations of nodes, midpoints along the path, and the user's estimated location in real time. The application also provides an interface for requesting paths, evaluating incoming replies, and selecting a path to follow.

$N \leftarrow$ number of recently reporting nodes
$h \leftarrow$ height of node traveling with user
$e \leftarrow$ centroid of the locations of the recently reporting nodes
**while** better estimate is desired **do**
  **for** $j = 1$ to $N$ **do**
    $d_j \leftarrow$distance from $n_j$ to $e$
    $f_j \leftarrow \sqrt{d_j^2 - h^2} - \|n_j - e\|$
    $g_j \leftarrow \frac{(n_j - e)}{\|n_j - e\|}$
    $A[j] \leftarrow g_j$
    $b[j] \leftarrow -f_j + g_j \cdot e$
  **end for**
  $e \leftarrow A^{-1} \cdot b$
**end while**
**return** $e$

Figure 2-13: Pseudo code for the multilateration algorithm. The **while** loop reflects the logic involved in the calculation; five iterations are sufficient in most cases. $n_j$ is the location of node $j$. $A$ contains $N$ rows and is a matrix of two dimensional vectors. $b$ is a one dimensional matrix of $N$ rows.

# Chapter 3

# Analysis and Modeling

Developing a framework for analyzing how well the system will perform in varying situations is an important addition to the design offered above. The following sections present analysis of various aspects of the system.

## 3.1 Threat Localization and Awareness

The accuracy of our threat localization technique is influenced by several factors. Chief among these are the detection radius of a typical sensor and the spatial density of sensor locations in the network. The group formation algorithm discussed earlier also influences accuracy, as it results in one threat location for each group of sensors reporting to a leader. In cases where a threat exists over a large, continuous area, several locations will be reported. When calculating the centroid of sensor locations to estimate the threat location, our approximation will tend to improve with each additional sensor that has detected the threat. Concentrating on the case where a threat can be modeled by a point in the plane and results in only one calculated location, we will examine the relationship between spatial density and the sensor detection radius with the error likely to result from the centroid calculation.

### 3.1.1 Localization Model

We model sensor locations as a continuous spatial poisson process with parameter $\lambda$ and assume a threat detection radius of $r_D$. In reality, the detection radius will vary across sensors and time, but using $r_D$ as the mean of the distribution of detection radii improves tractability without sacrificing the result. By the properties of a spatial poisson process, the probability of exactly $s$ sensors detecting a threat is given by

$$P(s = k) = \frac{(\lambda \pi r_D^2)^k e^{-\lambda \pi r_D^2}}{k!} \text{ for } k = 0, 1, 2, \ldots$$

It follows that the expected number of sensors detecting a threat is

$$E[s] = \lambda \pi r_D^2$$

Once a detection has been made and the centroid calculated, we know the actual location of the threat is located in the set of locations within the detection radii of each sensor. Figure 3-1 illustrates this situation, and provides some intuition on how uncertainty declines with the number of sensors detecting the threat. In general, with a linear increase in network density, detections are expected to rise linearly as well.

### 3.1.2 Awareness

The likelihood that all of the sensors within the region affected by a threat maintain consistent knowledge over time is influenced by several factors. The spatial density of sensor locations is directly related to this probability, as each node will have more opportunities to hear a report of the threat location as its neighbors increase. Similarly, increased RF transmission range will also increase the likelihood of consistency. Depending on the amount of communication traffic in the network, the frequency that group leaders broadcast threat locations is also relevant as it influences the amount of congestion in the local area.

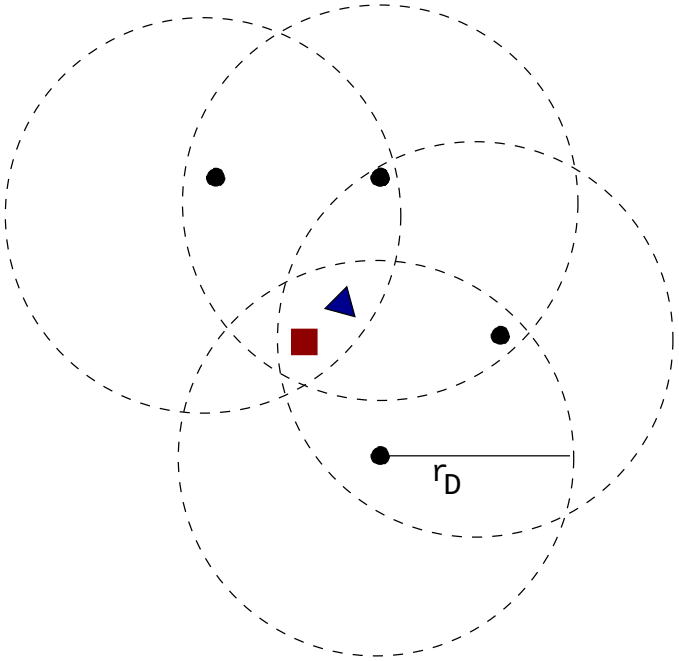


Figure 3-1: A visual representation of threat localization error. The circles represent sensor locations, the square the actual location of the threat, and the triangle the centroid. The area of the region where the four circles intersect represents the magnitude of the uncertainty.

## 3.2 Threat Model and Survival Probability

The survival probability metric coupled with a back propagation algorithm is one of several possible methods for selecting the optimal path given a series of threats. We examine other possibilities in this section and discuss their performance in various scenarios.

### 3.2.1 Iterative Gradient Descent

Rather than attempt to discover a path through back propagation, users may move iteratively through the field without long term planning. Typically, the direction of each step would be determined by a goal-seeking heuristic that attempts to maintain the maximum distance from the nearest threat. This strategy can be executed precisely with a localized network, but hop counts could serve as a proxy for distance if localization capability did not exist. In situations where few threats exist in the

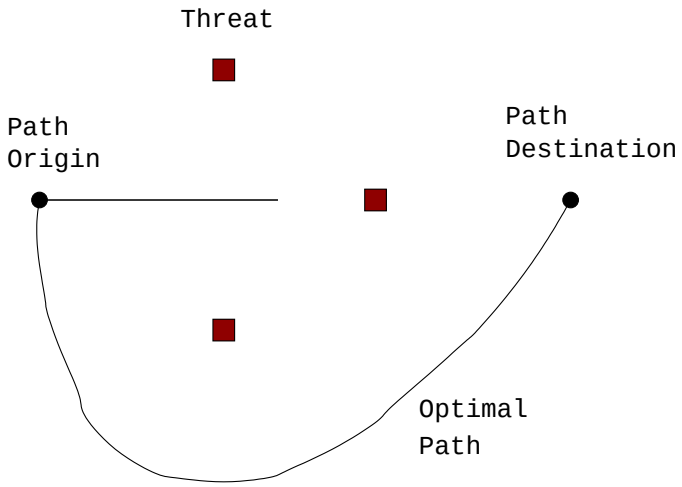


Figure 3-2: An example of a pitfall of the iterative gradient descent method. A user performing gradient descent is likely to follow the horizontal path, rather than travel around all three threats.

network, it is possible that this strategy could be less time consuming than waiting for back propagation to complete. Unfortunately, as it is performing a form of gradient descent, the strategy is likely to suffer from a local minima problem–without "looking ahead" it may lead users to inappropriate locations when multiple threats are present. Figure 3-2 illustrates a scenario where the problem rises; an optimal path around the threats is not likely to be found with an iterative method. As will be demonstrated below, the strategy of shifting midpoints between nodes whose locations approximately bisect threat locations creates optimal paths locally similar to those created by the iterative approach, but without the local minima problem.

### 3.2.2 Alternative Threat Representations

In order to evaluate the relative safety of a path, we can imagine adding a measure of the threat at each point on the path, rather than taking a product integral of survival probabilities. The summation procedure would take the form of a traditional integral, and could be implemented using the familiar back propagation algorithm. Intuitively, it does not distinguish between a severe threat encountered for a short distance and a less severe one present along a longer path. While total survival probability declines

exponentially with path length and threat intensity, the sum increases linearly as the path is extended or threats become more numerous. These two formulations may give similar results in many circumstances, but the survival probability metric is a more precise comparative measure of paths.

### 3.2.3 Midpoint Shifting

In order to justify the added computation and communication necessary to maintain optimal midpoint locations, it is important to examine the benefits of this process. The alternative is to naively select potential points to build paths from; sensor locations are a logical choice because no additional messaging is needed to maintain location state, but any strategy of selecting random locations should produce equivalent results. Intuitively, the shifted midpoints strategy is advantageous when sensors are located near threats, or in networks with low density and therefore fewer points to build paths from.

Analysis of the following simple example quantifies the advantage of the midpoint shifting process. As shown in Figure 3-3, we have threats located at (5, -5) and (5, 5) and a path request from (0, 0) to (10, 0). Sensor nodes are arranged randomly according to a spatial poisson process with parameter $\lambda_s$. Possible sensor locations are shown in the figure, but these are only representative. The straight line bisecting the threats along the origin demonstrates the result when the path is constructed with shifted midpoints–before back propagation occurs, sensors will be able to locate points in areas of highest survival probability and form an optimal path. The line connecting sensor locations represents an inferior path, but the best possible given the locations. As before, we model the survival probability at a distance $r$ from a threat as

$$s = 1 - b \cdot e^{-d \cdot r}$$

We are interested in developing an intuition for the difference in survival probabilities of the optimal and inferior paths. To calculate this value, we will first determine the expected vertical distance of each sensor location along the inferior path from the

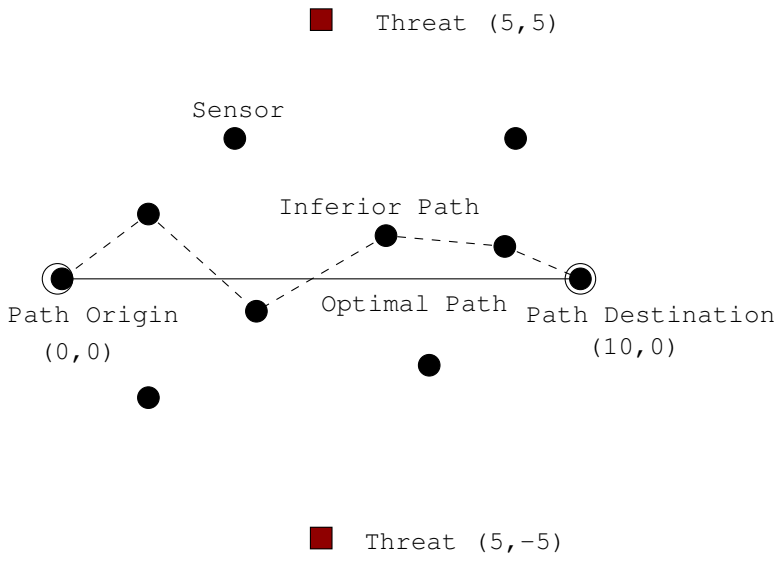


Figure 3-3: Example for midpoint shift analysis

x-axis. In order to ensure the problem is tractable we must assume a threshold of $r_T$ units beyond which every sensor has no neighbors. According to the spatial poisson process, each sensor's neighbors are distributed randomly within a circle of radius $r_T$ centered at its location (we therefore expect $\lambda_s \pi r_T^2$ neighbors). To simulate the result of the back propagation algorithm, we form the inferior path by iteratively selecting, among the neighbors closer to the origin, the one closest to the x-axis.

We can very closely approximate the expectation, $E[y]$, of the vertical distance of the selected node from the x-axis by developing a probability density function for the vertical component of its location. We know it will lie within $R_T$ horizontally, and the following expression relates the probability that $k$ sensors will appear in a rectangle of width $R_T$ and height $2y$ (above or below the x axis):

$$P(k) = \frac{(\lambda_s R_T 2y)^k e^{-\lambda_s R_T 2y}}{k!}$$

We are only interested in the sensor nearest the x axis, so the cumulative distribution function (CDF) of the smallest vertical distance is given by

$$F_D(y) = 1 - e^{-2\lambda_s R_T y}$$

Taking the derivative, the PDF is

$$f_d(y) = 2\lambda_s R_T e^{-2\lambda_s R_T y}$$

The expectation, therefore, is

$$E[y] = \int_0^\infty 2\lambda_s R_T y e^{-2\lambda_s R_T y} dy = \frac{1}{2\lambda_s R_T}$$

This series of steps tells us that we can expect a discrepancy of $\frac{1}{2\lambda_s R_T}$ from the ideal path at each sensor location along the inferior path. The difference in survival probability calculated at each step along the path is then

$$\prod_{i \in threats} 1 - b_i \cdot e^{d_i \cdot r_i} - \prod_{i \in threats} 1 - b_i \cdot e^{d_i \cdot r_i'}$$

where $r_i'$ is the absolute distance to threat $i$ given an expected vertical shift of $\frac{1}{2\lambda_s R_T}$ from the optimal path.

## 3.3 Performance and Scaling Issues

Given the limited communication and computation resources inherent in wireless sensor networks, it is important that we understand how the algorithms presented earlier scale. Our inputs to the system include the number of threats, the frequency of path requests by users, and the size (number of nodes and spatial density) of the network. Unlike in a traditional single processor computation model, we must be concerned with communication bandwidth utilized, energy expended, and computational complexity. The relation between these factors and overall system performance is not straightforward. Given the distributed nature of the system, communication in one geographic area of the network has no impact outside the broadcast radius. Similarly, excessive energy consumption leading to failures of individual sensor nodes has the potential to inhibit overall system performance, but the effect is unpredictable, especially if failure is correlated with geographic position. Our strategy will be to

examine the effects of the individual algorithms and then aggregate the results.

### 3.3.1 Detection and Awareness

The threat detection and awareness algorithms are employed whenever a threat is found by a set of sensors. Each is affected by the number of threats present and the network's density; path requests and network size do not influence the performance of the algorithms. For each threat, the detection algorithm creates a continuous series of local communication as nodes report to a leader who approximates the threat's location. Periodic announcements are flooded to nodes in the network within a fixed radius–as far as the threat affects the survival probability of a user traveling in the region. Consequently, each additional threat creates a fixed amount of communication for detection and awareness. As the density of nodes in a given area increases, the communication facilitating detection and flooding will rise proportionately.

### 3.3.2 Path Construction

The cost in terms of communication and energy consumption of the path construction algorithm is very significant, as it potentially involves a large percentage of the nodes in the network. Although additional threats do not impact the cost of running back propagation, the size and density of the network are important factors. Preventing nodes from advertising their path extensions when there is no possibility they will contribute to an optimal path is an important step. Clearly, frequency of path requests is the most important factor in this case, and network overload is a possibility if repeated requests are made in a short timeframe.

### 3.3.3 Conclusion

Although the path planning and threat avoidance application requires several algorithms and substantial resources, it is important to note that only the path construction component requires additional time to complete with larger network size. The threat detection, midpoint shifting, and survival probability algorithms run in local

areas and are not inhibited by larger networks. Because of the nature of the application, energy consumption is a difficult issue to tackle–we must maintain a sufficient number of aware sensors throughout the network over time so we can be confident of reliable threat detection. In the absence of a coordinated sleeping algorithm, incorporating future power saving features in hardware may be our best strategy.

## 3.4 Failure Tolerance

By using a wireless sensor network as a platform, it is likely our algorithms will be exposed to random hardware and communication failures. While the frequency of the malfunctions will depend on factors specific to the deployment, we should expect a fraction of the nodes to have failed at any given time. A time-varying radio frequency (RF) environment for communication renders predictions about which nodes received a particular message difficult to make. We make a distinction between sets of failures that are uncorrelated over space and time, and those that are correlated and thus possibly (in some cases) the result of a threat in one region of the network or an indication that all nodes are losing power. Ensuring our high level algorithms tolerate random issues facing a small percentage of the network is a reasonable goal, but it may be impossible to recover from highly correlated failures.

### 3.4.1 Node Failures

From a global perspective, node failures have the general effect of decreasing the average neighborhood size in the network. While this change may hurt the performance of the system by decreasing the expected number of nodes detecting a threat and the number of midpoints making up the search space, we should ensure the algorithms described in Chapter 2 are likely to recover from unexpected node failures. In particular, failures of the leader node within the detection algorithm will result in a reelection once the surrounding nodes purge the outdated leader announcements from their table. Failing follower nodes will be purged from the leader's centroid calculation, leading to a situation where malfunctions concentrated in a local area

would adversely affect detection capability. Within the midpoint shifting algorithm, failing nodes reduce the number of potential points covering the space but do not introduce inconsistencies as their neighbors drop them over time. A malfunction on the back propagation and user guidance algorithms can be problematic if it occurs within a narrow time window. Once the user has selected the optimal path and begins traveling, a failure among the nodes responsible for guidance could prove difficult to recover from.

### 3.4.2 Link Failures

The affect of unpredictable link failures on the system represent a more subtle and difficult issue to analyze. By the nature of wireless sensor networks, communication links will nearly always appear to fade in and out as the RF environment changes over time in the network. Furthermore, temporary problems such as collisions and more permanent issues like hidden nodes present difficulties. Within the detection algorithm, the presence of temporary failures ensures that we strike a balance between purging outdated followers and leaders, and waiting several cycles to determine whether it is merely a temporary communication problem. Repeated threat awareness broadcasts by a leader help to ensure that all nodes within the area affected by the threat remain up to date. Similarly, repeated broadcasts within the back propagation algorithm help to ensure link failures do not prevent the optimal path from being returned to the user.

# Chapter 4

# Implementation and Results

This chapter presents an implementation of the threat avoidance and path planning system on the Cricket v2 hardware using the TinyOS programming environment.

## 4.1 Platform Description

This section offers an overview of the hardware and software platform used to implement the threat avoidance system.

### 4.1.1 Hardware

The Cricket platform provides a foundation for ubiquitous computing and sensor network applications by offering real time location information to mobile users. Designed by members of the Networks and Mobile Systems group at MIT, individual Crickets measure approximately 9.5 cm long by 3.8 cm wide by 3.2 cm high and are powered by 2 AA batteries. They can serve as replacements for traditional motes in sensor network applications. Briefly, each Cricket contains an Atmel ATMega128L microcontroller with 128Kb of program memory and 4Kb of RAM, an RS232 serial interface, a ChipCon CC1000 radio, the ability to send and receive ultrasound signals, three LEDs, and several sensors. These components were chosen to support range measurements between a pair of crickets using a time distance of arrival (TDOA)

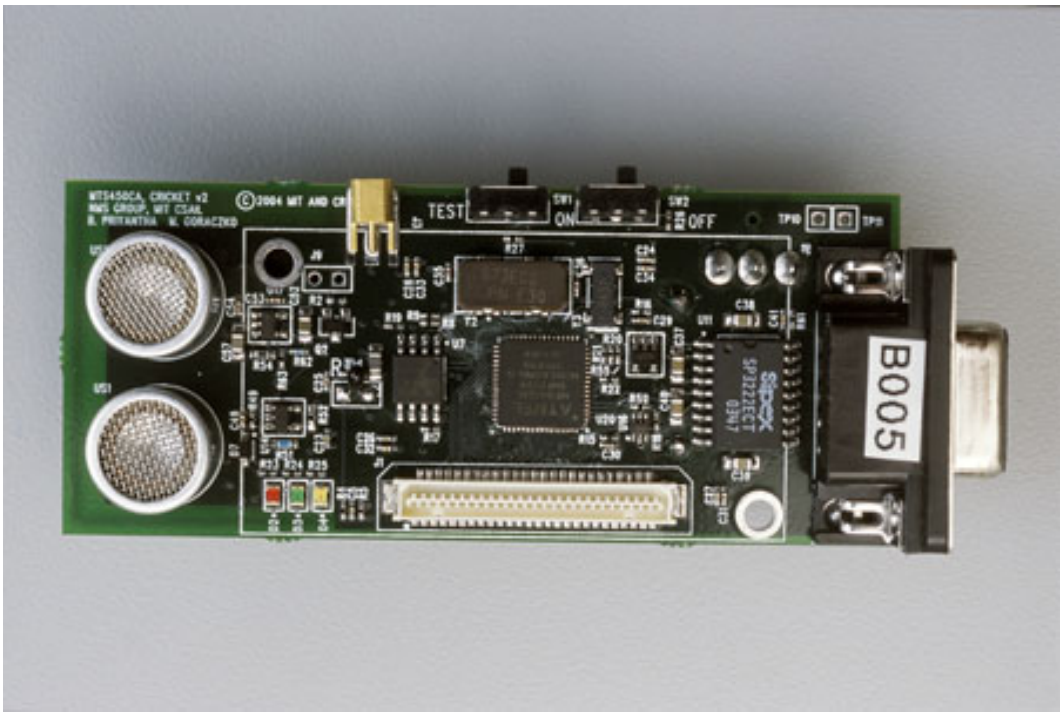


Figure 4-1: An individual Cricket

calculation with the radio and ultrasound. Figure 4-1 includes a photograph of a Cricket.

### 4.1.2 Software and Capabilities

The software included with the Cricket distribution enables each individual Cricket to serve as either a "beacon" or "listener" at any point in time. In a typical application, beacons are programmed with their location and periodically broadcast timed radio and ultrasound signals. By measuring the TDOA between the two messages, listeners can approximate their range to the beacon, typically within 1-3 cm of error. In an environment with several beacons, the Cricket software allows a mobile listener to calculate its position in real time. Typical applications enabled by this technology involve passive mobile agents that can benefit from location information.

Version 2 of the Cricket software distribution is written using TinyOS. TinyOS offers a component based architecture and the ability to assemble complex applications from stand-alone, reusable modules. The interactions between modules are governed

by well-defined interfaces, and an event driven model allows for a single execution thread that can be interrupted by hardware events that are serviced by dedicated handler methods [4]. TinyOS is written in nesC, a language designed specifically to support the component based architecture [5].

The Cricket distribution includes modules for communicating with a computer over the serial port, controlling ultrasound broadcast and reception, and allowing listeners to calculate ranges between themselves and beacons and multilaterate their own position. Using the serial connection and a terminal program running on a personal or mobile computer, users can query a Cricket for its configuration and change variables such as location and listener / beacon status. The *cricketd* application included in the distribution can substitute for a terminal program on the personal computer and serve as a conduit between a custom application and a Cricket.

## 4.2 Implementation

The implementation described in this section makes use of components included in version 2.3.0 of the Cricket software distribution. Serving as a demonstration of the path planning and threat avoidance algorithms detailed in Chapter 2, the implementation is primarily made up of custom nesC code written for the Crickets, but also includes an application written in Java and running on a mobile computer. Identical code runs on all the Crickets, and they are configured at runtime to serve as either a listener, a beacon, or a threat. Nearly all the Crickets are placed in listener mode and make up a sensor network responsible for detecting threats and guiding users through the field. One Cricket is connected to a mobile computer with a serial cable, configured as a beacon, and travels with the user. The application running on the mobile computer allows the user to request paths and provides an interface for navigation. To support the demonstration, an arbitrary number of Crickets are configured as threats and placed in the network to simulate the existence of threats in a real world scenario.

### 4.2.1 Functionality

The listener Crickets performs a variety of functions. For convenience, the application defaults to recognizing the specific "threat" messages sent by the Crickets configured as threats, although detection based on other factors such as temperature is also possible. The group formation algorithm described in Chapter 2 is implemented, and it is combined with the ability of each node to maintain a table of the approximate locations of threats detected by the network. The path request / reply framework and back propagation algorithm is supported, and the application includes a simplistic survival probability model to evaluate the relative safety of paths. Once a path is selected by the user, the nodes along the path can be "activated" and the user guided along the path by the application running on the mobile computer. Details of how these functions are implemented are offered in the Architecture section.

When configured as a beacon to travel with the mobile user, much of the listener functionality is disabled. The beacon emits a timed ultrasound pulse and radio packet approximately every two seconds, allowing each listener to calculate its distance from the beacon and send that distance and its location back in a randomly timed reply packet. Based on recent reports, the application running on the mobile computer periodically runs a multilateration algorithm to determine its own position. Although the beacon does not participate in the threat detection process or forward path requests or replies, it is able to originate path requests, receive the matching reply, and instruct nodes making up the path to begin actively guiding the user.

### 4.2.2 Architecture

The functionality described above is implemented in a collection of TinyOS modules. Each module offers one or more interfaces to its public functions and interacts with lower level components provided by the TinyOS and Cricket software distributions. For instance, the radio and time abstractions are utilized by nearly all the modules. Here is an overview of the roles each of the modules play:

- ThreatDetectionM – supports threat detection and awareness,

- PathPlanningM – implements the path request / reply framework and the back propagation algorithm,
- StorageM – maintains the table of neighbors and associated midpoints, implements the survival probability model, and
- ExternalM – serves as the interface with the desktop application over the serial port.

Details about the implementation of each of these components are offered in the subsequent sections. Appendix A contains a module dependency diagram for the software running on the Crickets, as well more detail about the interfaces each modules provides and uses.

A collection of original message types are defined within the application to support different functions. They are summarized in Table 4.1 and all fit within one packet. In all cases, the application uses the TinyOS Active Message abstraction to broadcast and receive messages over the radio. Within this framework, sending a message is a split phase operation, and the processor can continue to do meaningful work while waiting for a message to be broadcast. Handlers for receiving each type of message are located throughout the modules. In instances where we intend to forward or modify and forward a message (types *TA*, *PRQ*, *PRP*, *PAT*, and *PAC*), we are careful to avoid race conditions where buffers could be accidently overwritten. This is accomplished with a set of dedicated queues that allow us to compare the sequence numbers of received messages to ensure we are not repeatedly forwarding the same message. The queues support randomized hold times to avoid congestion and allow for the same message to be sent an arbitrary number of times when necessary to improve reliability. The mean and standard deviation of these hold times can be modified easily, and by default are set to approximately 0.38 seconds and 0.07 seconds, respectively.

The implementation makes extensive use of the TimerC module offered by TinyOS. TimerC allows for one-time and repeating timers that trigger an event upon expiration. Individual instances of TimerC can be started and stopped depending on the situation, and are used to schedule message sending, deletion of time sensitive data,

| Type | Function | Data Included |
|---|---|---|
| *LR* | Listener reply | Listener location and distance to beacon |
| *TM* | Threat detection | Group leader / follower status and location |
| *TA* | Threat awareness | Threat location, leader ID, and sequence numbers |
| *TTH* | Threat | None (periodically sent by Cricket in threat mode) |
| *PRQ* | Path request | Path origin, destination, and sequence numbers |
| *PRP* | Path reply | Survival probability and IDs of path, ads to neighbors |
| *PAT* | Path activation | IDs of Crickets along path, sequence numbers |
| *PAC* | Path clearing | Sequence numbers |
| *NMM* | Hello message | Cricket location and midpoints (optional) |

Table 4.1: Message types used in the implementation.

and routine calculations. Appropriately scheduled, the timers allow the processor to manage its computational resources so it does not become overloaded.

The performance of the system is influenced by a collection of constants. They are used to define the repeat intervals for timers, the size of data structures, and other important factors. In this implementation, their values were chosen based on the results of small tests to the system. The role and values of some of the most important constants are summarized in Tables 4.2 and 4.3.

| Description | Value |
|---|---|
| Neighbor records | 8 |
| Neighbor pair midpoint-to-midpoint survival probability calculations | 28 |
| Advertisements to neighbors' midpoints in a path reply message | 7 |
| Threats each listener can be aware of | 6 |
| Crickets reporting a threat detection to a single leader | 8 |

Table 4.2: Data structure sizes used in the implementation. Each value represents the maximum number stored at any given time.

| Description | Value |
|---|---|
| Leader declaration countdown | 4 - 5 |
| Frequency to report detected threats to leader | 5 - 6 |
| Frequency leader announces threat position | 8 - 9 |
| Delay before relaying messages | 0.25 - 0.5 |

Table 4.3: Time period constants used in the implementation. All values are in seconds. In every case, time values are selected from a uniform distribution bounded by the two constants.

### 4.2.3 Threat Awareness

Threat awareness is accomplished in four phases: detection, group formation, location estimation, and announcements. When a message of type *TTH* is received, the listener Cricket updates its status. A timer responsible for managing group formation within ThreatDetectionM fires periodically, and based on threat detection status and whether the Cricket has heard a group leader broadcast recently, determines whether to broadcast follower or leader detection (*TM*) messages. These messages continue until the threat is no longer detected, at which point the Cricket broadcasts a follower or leader bailout message. As specified in the group formation algorithm, a randomized leader declaration timer begins once a leader bailout is received.

The group leader is responsible for performing location estimation and initiating announcements. The leader maintains a table of the locations of any followers who have also recently reported the threat. A dedicated timer manages the process of calculating the centroid of all the locations and broadcasting an announcement. Using the message queueing system discussed above, any Crickets within a specific radius of the threat's location forward the announcement and store a record of the threat. A maintenance timer periodically removes stale threat records. The size of detection groups is limited to nine nodes (including the leader), and a Cricket can be aware of a maximum of six threats at any time.

### 4.2.4 Survival Probability and Midpoint Shifting

The algorithms and data structures required to perform survival probability calculations are located in StorageM. When a Cricket receives a message of any type, it updates a table with a time stamp and the one byte identification number and location (if present in the message) of the neighbor who sent it. This neighbor table supports up to eight records, and records with old time stamps are removed periodically. Each Cricket broadcasts messages of type *NMM* approximately every five seconds to ensure its neighbors are aware of its existence and location.

Using the table of threats, a method periodically calculates the survival probability

associated with traveling between the midpoints of each pair of its neighbors. The calculation for a pair of midpoints involves several steps. First, a value for survival probability at each midpoint is calculated using a step function that approximates an exponential. The function used in the implementation was chosen for simplicity and is shown in Figure 4-2. The survival probabilities at the two endpoints are averaged and the result is modified to reflect the ratio of the distance between the points to a reference distance of 10 feet. A method for calculating the survival probability of a path created by combining two smaller paths is included. It works by multiplying the probabilities associated with the constituent paths. In order to support rapid path replies, a table containing the survival probabilities between each pair of midpoints is maintained by a method called by a timer. This table holds a maximum of 28 entries, equivalent to one for each pair of at most eight neighbors.

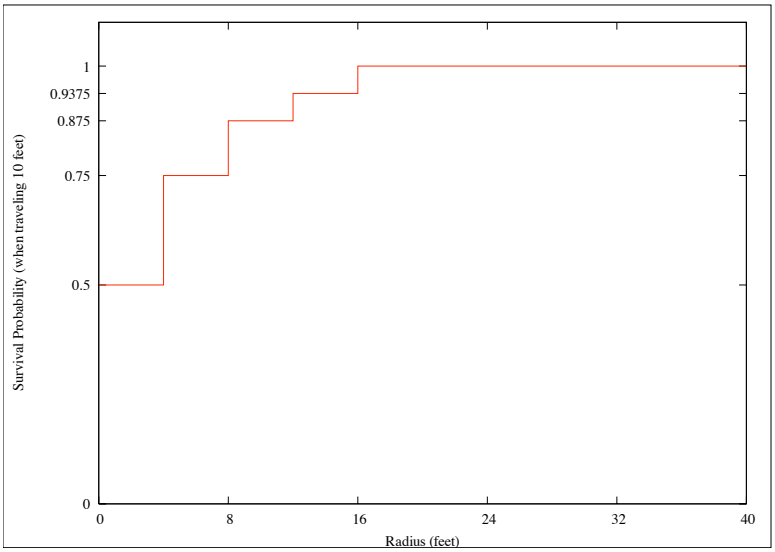


Figure 4-2: The survival probability function used in the implementation to approximate an exponential.

When a Cricket is aware of fewer than two threats in the network, it chooses the geometric midpoint of its location and each neighbor's location as the midpoint for that neighbor. When two or more threats are detected, however, each Cricket attempts to shift its midpoints to lie along the perpendicular bisector of the line connecting the threat locations. These shifts only occur if the distance between the

shifted location and the geometric midpoint is less than one half of the distance to the neighbor. By imposing this limit, we avoid choosing unrealistic midpoints and only shift in situations where the midpoint is likely to bisect two threats and form a path. Examples of the effects of this algorithm are offered later.

### 4.2.5 Path Planning Algorithms

The PathPlanningM module contains the path search and construction functionality, along with associated data structures and event handlers. Using messaging queues, *PRQ*, *PAT*, and *PAC* messages are forwarded to reach the entire network after they are initiated by the user. If a Cricket is within a specific distance of the destination in a *PRQ* message, it initiates a reply by broadcasting a *PRP* message with its own identification number in the path, a 100% survival probability and an array of advertisements to the midpoints of each of its neighbors. Each advertisement contains the one byte identification number of the neighbor and the survival probability from the destination to the midpoint.

The intelligence of the back propagation algorithm is contained largely in the modification and forwarding of *PRP* messages. Each Cricket maintains the optimal (highest survival probability, or in event of a tie, smallest hop count) *PRP* message it has forwarded in response to any path request. When a *PRP* message is received, the Cricket first calculates the survival probability of the path to its midpoint using the total accumulated survival probability of the path so far and the survival probability to its midpoint contained in the advertisement. If there is no advertisement to the receiving Cricket, the message is ignored. If, however, the reply is optimal, a set of advertisements is built using records from the midpoint survival probability table discussed in the previous section. As in the initial reply message, each advertisement contains the one byte identification number of the neighbor and the survival probability to extend the path to that neighbor. If a survival probability record is not found in the table, the calculation is done immediately. Finally, the Cricket appends its own identification number to the path and inserts the message into a queue for broadcast. To improve reliability, each Cricket broadcasts its optimal reply twice.

The beacon Cricket traveling with the user maintains a record of its most recent path request, and compares it with replies to ensure the optimal reply is stored. A custom string is printed over the serial port whenever a new reply is received to inform the application running on the mobile computer. At any point after the first reply is received, the user may choose to initiate a *PAT* message and "activate" the series of listener Crickets making up the reply. The significance of activation and its role in user guidance is discussed in the next section. Once path traversal is complete, a *PAC* message is initiated by the user and sent by the beacon to clear all activated Crickets.

### 4.2.6 User Guidance

User guidance is accomplished through a combination of the mobile application and the *LR* messages sent from the listeners to the beacon. The ExternalM module converts received *LR* messages on the beacon into a formatted string that is sent to the application over the serial port. The application includes a window displaying a scaled representation of the listener Cricket locations gathered from *LR* messages. Using a collection of the most recent of these messages, the application periodically multilaterates and displays the position of the beacon (and thus the user). When a set of listener Crickets involved in constructing a path are activated, they begin including the coordinates of their two midpoints in their *LR* messages. As the user travels from midpoint to midpoint and begins receiving *LR* messages from the next Cricket, the application displays the next midpoint along the path. A screenshot of the application is shown in Figure 4-3.

## 4.3 Performance Evaluation

Several tests of Cricket implementation were conducted in order to evaluate its performance. This section presents results from those tests and observations about the strengths and limitations of the implementation.

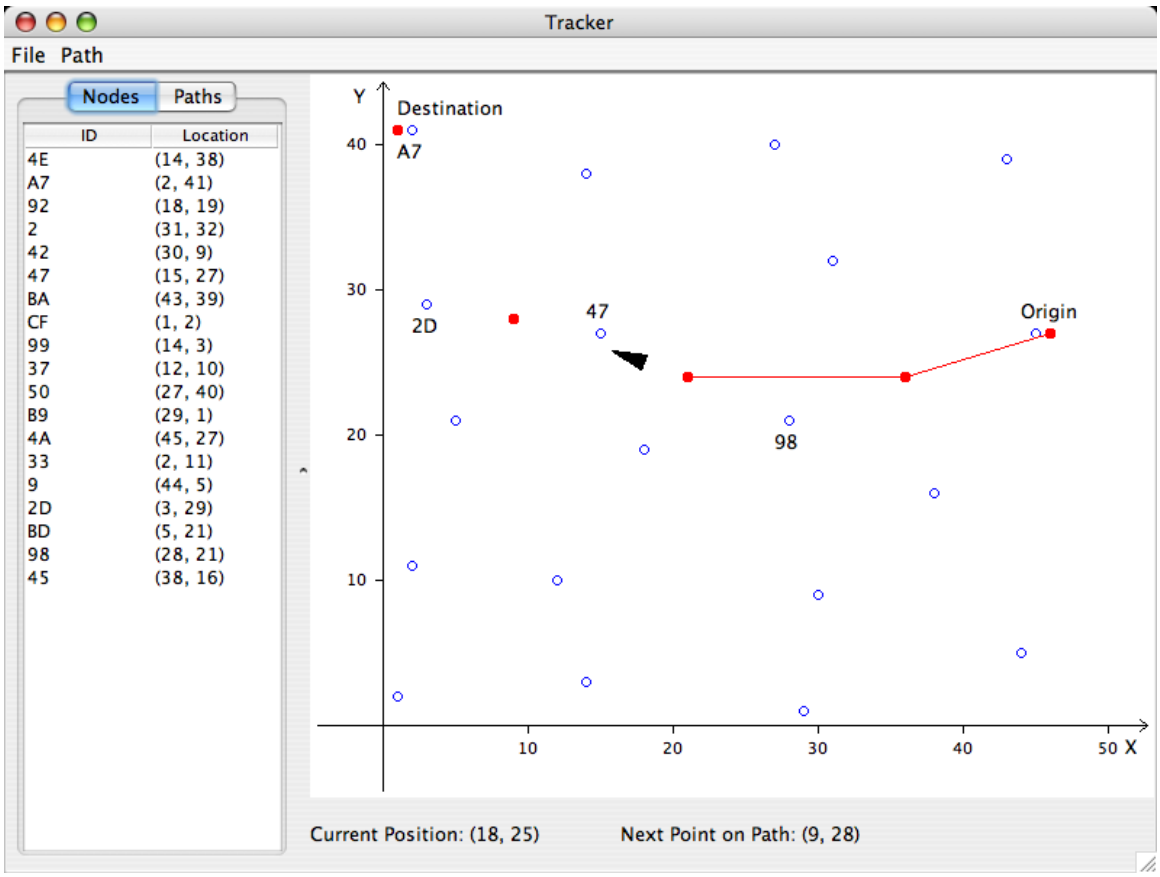


Figure 4-3: A screenshot of the mobile application in use. The user has requested a path from (46, 27) to (1, 41). The identification numbers of the Crickets along the path, the user's current location, the path traversed so far, and the next midpoint are labeled.

### 4.3.1 Test Setup

The tests were conducted using nineteen listener Crickets placed in random locations on a gymnasium floor within a rectangle of 44 feet by 39 feet. Antennas were not added to any of the Crickets to ensure that RF signal propagation would be limited. The result was the creation of a multihop network with a diameter of approximately 4-5 nodes. The listener Crickets were assigned locations using a serial connection to a mobile computer before they were placed, and had ample time to populate their neighbor table and midpoint locations. The spatial density of the Crickets was 0.0111 per square foot. Twelve paths were requested using a Cricket configured in beacon mode using its location as the origin of the path and a randomly selected location

near a listener Cricket as the destination.

### 4.3.2 Results

The system proved to be very reliable, responding with a path after every request. With only one source of path requests in the network, congestion was not an issue. The latency between a request and the receipt of an optimal reply ranged from 1.37 seconds to 2.94 seconds, with an average over all the requests of 1.90 seconds and a standard deviation of 0.52 seconds. The distance between the origin and destination locations of the paths ranged from 29.5 feet and 55.3 feet, with an average of 42.3 feet and a standard deviation of 7.7 feet. The average hop count of the optimal paths was 5.0. The correlation between latency and distance between the origin and destination was 0.719.

Threats were represented by Crickets configured in "threat" mode and emitting periodic *TTH* messages. For a threat placed at (28, 32) within the coordinate system of the network, the discrepancy between of the centroid of the nodes detecting the threat and the actual location was 2 feet. Seven listener Crickets reported a detection, and ranged in distance to the threat from 3 feet to 17.7 feet. The discrepancy between calculated and actual locations for a threat placed at (9, 9) was 2.24 feet. In this case, six listener Crickets detected the threat, and ranged in distance from 3.2 feet to 13.5 feet.

The difference between paths found by the back propagation algorithm when one threat was present and when none were present was significant. Figure 4-4 displays an example of the response to a request from (46, 27) to (1, 41) with no threats present. Each midpoint along the path is located exactly half way between the the locations of the two nodes surrounding it. The path arrived 2.43 seconds after the request, a period reflective of the 0.25 - 0.5 second hold times within the message queues for *PRQ* and *PRP* messages and a hop count of 5 nodes.

With one threat located at (28, 32), an identical request resulted in the path shown in Figure 4-5. The latency in this case was 2.24 seconds, a negligible difference given the variance of the message queue hold times. As before, each midpoint is

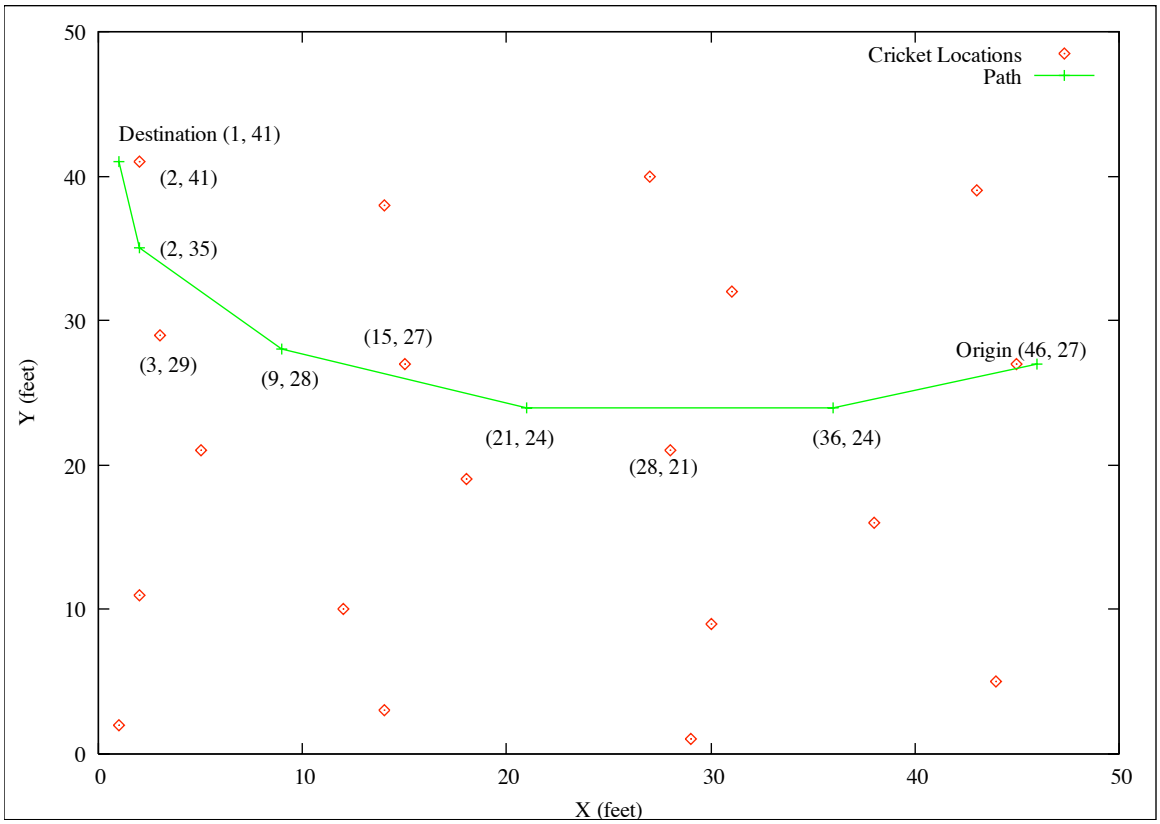


Figure 4-4: A path formed from (46, 27) to (1, 41) in a network with no threats present. The coordinates of the midpoints and the listener Crickets along the path are labeled.

located exactly halfway between the locations of the two Crickets surrounding it.

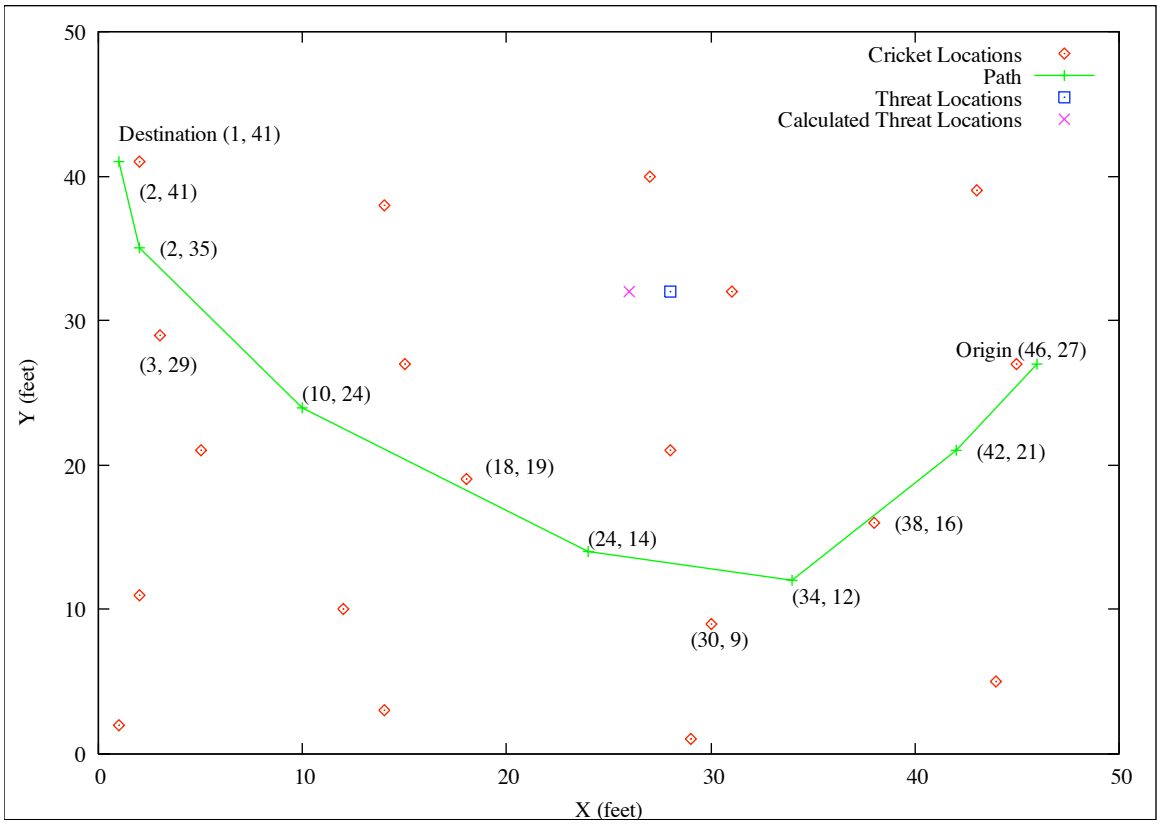


Figure 4-5: A path formed from (46, 27) to (1, 41) with one threat present in the network. The coordinates of the midpoints and listener Crickets along the path are labeled.

The presence of two threats in the network allows for an evaluation of the benefits of midpoint shifting. With an additional threat placed at (9, 9), Crickets located near the perpendicular bisector of the line connecting the threat locations shifted their midpoints. This phenomenon is illustrated by the path shown in Figure 4-6. The time required to discover the path was 2.94 seconds.

In this case, midpoint shifting generated an improvement when compared to a path constructed from unshifted midpoint locations. In order to determine the magnitude of the improvement, we evaluate the survival probability of each path using the threat model presented in Chapter 2. We use a survival probability function of

$$s = 1 - e^{-3.84 \cdot r} \tag{4.1}$$

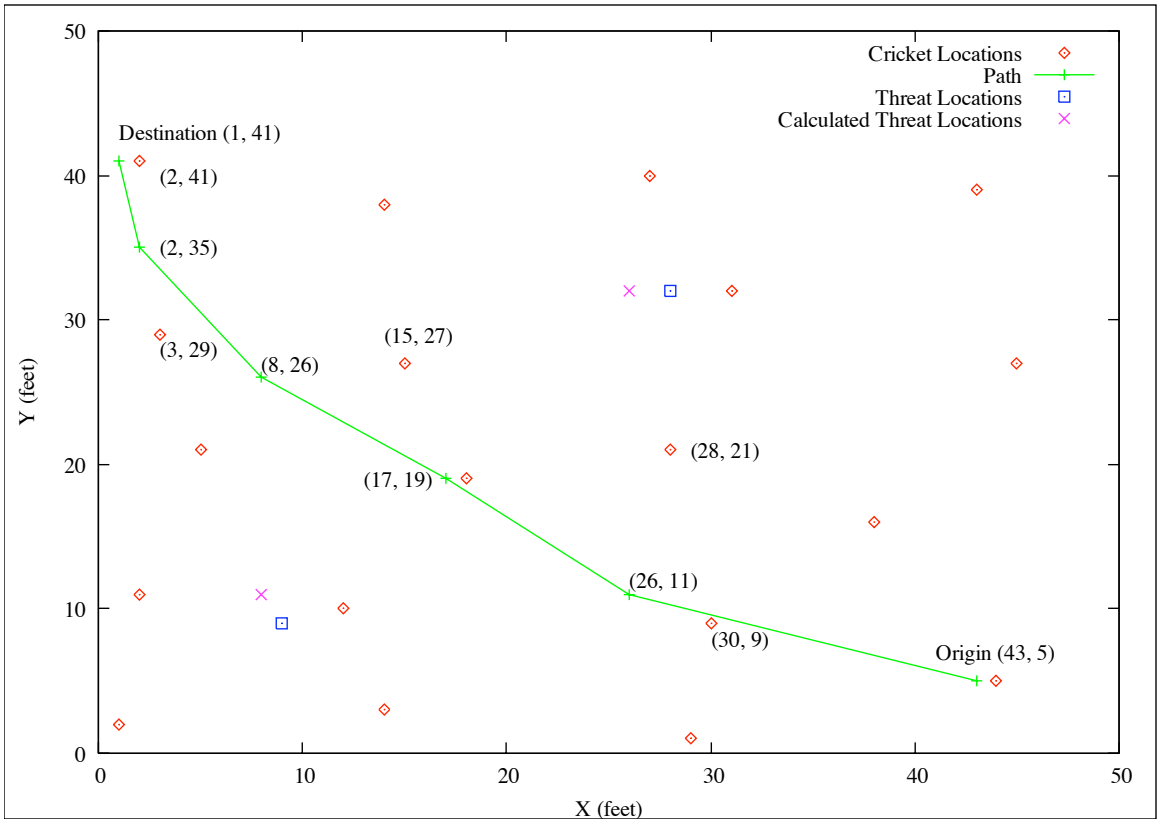


Figure 4-6: A path formed from (43, 5) to (1, 41) with two threats present in the network. The actual and calculated threat locations are shown. The coordinates of the midpoints and listener Crickets along the path are labeled. The midpoint located at (17, 19) was shifted from its default location of (21.5, 24), the midpoint located at (26, 11) was shifted from its default location of (29, 15), and the midpoint located at (8, 26) was shifted from its default location of (9, 28).

and one foot as the unit distance in order to approximate the threat model used in the implementation. The survival probability of the entire path is determined by multiplying the result of Equation 4.1 at one foot intervals along the path for each threat. The result for the path containing shifted midpoints is 35.0%, while the result for the path containing unshifted midpoints is 29.1%. In this instance, the midpoint shifting algorithm offers an improvement of $\frac{35.0-29.1}{29.1} = 20.2\%$. The path containing unshifted midpoints is shown in Figure 4-7.

Given our the locations of the two threats and our threat model, we can also determine the ideal path through the continuous space covered by the network. The ideal path maintains a course along the perpendicular bisector of the line connecting the

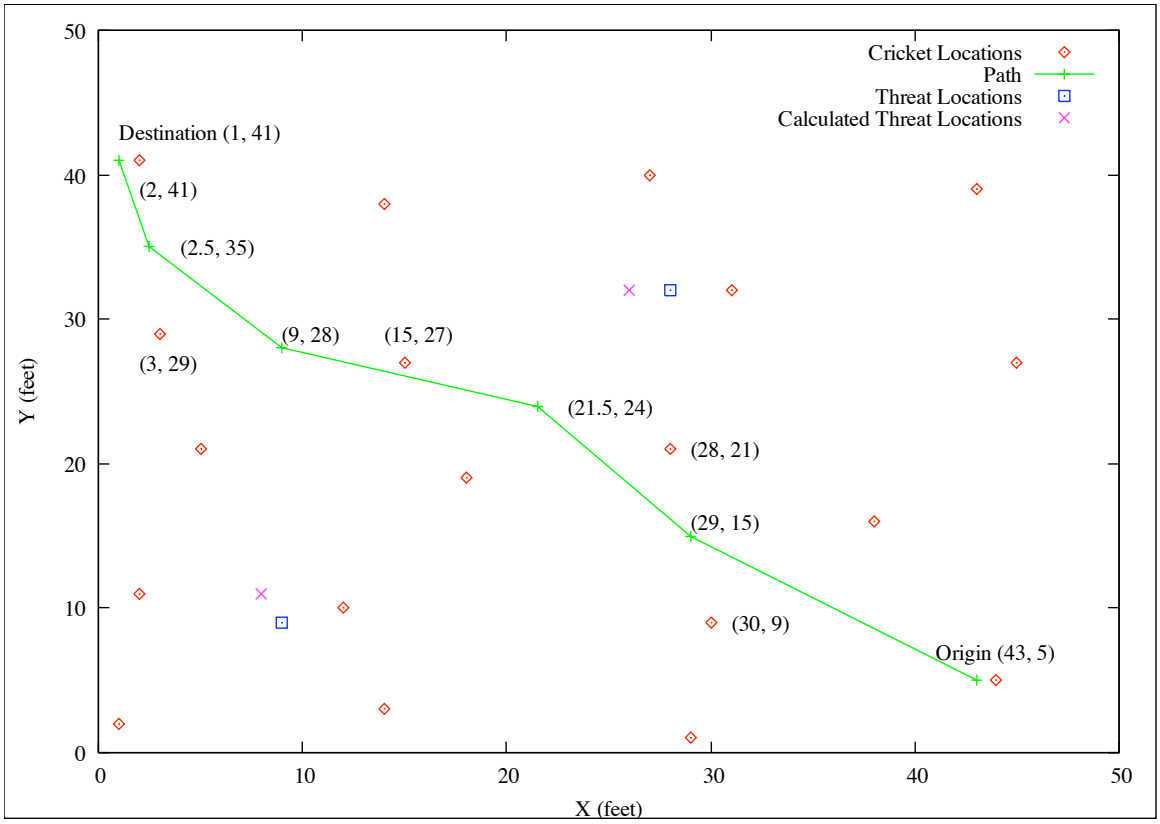


Figure 4-7: A hypothetical path formed from (43, 5) to (1, 41) with two threats present in the network and no midpoint shifting. The actual and calculated threat locations are shown. The coordinates of the midpoints and listener Crickets along the path are labeled.

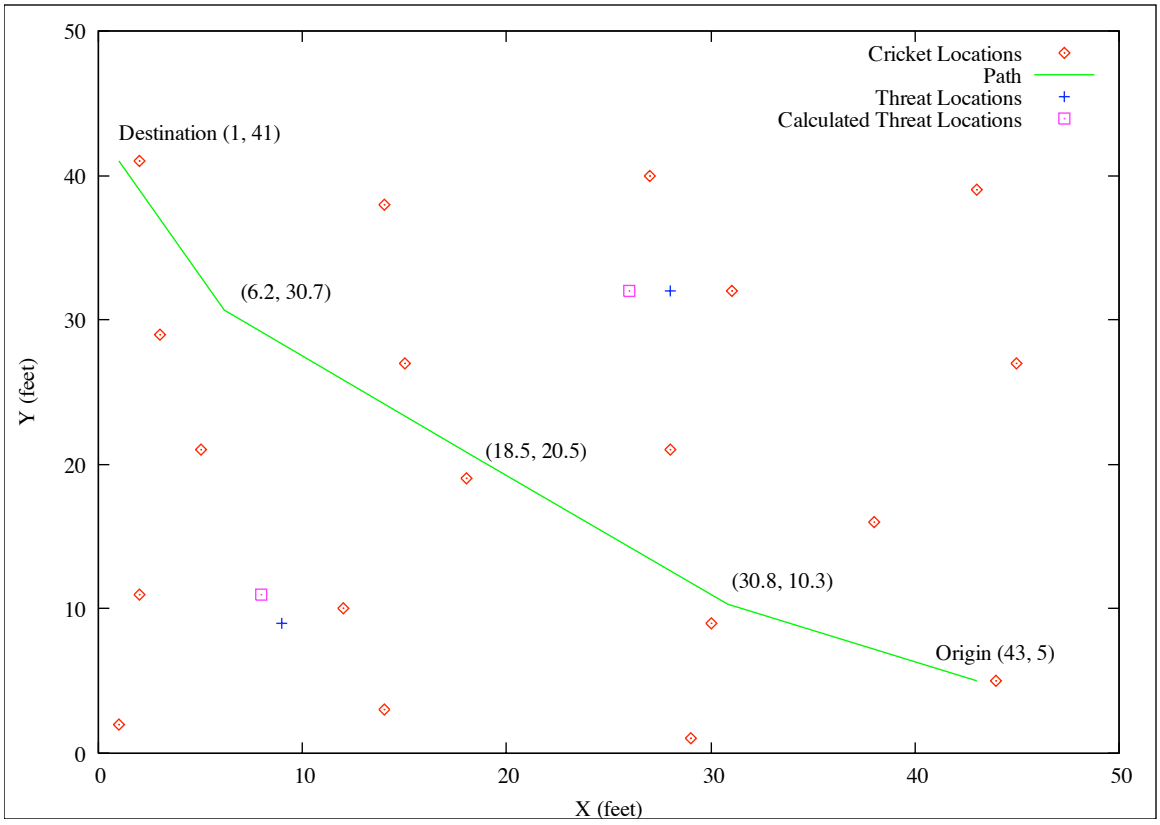


Figure 4-8: The optimal path from (43, 5) to (1, 41) with two threats present in the network. The actual and calculated threat locations are shown and the points along the path are labeled.

locations of the threats and is shown in Figure 4-8. Using the same process as before, this path has a survival probability of 42.4%, representing a 20.7% improvement on the path constructed from shifted midpoints.

### 4.3.3 Observations

The results presented above demonstrate the effectiveness of the implementation in a controlled environment. Although the survival probability model used is relatively simplistic, it allows the application to accurately compare the relative fitness of paths and find the optimal one. We could imagine using survival probability function that better approximates the exponential or is tuned for a different relationship between users and threats, but the mechanics of the system remain the same. Similarly, the

hold times of the message queues and the decision to broadcast some messages twice improve reliability, but the specific constants may be changed in a different scenario. The threat detection algorithms performed well in the tests, although more extensive testing would be required to assess their accuracy when threats cover a region, rather than a point.

Based on the specifications of the implementation and the tests that were conducted, we can estimate several system properties. In both communication bandwidth utilized and computation, the system can scale far larger than nineteen listener Crickets. The frequency of certain timers play a large role in scalability. For instance, if we were recomputing midpoint locations and survival probability records between midpoints more frequently, we would not have as much processing time left for other tasks. Although each listener Cricket currently supports replying to three simultaneous path requests, this limit is arbitrary and could be raised. Practically, however, the communication traffic generated with too many requests would force request response time to decrease. The average response time of 1.9 seconds observed in the tests could be lowered by decreasing the message queue hold time, but at the expense of lower throughput and a greater possibility that ideal paths would not reach the requesting node.

## 4.4 Conclusion and Future Work

The implementation presented in this chapter demonstrates the practicality of the design offered in Chapter 2. Although the survival probability model was not ideal, and approximations were made in the code to improve efficiency, the system performed reliably in tests. Ideal parameters for rebroadcasting and timing constants were not able to be determined, but those used in the implementation yielded accurate results with acceptable latency. In addition, the degree to which varied RF environments impact the performance of the system is not known, but parameters could be modified, perhaps at runtime, to accommodate different environments.

Larger tests and more analysis is needed to determine the impact of midpoint

shifting and the back propagation algorithm in large scale networks or in situations where the node density varies widely. In very large networks with high density, it may make sense to explore the possibility of avoiding midpoint shifting and instead filtering the set of nodes that are involved in back propagation. The decision to adopt this strategy would be based on a desire to minimize contention for the broadcast medium and scale more effectively in large networks. Although distinct from using shifted midpoints to cover the search space, this tactic would address the same goal: balancing efficiency with accuracy when constructing paths.

It is also possible that greater awareness of link quality on each node could help to improve the performance of the application. Midpoints could be selected exclusively from neighbors where connectivity surpassed a certain threshold. This step would increase the odds of matching path advertisements with recipients and thus improve the likelihood of optimal paths reaching the user. Of the issues discussed in the section, the specific challenges of real world situations will determine which receives the most attention.

# Appendix A

# Implementation Architecture

The modules described in Chapter 4 interact with each other through a collection of interfaces. Within TinyOS, each module can provide any number of interfaces and use the interfaces of any other module. Table A.1 describes the interfaces provided by StorageM, Table A.2 describes the interfaces provided by ThreatDetectionM, and Table A.3 describes the interfaces provided by PathPlanningM. Figure A-1 contains a dependency diagram for the modules and interfaces within the application.

| **Name** | **Functionality** |
|---|---|
| StorageMessage | Printing data containing neighbors, threats, and survival probability between midpoints |
| NeighborManagement | Inserting and querying neighbor information, including midpoint locations |
| ThreatManagement | Inserting and querying threat information; determining survival probability of paths |
| ActivationManagement | Activating and deactivating paths; querying midpoint locations to enable user guidance |

Table A.1: Interfaces provided by StorageM

| **Name** | **Functionality** |
|---|---|
| ThreatMessage | Printing data containing threat detection information |

Table A.2: The interface provided by ThreatDetectionM

| Name | Functionality |
|---|---|
| PathMessage | Printing records of received and forwarded path request, reply, and activation messages |
| PathOperations | Requesting, activating, and clearing activated paths |

Table A.3: Interfaces provided by PathPlanningM

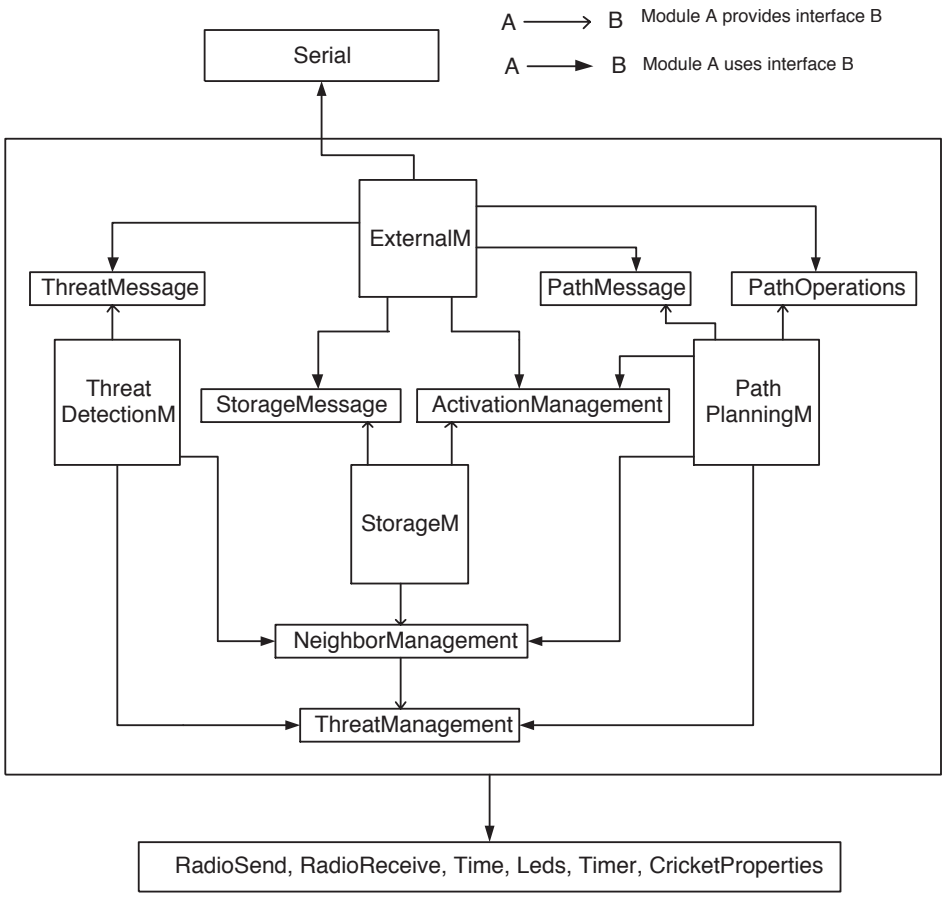


Figure A-1: A module dependency diagram for the implementation.

# Bibliography


[1] Maxim A. Batalin and Guarav S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunications Journal, Special Issue on Wireless Sensor Networks*, 2004.

[2] Maxim A. Batalin, Guarav S. Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. *IEEE International Conference on Robotics and Automation*, pages 636–642, April 2003.

[3] Peter Corke, Ron Peterson, and Daniela Rus. Networked robots: Flying robot navigation using a sensor net. *Proc. Int. Symp. Robotics Research*, November 2003.

[4] David E. Culler, Jason Hill, Philip Buonadonna, Rober Szewczyk, and Alec Woo. A network-centric approach to embedded software for tiny devices. *EMSOFT*, October 2001.

[5] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. *Proceedings of Programming Language Design and Implementation*, June 2003.

[6] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, Massachusetts, 1991.

[7] Qun Li and Daniela Rus. Distributed algorithms for guiding navigation across a sensor network. *ACM Mobicom*, pages 313–325, September 2003.

[8] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. *First ACM Workshop on Wireless Sensor Networks and Applications*, September 2002.

[9] Radhika Nagpal and Daniel Coore. An algorithm for group formation in an amorphous computer. *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems*, October 1998.

[10] Richard T. Vaughan, Kasper Stoy, Guarav S. Sukhatme, and Maja J. Mataric. Lost: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18:796–812, 2002.