SWARM INTERPOLATION WITH MOBILE SENSORS

by

Joshua Kirby

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Summer 2017

© 2017 Joshua Kirby
All Rights Reserved
SWARM INTERPOLATION WITH MOBILE SENSORS

by

Joshua Kirby

Approved: 

Kathleen F. McCoy, Ph.D.
Chair of the Department of Computer Science

Approved: 

Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: 

Ann L. Ardis, Ph.D.
Senior Vice Provost for Graduate and Professional Education
I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed:  
Chien-Chung Shen, Ph.D.  
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed:  
Louis F. Rossi, Ph.D.  
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed:  
Adarsh Sethi, Ph.D.  
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed:  
Keith Decker, Ph.D.  
Member of dissertation committee
# TABLE OF CONTENTS

| LIST OF TABLES | vi |
| LIST OF FIGURES | vii |
| ABSTRACT | xii |

## Chapter

1 INTRODUCTION ................................................. 1
   1.1 General Statement of Problem ......................... 1
   1.2 Swarming ............................................. 3

2 SWARMING FOR UNIFORM COVERAGE OF A REGION .. 8
   2.1 Coverage Through Multiple State Swarming ............ 8
      2.1.1 Assumptions ..................................... 10
      2.1.2 Algorithm overview ................................ 11
      2.1.3 Scaling ........................................... 12
      2.1.4 Uniform alignment in the boundary ................. 16
      2.1.5 Uniform alignment in the interior ................. 23
      2.1.6 Uniform boundary and interior spacing .......... 28
      2.1.7 Theoretical analysis of node interactions ....... 33
   2.2 Experiments ........................................... 38
      2.2.1 Gaussian field .................................... 38
      2.2.2 Interaction decay rate ($k_I$ and $k_B$) ........ 42
      2.2.3 Interior node strength and boundary seeking velocity ($A_I$ and $v_B$) . 45
      2.2.4 Maximum interaction range ....................... 47
      2.2.5 Node reassignment tolerance .................... 47
      2.2.6 Adaptation to region size ........................ 51
         2.2.6.1 A larger region ............................ 51
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Algorithm Parameters</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Node State Parameters</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Index of procedures for algorithm</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Default Experiment Parameters</td>
<td>38</td>
</tr>
<tr>
<td>3.1</td>
<td>Default Experiment Parameters</td>
<td>81</td>
</tr>
<tr>
<td>3.2</td>
<td>Error Ratios for Circular Regions</td>
<td>88</td>
</tr>
<tr>
<td>3.3</td>
<td>Error Ratios for Square Regions</td>
<td>92</td>
</tr>
<tr>
<td>3.4</td>
<td>Error Ratios for Multiple Peak Regions</td>
<td>95</td>
</tr>
<tr>
<td>3.5</td>
<td>Error Ratios for Crescent Regions</td>
<td>98</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>Flowchart showing the overall structure of the algorithm controlling the behavior of an individual sensor node.</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Diagram showing the spacing and interaction range of an interior node, located in the center of the diagram. The large inner circle indicates the local spacing $s_n$ for the node, based on the mean of the distances to the three nearest interior neighbors. The large outer circle indicates the maximum interaction range, with $R_s = 3$. The smaller circles represent nearby nodes, with nodes outside of the interaction neighborhood indicated by dotted borders.</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Diagram showing the classification of nodes near the boundary of the swarm. The black circles represent boundary nodes, while the white circles represent interior nodes. $N$ and $T$ are the normal and tangent vectors of the boundary, while $\beta$ is the argument of the normal vector relative to standard coordinates.</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Diagram showing interactions between boundary nodes. The dashed line represents the level set of a region, with the area above outside the region and the area below inside. Boundary nodes move perpendicular to the edge of the swarm to seek the region boundary, and feel repulsive interactions from neighboring boundary nodes parallel to the edge of the swarm.</td>
<td>24</td>
</tr>
<tr>
<td>2.5</td>
<td>Diagram showing the interaction between an interior node, on the bottom, and a pair of boundary nodes on the top. The white circle represents an interior node, the black circles represent a pair of adjacent boundary nodes, while the dotted circles represent additional positions on a quadrature between the boundary nodes. The arrows represent the magnitude and direction of the interactions felt by the interior node.</td>
<td>26</td>
</tr>
</tbody>
</table>
**2.6** Diagram showing interactions between an interior node, located at the center, and its neighbors. Solid circles represent boundary neighbors, hollow circles represent interior neighbors, and dotted circles represent nodes outside of the interaction range. The central node is directly repelled by interior neighbors, while being repelled by a piecewise linear interpolation of the boundary neighbors. 

**2.7** Diagram showing the placements of nodes in the model configuration. The bottom side contains a set of interior nodes placed in a triangular grid of width $s_I$, shown as white circles, centered on a single boundary-facing interior node, shown as a gray circle. The grid extends to a distance of $s_IR_s$ from the central node. The top side contains a continuous arrangement of nodes representing the boundary nodes, shown as black circles, with the density of the field equal to $1/s_B$. There is a gap between the boundary and the beginning of the interior of width $\Delta$. Gap nodes such as the one indicated in the center of the diagram are adjacent to the gap. 

**2.8** Contour plot of edge node velocities toward the interior when $\rho$ is determined by (2.24). In this case, $s_I = \Delta$ is a stable equilibrium by design. 

**2.9** Node positions at various points in time during the simulation on a circular region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line. 

**2.10** Mean node speeds during the circular simulation. 

**2.11** Mean node spacings during the circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it. 

**2.12** Mean node speeds based on modifying $k_I = k_B$. 

**2.13** Mean node speeds based on modifying $A_I$ while holding $v_B$ constant. 

**2.14** Mean node speeds based on modifying $r_{\text{max}}$. 

**2.15** Ratio of actual boundary spacing to interior spacing (left) and ratio of the number of boundary nodes to interior nodes as a function of $s_{\text{max}}$. The ratio for an ideal solution is indicated by the horizontal line.
2.16 Mean node speeds and spacings during the larger circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.  

2.17 Mean node speeds and spacings during the smaller circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.  

2.18 Node positions at the end of the square simulation.  

2.19 Mean node speeds and spacings during the square simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.  

2.20 Node positions at the end of the crescent simulation.  

2.21 Mean node speeds and spacings during the crescent simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.  

2.22 Node positions at various points in time during the simulation on a circular region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.  

2.23 Mean node speeds and spacings during the translating circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.  

2.24 Node positions at various points in time during the simulation on a deforming region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.  

2.25 Mean node speeds and spacings during the deforming shape simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.
2.26 Node positions at various points in time during the simulation on the baseline scaling region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line. ................................................................. 66

2.27 Mean speeds for the three dimensional rescaling runs, scaled based on $L$ and $\tau$. ................................................................. 67

3.1 Chebyshev distribution: 1D vs. 2D ................................. 70

3.2 Paths formed from each node towards the edge of the swarm and towards the interior of the swarm ................................. 79

3.3 Distributions for nodes within a circular region. The x axis groups nodes based on their distance from the midpoint of the region, while the y axis specifies the percentage of nodes that fall within each group. ................................................................. 85

3.4 Effect of perturbation with maximum wave number $k$ on a circular Gaussian field. ................................................................. 87

3.5 Formations generated by running algorithm on an unperturbed circular region. ................................................................. 89

3.6 Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4 ................................. 90

3.7 Formations generated by running algorithm on a unperturbed square region. ................................................................. 93

3.8 Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4 ................................. 94

3.9 Formations generated by running algorithm on an unperturbed multiple peak region. ................................................................. 96

3.10 Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4 ................................. 97

3.11 Formations generated by running algorithm on a unperturbed crescent-shaped region. ................................................................. 99
Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4
ABSTRACT

Sensors can be used in a variety of monitoring applications, such as measuring the state of large-scale, spatially complex phenomena like fires or oil spills. In applications like these, the primary goal is to collect information about the region of interest simultaneously across the region, assuming a sufficient number of sensors are available. However, it can be difficult and even dangerous to have humans manually place sensors and collect readings. Groups of autonomous mobile sensors under the control of an appropriate algorithm can address this issue by being able to move into position and gather data without human intervention.

In order to gain an accurate view of a region as a whole, it is important that the sensors be distributed to provide effective coverage. For fixed regions of known shape, the ideal positions for sensors can be computed in advance, with sensors simply moving to assign positions. However, if the shape of the region is not known in advance, or if the region changes its shape over time, then ideal positions will not be initially available, and must be identified by the autonomous sensors after they have already been placed and are in motion.

One means by which this can be accomplished is through a technique called swarming, in which individual elements coordinate their own behaviors based on local information in order to achieve a cohesive behavior for the entire group. If an appropriate swarming algorithm is used, mobile sensors can be made to seek out and settle in positions that are ideal for gathering data that will be useful for approximating the region as a whole.

In this dissertation, I first present an algorithm designed to achieve uniform mobile sensor coverage of an unknown region of interest using only local information. The algorithm is capable of scaling to different region sizes and shapes, dynamically
adapt the behavior of the sensor nodes based on locally observed conditions. The algorithm divides nodes into two groups, boundary and interior, selected based on their positions within the swarm. Boundary nodes seek out the edge of the region of interest and distribute themselves evenly along it, while interior nodes use repulsive interactions to uniformly distribute themselves within the region. Uniformity between the boundary and interior is achieved through exchanging nodes between the two groups. Through simulation, I demonstrate the ability of the algorithm to adapt to a variety of regions, and analyze the effect that a number of control parameters have on the behavior of the swarm.

While a uniform distribution is sufficient to achieve complete coverage of a region, it may not be the optimal distribution for gathering data for interpolation. As an expansion of my work on generating uniform distributions, I present a system to generate non-uniform distributions using the concept of virtual coordinates, and show how changing the apparent positions of nodes can coerce a swarm to behave in a different manner without altering the underlying algorithm controlling them. I develop a technique to allow nodes to determine their relative location within a swarm based only on local communications by counting the number of local communication hops to the edge of the swarm, and to use that information to remap their radial distances from the center of the swarm. I present a remapping that will convert a 2D radial Chebyshev distribution into a uniform distribution in virtual coordinates. I adapt the uniform swarming algorithm to use this system of virtual coordinates, and demonstrate that the modified algorithm generates 2D Chebyshev distributions in real space. I simulate the exploration and measurement of regions of interest of a variety of shapes, and analyze the ability of both the uniform and Chebyshev algorithms to generate accurate maps of those regions through radial basis interpolation.
Chapter 1
INTRODUCTION

1.1 General Statement of Problem

When carrying out the task of monitoring a field covering a large environment, such as the Gulf oil spill, it is desirable to be able to gather information across multiple points in the affected area simultaneously, without needing to manually carry out all of the measurements. This can be accomplished through the use of independent sensors, which are placed throughout the region and automatically gather information that is then either transmitted to a central location for analysis or stored until the sensor is physically accessed. New data points can be constructed within the range of this discrete set of known data points by using the technique of interpolation.

The task of utilizing a swarm of sensors to gather information about a region of interest requires that the sensors be positioned in such a way that their measurements will be effective in order to approximate the sensed field. It is known that for sensors with a communication range much greater than their sensing range, the optimal distribution to maintain full coverage is a triangular grid [26]. For sensors with a fixed sensing radius, placing them in a triangular grid $\sqrt{\pi}$ times that sensing radius apart will minimize overlap (and thus maximize efficiency) while still ensuring coverage. If the sensing radius is instead controllable, then building a triangular grid within a region and then calibrating the sensors to that grid will minimize the power needed to ensure coverage. Even for point sensors, a triangular grid will minimize the maximum distances from any point to its nearest sensor. Also if the shape of the region is known in advance, then techniques like the ones used by Cortes [3], which divides a region into Voronoi partitions and attempts to equalize their areas, and Krause [13], which directly places sensors based on mutual information, can be used.
For smaller or simpler environments, such methods can be used to place sensors directly onto their targets, with them remaining there with no further human interaction until their measurements are complete. However, as environments become more complex and dynamic, this may cease to be practical.

For example, as regions grow in size, it becomes less sensible to physically travel to the target positions of each of the sensors. The environments being examined may be too dangerous for humans to enter, such as for a wildfire or a toxic spill. Conditions such as water currents or wind may move the sensors out of their initial positions. In addition, the regions themselves may evolve over time, rendering the initial placements of the sensors less useful.

The solution to these issues is to make the sensors themselves be mobile, capable of understanding their position and moving themselves to reach a target or adjust their settled position as needed. An example of such is described in Leonard [14]. By allowing sensors to move themselves, issues of distance or danger become less relevant, since humans will not need to chaperon them to their positions. The sensors will also be able to correct their positions as needed, whether to counteract drift or to adjust for changing conditions in the region. Deployment and recovery is also simplified, since sensors can be released from a single location, or be instructed to return to one once their measurements have been completed. This technique can also be used to increase the amount of information available to the sensor network by having sensors patrol the region in formations, as was done in [14], though this comes at the cost of the increased energy usage required to keep sensors in constant motion.

While it is relatively trivial to order a group of mobile sensors to move to predetermined positions and remain in them, or to follow predetermined paths, the problem becomes much more complex if less information is known in advance about the size or shape of the region of interest. In those cases, the sensors will need to determine their desired positions during their deployment, adjusting their behavior based on the conditions they and their neighbors observe.

One way to accomplish this is to aggregate all of the information observed by
the sensors into a single location, which then develops an individual motion plan for each of the sensors and transmits it back to them. This has the advantage of allowing for planning to be based on the state of the group as a whole, but comes with many disadvantages. It can require substantial communication overhead, especially if the method of communication cannot cover the entire group and requires that messages be passed through multiple nodes. If the time needed for communication and computation is significant, then the sensors will be slow to react to changes in their environment, and by the time that their new instructions have been received they may have been rendered irrelevant by further changes. It also provides a single point of failure, since the planner breaking would render the entire group uncontrollable.

1.2 Swarming

An alternative is to use a technique called swarming, in which a node decides its own actions based entirely on the information available to it locally. While each node will have a limited understanding of the overall environment and goals for the group as a whole, by coordinating with the actions of their neighbors, they can act as a coherent group in order to accomplish a task. An example of such coordination can be found in the natural world in schools of fish or flocks of birds, which travel and avoid danger as a group despite having awareness of only their immediate surroundings.

When applied to controlling the behavior of mobile sensors, swarming is accomplished by tracking the relative positions and behavior of nearby neighbors of a sensor node, and using that information to modify the node’s own behavior. For example, if a swarm seeks to maintain a consistent distance between neighbors, nodes might be repelled by neighbors that are too close and attracted to neighbors that are slightly beyond the desired distance. If the swarm is attempting to coordinate the motion of nodes to a single direction, nodes might align their own direction of travel with the average orientation of their neighbors. Nodes may also have their own tasks that are independent from the behavior of others. For example, in a traveling swarm some of the nodes might act as leaders, steering themselves and thus their neighbors towards
a known destination. These behaviors can be combined together as well. A natural example of such can be found in flocks of migrating geese, where younger members are not yet aware of what their destination will be. Older, more informed geese are responsible for controlling the direction of the flock, but still need to maintain their formation with uninformed members.

Using swarming to control the motion of mobile sensors resolves many of the issues faced by more centralized control. As nodes base their behavior entirely on local information, communication only needs to occur over a short distance. Since decisions are performed by the sensors themselves based on information that is either immediately available or sent from neighbors a single (or a few) communication hops away, they are able to react much more rapidly to changes in their environment than if they had to wait for instructions from a distant source. Finally, since all decisions are made locally, a malfunction in a single sensor will only ever disrupt the behavior of its immediate neighbors rather than the group as a whole.

There are many existing algorithms that use swarming to achieve some form of coverage. One technique involves taking advantage of the mobility of the sensors to gather readings at various points while the sensors move to adapt to the region. Turduev [24] takes this approach, recording sensor readings while the nodes seek out high strength areas of the region. However, this does not guarantee that all points of interest will be seen, and this approach is compromised if the region is changing over time. In some applications, only the boundary of the region is of interest. In these cases, a possible method is to have sensors seek out and travel along the edge or a level set, gathering data as they move. Examples of such a technique include Zhang [25], which uses a formation of mobile sensors to track level sets of a region, Sun [21], which uses a neural network to control a node's motion across contours, Hsieh [9], which uses collision avoidance algorithms to coordinate the motion of relatively large size nodes to and along a boundary, and Frihauf [8], which uses a leadership-based system to deploy nodes along a curve. This is an efficient means of gathering boundary data with small groups of sensors, and can adapt well to changes in a region over time. However, it
does not provide any data from the interior.

For this case, the ideal solution would be to adaptively track the boundary of the region, and scatter the sensors within that boundary. This way, up to date information at all of the sensed positions will always be available, and any changes to the region shape can easily be transferred to the swarm. Bertozzi [2] presents an algorithm for moving a set of nodes to align along the edge of a region, though the interior of the region is left empty. Kalantar [10] expands on this work with a similar system that selects only some of the nodes to seek the boundary, with the remaining nodes held within the interior in a uniform cluster shaped by the boundary.

My work expands on [10] by asserting more control over the relationship between boundary and interior nodes. Rather than using Kalantar et. al.’s approach of assigning a set of boundary nodes that remains fixed for all time, I present a system\(^1\) for dynamically adjusting the membership and behavior of the set of boundary nodes to consistently manage the swarm density within the interior and along the boundary.

To achieve coverage, a swarm needs to seek out the full extent of the region, and spread its mobile sensors across it. Any sensors that fall outside of the region will not be generating useful information, so in order to utilize the measuring capability efficiently, the swarm will need to constrain itself within the boundary of the region. If the region is changing over time, the swarm will need to evolve with the changing region. The problem is made more challenging by the fact that individual members of the swarm have limited awareness of their environment, only being able to directly measure their own position and communicate with neighbors over a limited range. Based entirely on local information, the sensors will need to coordinate their individual behaviors to carry out the global task of coverage, working together to discover the boundaries of the region and distribute themselves in a desired formation within it.

In Chapter 2, I present an algorithm that is designed to achieve uniform coverage

\(^1\) In the dissertation, swarming operations are presented and discussed at the level of algorithm without concerning issues such as messaging, communications, and collisions which are typically addressed at the level of protocol.
through only local information. In this algorithm, nodes are divided into two groups: Boundary and Interior. Boundary nodes form themselves into a chain around the edge of the swarm, and expand or contract the local swarm boundary in order to seek the edge of a region of interest. Interior nodes are repelled by both the boundary and each other, moving them towards a uniformly spaced stable equilibrium throughout the area within the chain of Boundary nodes. I present an analysis of the impact of a number of parameters on the performance of the algorithm, and show that it is able to correctly cover regions with challenging geometric features, as well as regions that are moving or deforming over time.

While most individual sensors can only gather data around their immediate location, it is possible to predict the state of a region in other locations through interpolating between those scattered measurements. Since sensors and thus measurements are generally limited in availability, it is important to ensure that the sensors are placed in optimal positions. If sensors are packed closely together in an area, then that means that the information gathered by those sensors will be overly similar and thus less individually useful, while other more sparsely populated areas may be undersampled. By controlling the spacing between nodes, they can be ensured to fall into positions that will more fully utilize their capabilities.

The algorithm presented in Chapter 2 will produce a uniform formation of nodes, but other forms of alignment are possible as well. One significant example is a Chebyshev distribution, in which nodes are spaced together more closely near the boundary of the swarm than in the interior. Chebyshev distributions generate optimal data points for rectilinear interpolation, meaning that they serve as candidates for determining sensor placement. Generating such a distribution requires a greater degree of coordination, since nodes will not inherently be aware of their global position relative to the boundaries and will need to share information with their neighbors to build that understanding. Even when they are aware of their locations, they need to correctly adapt their behavior as appropriate in order to yield the desired formation.
In Chapter 3, I present an algorithm designed to achieve a Chebyshev distribution through *virtual coordinate mapping*, based on a refinement of my earlier work in [11] and [12]. The algorithm establishes the relative positions of sensors within the swarm by linking chains of nodes between the boundary and center of the swarm, and uses those relative positions to build a set of virtual coordinates for the sensors, allowing them to interact based on their virtual positions instead of their real ones. In this way, a group of sensors in a Chebyshev formation can be made to view themselves as being in a stable uniform formation, or to converge to one if they are not. I present an analysis of the effect of modifying parameters related to virtual coordinate mapping, and show that the algorithm is able to produce a Chebyshev formation under a variety of conditions.

I also present a method for interpolation involving radial basis functions, and describe how it can be used to predict the state of a region at any position. I apply that technique to swarms with either uniform or Chebyshev distributions, and demonstrate how changing the distribution of nodes will affect the quality of interpolation.
Chapter 2
SWARMING FOR UNIFORM COVERAGE OF A REGION

In this chapter, we present a system for carrying out the task of swarm coverage, the distribution of a swarm of mobile sensors into stable positions across a region of interest such that the sensors gather useful information about the entirety of the region. For a region that is initially unknown, sensors will be required to coordinate among themselves to explore the region and position themselves throughout it, while relying only on local information.

We present a swarming algorithm designed to achieve uniform coverage of an unknown region of interest, based on dividing the tasks of exploration and distribution across two groups. Sensors in the boundary group form themselves into a chain around the edge of the swarm and expand or contract that chain to discover and match the boundary of the region, while sensors in the interior group expand to uniformly fill out the area within that chain. Uniformity between the boundary and interior is maintained by transferring nodes between the two groups as necessary.

We analyze the influence that a number of configuration parameters have on the behavior of the algorithm, as well as the ability of the algorithm to adapt to a variety of region shapes and sensor counts.

2.1 Coverage Through Multiple State Swarming

To identify and track the boundary and cover the interior, each node self-identifies as having one of two states that uniquely assign it to one of two groups. One group covers the boundary, and the other group covers the interior in a similar manner to the stateless algorithm. Rather than adapting to the strength of the sensed field, the interior nodes are constrained by repulsive interactions with the boundary.
The result is a system of mobile sensors that adaptively covers the region of interest, without extending beyond its boundary.

Kalantar [10] presents such an algorithm, but the systems produced by that algorithm lack some of the properties that would be necessary for them to achieve fully uniform coverage. In Kalantar’s algorithm, membership within the two groups is static, with no procedure for transferring nodes between groups. Since initial ratio between the boundary distance and interior area is not known apriori, the result is that the density of nodes along the boundary will inevitably differ from the density of nodes within the interior. The algorithm has no feature for controlling the influence of the boundary on the interior, the distance between the interior and the boundary is not managed relative to the boundary node separations or interior node separations.

In this chapter, we present an algorithm that comprehensively resolves these issues. We have three specific objectives for our algorithm:

1. We seek an algorithm where sensors react solely to the locally sensed field and on information from nearby sensors.

2. We seek an algorithm that covers a region of interest with sensors so that the sensor density is approximately uniform. Furthermore, we seek an algorithm where the spacing of sensors along the boundary matches the spacing in the interior.

3. We seek an algorithm that scales consistently, so that a solution obtained with one set of physical units is identical to solutions obtained by the algorithm applied to the same problem with a consistent change of variables into different physical units. (e.g. changing length measurements from meters to inches does not change the performance of the algorithm.) Ultimately, all key algorithmic parameters are dimensionless.

To meet these goals, our algorithm continuously tracks the density of the boundary and interior while nodes are in motion, and under certain conditions allows nodes to transfer between states in order to equalize the balance between the boundary and interior. Our algorithm also alters the interactions between nodes to manipulate the spacing of the gap between the boundary and interior. This algorithm covers regions
of a variety of sizes and shapes without any a priori knowledge of the sensed region using only local node spacings and the locally sensed field to drive its behavior.

2.1.1 Assumptions

The system consists of a set of independently functioning mobile nodes located on an infinite plane containing a measurable field that varies in strength. The region of interest is areas where the sensed field is greater than a threshold level of $F_B$. We assume that the sensed field is a non-negative, differentiable function, $F(x, t)$. Also, we assume that the region of interest, $R = \{x | F(x, t) \geq F_B\}$, is connected and uniformly bounded for all time. The algorithm is designed to operate on fully solid regions but may be able to adapt to regions with gaps in their interior, with the caveat that any gaps that are fully enclosed within the initial distribution of the swarm will not be discovered by the evolution of the boundary and will remain covered.

Nodes are capable of sensing both the strength and gradient of the field at their current position. They are also capable of determining the relative positions of other nearby nodes, either through measuring their absolute positions through a system such as GPS and communicating that information to other nodes, or through directly measuring relative angles and distances through a system such as sonar or infrared range-finding. They also maintain persistent values for their state and the estimated local spacing, and communicate those values with their neighbors along with their positions.

For the purposes of theoretical analysis, the nodes are assumed to be capable of moving at arbitrary velocities, with no limits on acceleration. The nodes are also capable of communication with other nearby nodes with negligible delay. The algorithm does not assume that these conditions hold or depend on them in order to function, but the experiments were focused on analyzing the functionality of the algorithm under more abstract conditions.
Table 2.1: Algorithm Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Semantics</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t$</td>
<td>Time interval between updates</td>
<td>time</td>
</tr>
<tr>
<td>$A_I$</td>
<td>Interior interaction strength</td>
<td>unitless</td>
</tr>
<tr>
<td>$k_I$</td>
<td>Interior interaction decay rate</td>
<td>unitless</td>
</tr>
<tr>
<td>$A_B$</td>
<td>Boundary interaction strength</td>
<td>unitless</td>
</tr>
<tr>
<td>$k_B$</td>
<td>Boundary interaction decay rate</td>
<td>unitless</td>
</tr>
<tr>
<td>$v_B$</td>
<td>Normal boundary speed factor</td>
<td>unitless</td>
</tr>
<tr>
<td>$v_{max}$</td>
<td>Maximum speed factor</td>
<td>unitless</td>
</tr>
<tr>
<td>$\theta_{min}$</td>
<td>Minimum angle gap for boundary classification</td>
<td>angle</td>
</tr>
<tr>
<td>$\alpha_{max}$</td>
<td>Scaling factor for boundary stability</td>
<td>unitless</td>
</tr>
<tr>
<td>$s_{max}$</td>
<td>Reassignment tolerance</td>
<td>unitless</td>
</tr>
<tr>
<td>$R_s$</td>
<td>Neighbor communication range factor</td>
<td>unitless</td>
</tr>
<tr>
<td>$D_{dist}$</td>
<td>Demoted node travel distance factor</td>
<td>unitless</td>
</tr>
<tr>
<td>$F_B$</td>
<td>Minimum field strength of the region</td>
<td>—¹</td>
</tr>
<tr>
<td>$\zeta$</td>
<td>Number of boundary subdivisions</td>
<td>unitless</td>
</tr>
<tr>
<td>$P(k_I, R_s)$</td>
<td>Ideal ratio of boundary to interior interactions</td>
<td>unitless</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Time scale</td>
<td>time</td>
</tr>
</tbody>
</table>

2.1.2 Algorithm overview

The fundamental purpose of our algorithm is to manage the behavior of mobile sensor nodes to provide uniform coverage of an a priori unknown sensed region. The algorithm functions by dividing the nodes into two primary state-based groups: boundary and interior. Nodes in the boundary state take on the task of exploring the region and distributing themselves uniformly along the region boundary once they have discovered it. Nodes in the interior state take on the task of distributing themselves uniformly within the interior of the region, limiting their spread to within the perimeter established by the boundary nodes.

A key part of our algorithm is determining when an interior node should become a boundary node or when a boundary node should become an interior node. When an interior node determines that it should join the set of boundary nodes, we refer to that node as being in the promoted state. When a boundary node determines that it

¹ Field strength is a scalar, whose unit is application dependent, such as temperature, acidity, etc.
Table 2.2: Node State Parameters

should join the set of interior nodes, we refer to that node as being in the demoted state.

Rather than relying on centralized control, the nodes carry out their behavior autonomously, self-selecting their states and deciding their behavior based on locally available information. The algorithm we present therefore applies to the control of a single node, with each node in the swarm carrying out its own instance of the algorithm independently of all others.

The algorithm as a whole is broken into multiple procedures, an index for which is given in Table 2.3. Procedure 1, presented in flowchart form in Figure 2.1, serves as the overall control for a node, initializing its initial state and running a timer-based event to update its motion and state at fixed intervals. All other procedures are invoked by Procedure 1, and carry out different aspects of the algorithm.

2.1.3 Scaling

For solutions to scale consistently, we consider the abstract problem of swarming over a region \( \tilde{R} = \{ \tilde{x} | f(\tilde{x}, \tilde{t}) \geq F_B \} \) where all variables with tilde’s are considered to be dimensionless. Our algorithmic solution to this problem will take the form
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Num</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Algorithm</td>
<td>Initialization and main loop</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>GetNodeSpacings</td>
<td>Determine and set local spacing for node</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>GetNodeBoundaryState</td>
<td>Identify if node is on boundary of swarm</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>GetNodeBoundaryNeighbors</td>
<td>Identify adjacent neighbors of boundary node</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>GetPerpendicularStability</td>
<td>Identify whether boundary node is moving perpendicular to swarm boundary</td>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td>GetParallelStability</td>
<td>Identify whether boundary node is moving parallel to swarm boundary</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>GetBoundaryInteractions</td>
<td>Set new velocity for boundary node</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>GetInteriorInteractions</td>
<td>Set new velocity for interior node</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>UpdateNodeState</td>
<td>Determine if node state should change and set if necessary</td>
<td>9</td>
<td>31</td>
</tr>
<tr>
<td>GetDemotedInteractions</td>
<td>Set new velocity for demoted node</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>GetPromotedInteractions</td>
<td>Set new velocity for promoted node</td>
<td>11</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 2.3: Index of procedures for algorithm
**Procedure 1** Algorithm for node behavior

1: ON INITIALIZATION:
2: Set node state to interior
3: Set node spacing to maximum possible sensing range
4: Immediately trigger update event
5:
6: ON UPDATE EVENT:
7: if \( n \) has received a promotion request since the last update then
8:  Set \( n \)’s state to promoted
9:  end if
10: \{Update node state\}
11: Run GetNodeSpacings
12: Run GetNodeBoundaryState
13: Run UpdateNodeState
14: Run GetNodeSpacings
15: \{Update node velocity\}
16: switch (State of node)
17: case interior:
18:  Run GetInteriorInteractions
19: case boundary:
20:  Run GetNodeBoundaryNeighbors
21:  Run GetBoundaryInteractions
22:  Run GetParallelStability
23:  Run GetPerpendicularStability
24: if \( \alpha^+ \) is true then
25:  Set \( \vec{v}_n \) to \( \vec{v}^+ \)
26:  end if
27: case promoted:
28:  Run GetPromotedInteractions
29: case demoted:
30:  Run GetDemotedInteractions
31: end switch
32: if \( \|\vec{v}_n\| > \frac{\alpha_n}{\tau} v_{max} \) then
33:  Rescale \( \vec{v}_n \) so that \( \|\vec{v}_n\| = \frac{\alpha_n}{\tau} v_{max} \)
34: end if
35: Broadcast current position and state of node to neighbors
36: Begin moving at velocity \( \vec{v}_n \)
37: Set timer to trigger update event in \( \Delta t \) time units
Figure 2.1: Flowchart showing the overall structure of the algorithm controlling the behavior of an individual sensor node.

of node trajectories $\vec{x}_n(\tilde{t})$ in response to a sensed field $f(\vec{x},\tilde{t})$. We require that all equivalent problems where the field has the form $F(x,t) = F(L\vec{x},\tau\tilde{t}) = f(\vec{x},\tilde{t})$, where $L$ is the characteristic length scale and $\tau$ is the characteristic time scale, should have the corresponding scaled solution, $\tilde{x}_n(t) = L\vec{x}(\tau\tilde{t})$. Many elements of our interactions depend on local estimates of node spacings ($s_n$ is the estimated mean node spacing near node $n$) which automatically scale with $L$. In practice, length and time scales are relevant when comparing the algorithm’s performance to different problems that are qualitatively similar by choosing the same dimensionless algorithmic parameters. If $L$ and $\tau$ are not known, comparisons between solutions can be made by treating one scale relative to another, as will be shown in a later example. For the remainder of this
dissertation, the interaction algorithms will be presented in dimensional form with $L$ (represented as local spacing $s_n$) and $\tau$ included appropriately.

Both types of nodes, boundary and interior, maintain an estimate of the spacing to nearby neighbors. This local estimate is used to adjust the behavior and interactions of the nodes. Boundary nodes exist in a single closed curve, so each boundary node will use a local spacing estimate that is the average distance to the node’s immediate clockwise and counterclockwise neighbors along the curve. Interior nodes estimate their spacing using the distance to the node’s immediately adjacent interior neighbors. Each interior node will use a local spacing estimate that is the average of the distances to the nearest three neighbors. Therefore, we require that each interior node have at least three adjacent interior neighbors even if it is located near a boundary. The algorithm used for determining the spacing $s_n$ of a node $n$ is given by Procedure 2.

In order to maintain full scalability of the algorithm, the range at which nodes can identify and communicate with their neighbors also scales with the length scale of the algorithm. The parameter that controls this is $R_s$, with $s_n R_s$ indicating the maximum sensing range for a node $n$. All nodes that fall within this range are considered to be neighbors of $n$ and will be incorporated into $n$’s behavior, while all nodes that fall outside that range will be ignored. This range is independent of any physical communication range for the nodes, meaning that distant nodes may be detected but not considered to be neighbors for the purposes of the algorithm. A diagram demonstrating the relationship between node spacing and interaction range is shown in Figure 2.2.

2.1.4 Uniform alignment in the boundary

As part of the algorithm, nodes classify themselves as one of two types: boundary or interior. Boundary nodes seek out the boundary of the region of interest and spread themselves uniformly across it, delineating the boundary for the remainder of the swarm.

As there are clearly no nodes beyond the boundary of a swarm, classification can be performed by searching for large areas in which no neighbors can be found.
Nodes convert the positions of their neighbors into angles off of their own positions, sort them, and look for the largest angular gap in which no nodes are present. If the gap is sufficiently large, then it is assumed to indicate that the node is adjacent to empty space, and is thus a boundary node. For our purposes we have chosen a minimum gap of $3\pi/4$, which in practice seems to provide a clear identification of the boundary with minimal fluctuation over time. Other values may function just as well, though values too close to $\pi$ could result in slightly recessed boundary nodes being classified as interior, while values that are too low could result in interior nodes being able to see gaps through the spacing of the boundary nodes. This method is similar to one described by McLurkin [17], though it is adapted towards the goal of building the denser boundaries needed for grid approximation, rather than the sparse boundaries [17] uses. An example of boundary classification is given in Figure 2.3.

One advantage of this method in comparison to one that relies on the strength of the sensed field is that it identifies nodes that are on the boundary of the swarm so that if the swarm is connected, the boundary nodes will form a simple closed chain.
**Procedure 2 GetNodeSpacings**

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with updated neighbor spacing \( s_n \)

1: if \( n \) is a boundary node then
2:   Set \( s_n \) to the average of the distances from \( n \) to \( n \)'s immediately adjacent boundary neighbors
3: else
4:   if \( n \) has at least three interior neighbors then
5:     Set \( s_n \) to the average of the distances from \( n \) to \( n \)'s three nearest interior neighbors
6:   else if \( n \) has at least one interior neighbor then
7:     {These next two cases are for elegantly handling edge conditions, and are not relevant in the normal operation of the algorithm}
8:     Set \( s_n \) to the average of the distances from \( n \) to \( n \)'s interior neighbors
9:   else
10:   Keep \( s_n \) at its current value
11: end if
12: end if

**Procedure 3 GetNodeBoundaryState**

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with \( \gamma_n \) set to \( True \) if \( n \) is determined to be on the boundary of the swarm or \( False \) if \( n \) is determined to be on the interior, and \( \beta_n \) set to the angle centered on the largest angle gap between neighbors of the node

1: Identify all neighbors of \( n \) within a distance of \( s_n R_s \)
2: if There are fewer than two neighbors within that range then
3:   Set \( \gamma_n \) to \( False \)
4: else
5:   Sort those neighbors of \( n \) into a circular list based on the angle of their positional displacement from \( \vec{x}_n \)
6:   Identify the adjacent neighbor pair with the widest gap between angles
7:   Set \( \beta_n \) to the angular midpoint of the gap
8:   Set \( \theta \) to the angular width of the gap
9:   if \( \theta \geq \theta_{min} \) then
10:      Set \( \gamma_n \) to \( True \)
11: else
12:      Set \( \gamma_n \) to \( False \)
13: end if
14: end if
**Procedure 4 GetNodeBoundaryNeighbors**

**Input:** The current state of node $n$

**Output:** The updated state of node $n$, with $C^+_n$ set to the node located immediately clockwise from $n$ on the chain of boundary nodes, and $C^-_n$ set to the node located immediately counterclockwise from $n$ on the chain of boundary nodes.

1: Identify all neighbors of $n$ within a distance of $s_nR_s$
2: if There are fewer than two neighbors within that range then
3: Set $C^+_n$ to $n$
4: Set $C^-_n$ to $n$
5: else
6: Sort those neighbors of $n$ into a circular list based on the angle of their positional displacement from $\vec{x}_n$
7: Identify the adjacent neighbor pair with the widest gap between angles
8: Set temporary variable $c^+$ to the clockwise neighbor of the pair
9: Set temporary variable $c^-$ to the counterclockwise neighbor of the pair
10: Build a list of clockwise candidates initially containing $c^+$
11: Build a list of counterclockwise candidates initially containing $c^-$
12: for all boundary nodes $m$ other than $c^+$ and $c^-$ with $\|\vec{r}_{nm}\| < s_nR_s$ do
13: Project a ray starting from $\vec{x}_m$ and traveling at angle $\beta_m$
14: if The ray passes between $n$ and $c^+$ then
15: Add $m$ to the list of clockwise candidates
16: end if
17: if The ray passes between $n$ and $c^-$ then
18: Add $m$ to the list of counterclockwise candidates
19: end if
20: end for
21: Set $C^+_n$ to the clockwise candidate located closest to $n$
22: Set $C^-_n$ to the counterclockwise candidate located closest to $n$
23: end if
Figure 2.3: Diagram showing the classification of nodes near the boundary of the swarm. The black circles represent boundary nodes, while the white circles represent interior nodes. $N$ and $T$ are the normal and tangent vectors of the boundary, while $\beta$ is the argument of the normal vector relative to standard coordinates.

enclosing the swarm while they seek out the edge of the sensed field. If nodes identify with the boundary of the sensed region, there is no guarantee that boundary nodes will have this property.

Boundary nodes follow two behaviors. First, they detect the strength of the field at their current position, and move themselves towards the boundary of the region of interest based on whether the sensed field value is lesser or greater than the boundary value. Second, they are repelled by all other boundary nodes based on their distances. Boundary nodes ignore interior nodes. As the convexity or concavity of the boundary results in interactions forcing nodes respectively outwards or inwards, these interactions are constrained to only act perpendicular to the edge of the swarm, allowing nodes to slide along the boundary but not leave it.

The outward unit normal $\vec{N}_n = [\cos(\beta_n), \sin(\beta_n)]$ at the position of a node along the boundary is based on the widest angle gap found during the boundary segmentation process, with the normal falling exactly in the middle of that gap, while $\vec{T}_n = [-\sin(\beta_n), \cos(\beta_n)]$ is a unit vector perpendicular to that normal. The equations of motion for boundary nodes have two components, a term $\vec{v}_n^\perp$ that controls motion perpendicular to the level set and a term $\vec{v}_n^\parallel$ that controls motion parallel to the level
set. Thus, a potential equation of motion for a boundary node \( n \) is as follows:

\[
\vec{v}_n = \vec{v}_n^\perp + \vec{v}_n^\parallel
\]  

(2.1)

Given \( N_b \) as the set of all boundary nodes, we define

\[
\vec{v}_n^\perp = \begin{cases} 
-v_B \frac{s_n}{\tau} \vec{N}_n, & \text{if } F(\vec{x}_n, t) > F_b \\
v_B \frac{s_n}{\tau} \vec{N}_n, & \text{if } F(\vec{x}_n, t) < F_b 
\end{cases}
\]  

(2.2)

\[
\vec{v}_n^\parallel = -A_B \frac{s_n}{\tau} \left( \sum_{m \in N_b} \exp\left(-\frac{k_B}{s_n} ||\vec{r}_{nm}||\right) \right) \cdot \vec{T}_n
\]  

(2.3)

where \( v_B \) is the region boundary tracking strength for boundary nodes, \( A_B \) is the repulsive boundary interaction strength, and \( k_B \) is the boundary interaction length scale.

While this formulation is sufficient to evolve nodes towards the boundary of the region, it can lead to nodes oscillating back and forth across the boundary rather than settling into stable positions. We propose a method using stability conditions to force nodes to immediately settle once they have reached appropriate positions. The stability of a node \( n \) perpendicular to the boundary is given by \( \alpha_n^\perp \), while the stability parallel to the boundary is given by \( \alpha_n^\parallel \). The equations for these conditions are

\[
\alpha_n^\perp = (F(\vec{x}_n) > F_b) \land \left(F(\vec{x}_n) + \nabla F(\vec{x}_n) \cdot \vec{v}_n^\perp \Delta t < F_b\right)
\]  

(2.4)

\[
\alpha_n^\parallel = \left(||\vec{v}_n^\parallel|| < \frac{s_n}{\tau} \alpha_{max}^\parallel\right)
\]  

(2.5)

where \( \Delta t \) is the time step, and \( \alpha^\parallel \) is a parameter controlling the minimal motion threshold required for stability. For a node \( n \), \( \alpha_n^\perp \) will be true if the node is currently inside the boundary of the region but a linear extrapolation of the next step of motion would cause it to cross to the outside, and false otherwise. \( \alpha_n^\parallel \) will be true if the net interactions with boundary neighbors lead to a velocity lower than \( \alpha_{max}^\parallel \), and false
**Procedure 5** GetPerpendicularStability

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with \( \alpha_n^\perp \) indicating whether the node’s current motion will move it from the interior to the exterior of the region

1: \{If the node senses a field value greater than the boundary level set and moving at the current velocity over the update interval is predicted to reduce the sensed field below the boundary level set\}

2: if \( F(\vec{x}_n) \geq F_b \) and \( F(\vec{x}_n) + \nabla F(\vec{x}_n) \cdot \vec{v}_n \Delta t < F_b \) then

3: Set \( \alpha_n^\perp \) to True

4: else

5: Set \( \alpha_n^\perp \) to False

6: end if

**Procedure 6** GetParallelStability

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with \( \alpha_n^\parallel \) indicating whether the node’s motion parallel to the level set is within the range allowed by \( \alpha_{\text{max}}^\parallel \)

1: \{If the component of the velocity parallel to the swarm boundary (perpendicular to the outward normal) is less than the node spacing times the boundary stability factor \( \alpha_{\text{max}}^\parallel \)\}

2: if \( \vec{v}_n^B < s_n \alpha_{\text{max}}^\parallel \) then

3: Set \( \alpha_n^\parallel \) to True

4: else

5: Set \( \alpha_n^\parallel \) to False

6: end if
otherwise. Procedure 5 will return the state of $\alpha_{n}^\perp$, while Procedure 6 will return the state of $\alpha_{\text{max}}^\parallel$.

The fact that $\alpha_{n}^\perp$ only tests for transitions from the interior to the exterior is intended, in order to lead to boundary nodes consistently settling just inside the region rather than possibly extending beyond it. If a boundary node does cross from the exterior to the interior, it will most likely reverse its motion at the next step, triggering the stability condition that way.

Using the perpendicular stability condition to throttle the motion of the node, the equation for perpendicular motion becomes

$$
\vec{v}_{n}^\perp = \begin{cases} 
0, & \text{if } (F(\vec{x}_n) > F_b) \land \alpha_{n}^\perp, \\
-v_B \frac{s_n}{\tau} \vec{N}_n, & \text{if } (F(\vec{x}_n) > F_b) \land \neg \alpha_{n}^\perp, \\
v_B \frac{s_n}{\tau} \vec{N}_n, & \text{if } (F(\vec{x}_n) < F_b) 
\end{cases} \quad (2.6)
$$

A similar alteration could be made to $\vec{v}_{n}^\parallel$ to halt parallel motion if the parallel stability condition is met, but this should be unnecessary as the net interaction strength parallel to the level set will already approach zero as the boundary nodes approach uniformity. While it is not used for constraining motion, $\alpha_{n}^\parallel$ will be useful elsewhere.

When combined, these two behaviors produce a single velocity that acts to push boundary nodes towards the boundary of the region of interest while simultaneously repelling other boundary nodes. At equilibrium, the nodes will have a uniform spacing along the boundary. The interaction strength controls the rate at which nodes arrive at equilibrium, but the equilibrium state is independent of the interaction strength. A diagram demonstrating the interactions and behavior of boundary nodes is shown in Figure 2.4.

**2.1.5 Uniform alignment in the interior**

Nodes that cannot see a sufficiently large gap are classified as interior nodes. These nodes spread out to cover the interior via repulsion. Interior nodes are repelled by both interior and boundary neighbors. Interactions with interior neighbors serve to
encourage nodes to spread out across all available space and settle into positions that are equidistant with their neighbors, producing uniform coverage. Interactions with boundary neighbors serve to restrict the spread of interior nodes, preventing them from leaving the interior of the swarm.

If the boundary interactions are not sufficient to constrain interior nodes within the swarm, then they will begin to flow outward though the gaps between the boundary nodes. This will be handled with minimal disruption by immediately reclassifying interior nodes that manage to reach the boundary as boundary nodes. However, this is not intended to serve as a primary means of transitioning between states, but rather as a means of resolving any edge cases that may arise.

It is important to assure that the interactions with the boundary are consistent regardless of the alignments of the boundary and interior nodes, so that interior nodes are not able to leak through the gaps between boundary nodes. To achieve this, instead of perceiving individual boundary nodes as sources of point interactions, interior nodes perceive pairs of adjacent boundary nodes as sources of continuous interactions along a line between the two boundary nodes.
**Procedure 7 GetBoundaryInteractions**

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with \( \vec{v}_n \) set to the net interaction with all neighbors and \( \vec{v}_B^n \) set to the net interaction with boundary neighbors

1: Set temporary variable \( \vec{n}_n \) to \([\cos(\beta_n), \sin(\beta_n)]\)

2: Set \( \vec{v}_B^n \) to \([0, 0] \)

3: \{Move towards boundary parallel to outward normal\}

4: if \( F(\vec{x}_n, t) > F_B \) then

5: Set \( \vec{v}_n \) to \(-\vec{v}_B^n \) \{If sensed value greater than level set, move outwards\}

6: else

7: Set \( \vec{v}_n \) to \( \vec{v}_B^n \) \{If sensed value lesser than level set, move outwards\}

8: end if

9: \{Feel exponentially decaying repulsion from boundary neighbors perpendicular to outward normal\}

10: for all boundary nodes \( m \) with \( \vec{r}_{nm} < s_n \ast R_s \) do

11: Set temporary variable \( \vec{n}_m \) to the normalized vector of \( \vec{r}_{nm} \)

12: Set temporary variable \( \vec{i}_m \) to \( -A_B s_n \tau \exp(\frac{k_b}{s_n} \parallel \vec{r}_{nm} \parallel) \vec{n}_m \)

13: Set \( \vec{v}_n \) to \( \vec{v}_n + \) the component of \( \vec{i}_m \) perpendicular to \( \beta_n \)

14: Set \( \vec{v}_B^n \) to \( \vec{v}_B^n + \) the component of \( \vec{i}_m \) perpendicular to \( \beta_n \)

15: end for

As the equations for interaction do not lend themselves to symbolic integration, numerical quadrature (trapezoid rule) is used to approximate the result of the integration. Given an interior interaction strength \( A_I \), boundary interaction decay rate \( k_I \), and boundary interaction strength ratio \( \rho \), along with a quadrature subdivision count of \( \zeta \), the equation for the interaction strength of a pair of adjacent boundary nodes \( j \) and \( k \) on an interior node \( n \) is thus

\[
\vec{r}_{nm} = \vec{r}(\vec{x}_n, \vec{x}_j) + \frac{m}{\zeta - 1} (\vec{x}_k - \vec{x}_j)) \quad (2.7)
\]

\[
\vec{f}_{jk} = \frac{A_I s_n}{\zeta} \sum_{m=0}^{\zeta-1} \exp \left( -\frac{k_I}{s_n} \parallel \vec{r}_{nm} \parallel \right) \frac{\vec{r}_{nm}}{\parallel \vec{r}_{nm} \parallel} \quad (2.8)
\]

with the assumption that the spacing \( j_n \) is valid for all generated points. The interactions are intended to be between individual nodes rather than pairs of boundary neighbors, so for an interaction with node \( j \), the quadrature is performed twice, for \( \vec{f}_{jC_j}^- \) and \( \vec{f}_{jC_j}^+ \), with each of the quadratures halved in strength. This is balanced by
the fact that each pair of boundary neighbors will be handled twice. (In Section 2.1.7, we will systematically derive a relationship between \( \rho \) and algorithm parameters so that the internal node spacing, boundary node spacing and the space between these two types of nodes is all the same. More specifically, we find that \( \rho = P(k_I, R_s) \), where \( P(k_I, R_s) \) is defined in Section 2.1.7.)

If \( N_B \) is the set of boundary nodes as before and \( N_I \) is the set of interior nodes, the equation controlling the behavior of an interior node \( n \) is

\[
\vec{v}_n = \sum_{j \in N_B} \frac{1}{2} \left( \vec{f}_{jC_j^+} + \vec{f}_{jC_j^-} \right) + A_I \frac{s_n}{\tau} \sum_{j \in N_I} \exp \left( -\frac{k_I}{s_n} \frac{\|\vec{r}_{nj}\|}{\|\vec{r}_{nj}\|} \right) \|\vec{r}_{nj}\| \quad (2.9)
\]

where \( C_j^+ \) is the boundary node immediately clockwise from boundary node \( j \), \( A_I \) is the repulsion interaction strength amongst interior nodes, and \( k_I \) is the interior interaction decay rate. A diagram demonstrating the repulsion experienced by an interior node from a pair of boundary nodes is given in Figure 2.5, while a diagram demonstrating the full set of interactions on an interior node is given in Figure 2.6.

![Diagram](image)

**Figure 2.5:** Diagram showing the interaction between an interior node, on the bottom, and a pair of boundary nodes on the top. The white circle represents an interior node, the black circles represent a pair of adjacent boundary nodes, while the dotted circles represent additional positions on a quadrature between the boundary nodes. The arrows represent the magnitude and direction of the interactions felt by the interior node.

Note that interior interactions ignore the strength of the sensed field. Controlling the shape of the swarm is left solely to the boundary nodes, and interior nodes simply
Procedure 8 GetInteriorInteractions

Input: The current state of node \( n \)

Output: The updated state of node \( n \), with \( \vec{v}_n \) set to the net interaction with all neighbors

1: Set \( \vec{v}_n \) to \([0, 0]\)
2: for all Nodes \( m \) with \( r_{nm} < s_n \ast r_s \) do
3:   if \( m \) is a boundary node then
4:      {Feel exponentially decaying repulsion from piecewise linear interpolation of boundary neighbors}
5:      Set temporary variable \( \zeta_{int} \) to \( \frac{1}{\zeta + 1} \)
6:      Set temporary variable \( \zeta_R \) to \( \frac{1}{\zeta + 2} \)
7:      Set temporary variable \( o \) to \( C_n^- \)
8:      Set temporary variable \( \rho \) to \( P(k_I, R_s) \)
9:      {Use quadratures of \( \zeta \) points to approximate interpolations between adjacent boundary nodes}
10:     for \( j \) over the range 0 to 1 at intervals of \( \zeta \) do
11:        Set temporary variable \( \vec{x}_p \) to \( \vec{x}_m + (\vec{x}_o - \vec{x}_m)j \)
12:        Set temporary variable \( \vec{n}_p \) to the normalized vector of \( r_{np} \)
13:        Set temporary variable \( \vec{i}_p \) to \( -\zeta_R A_I \rho \frac{\vec{x}_m}{s_n} \exp\left(\frac{k_I}{s_n} ||\vec{r}_{np}\||\right) \vec{n}_p \)
14:        Set \( \vec{v}_n \) to \( \vec{v}_n + 0.5\vec{i}_p \)
15:    end for
16:    end if
17:    if \( m \) is an interior node then
18:      {Feel exponentially decaying repulsion from individual interior neighbors}
19:      Set temporary variable \( \vec{n}_m \) to the normalized vector of \( r_{nm} \)
20:      Set temporary variable \( \vec{i}_m \) to \( -A_I \frac{\vec{x}_m}{s_n} \exp\left(\frac{k_I}{s_n} ||\vec{r}_{nm}\||\right) \vec{n}_m \)
21:      Set \( \vec{v}_n \) to \( \vec{v}_n + \vec{i}_m \)
22:    end if
23: end for
fill out the space confined by the boundary nodes. Furthermore, equilibrium states depend upon the ratio of $A_I$ and $\rho$. That is, increasing $\rho$ can drive interior nodes further away from the boundary.

### 2.1.6 Uniform boundary and interior spacing

Repulsive interactions will drive interior nodes to have uniform spacing within the swarm, and similarly, boundary interactions will drive the boundary nodes to having uniform spacing along the edge of the sensed field. However, our scheme does not guarantee that the spacing between interior nodes will be the same as the spacing between boundary nodes. Since it is desirable to control the relationship between the spacings, either to achieve a common separation between all nodes or for some other purpose, we introduce an algorithm to promote nodes from the interior to the boundary or demote nodes from the boundary to the interior. In this dissertation, we focus on making the interior and boundary node spacings the same.

This algorithm is a dynamic process for adjusting the interior and boundary spacings toward a specified goal. The dynamics are not guaranteed to converge to
a fixed configuration, and we will explore the sensitivity of the algorithm to the key control parameter. This algorithm selects particular nodes to be moved between the boundary and interior in order to adjust the ratio between the local mean boundary spacing and the local mean interior spacing. Demoted boundary nodes move themselves to deep within the swarm, and promoted interior nodes move themselves to the boundary. In a swarm, the decision to promote or demote a node must be made by each node based on local information shared by its neighbors.

Before a boundary node can begin the process of node reassignment, it must first settle into a quasi-steady configuration along the boundary. This is determined by the stability conditions described in (2.4) and (2.5). If $\alpha_n ^\perp$ and $\alpha_n ^\parallel$ hold for a boundary node $n \in N_B$, then that node is considered to be eligible for performing boundary reassignment. The perpendicular stability and to a lesser extent the parallel stability conditions ensure that the boundary is not actively evolving, while the parallel stability condition prevents new changes until the disruption caused by demoting or promoting a node has resolved, as these disruptions reduce the accuracy of the density estimates.

At each cycle, each eligible boundary node $n$ with spacing $s_n$ identifies the closest interior neighbor $m \in N_I$, and that node’s spacing $s_m$. using its own spacing and the density of the interior neighbor as the boundary and interior densities.

If $s_n > s_m$, then this means that the local spacing on the boundary is greater than the spacing on the interior. To correct for this, $m$ may need to be promoted from the interior to the boundary. As a completely perfect spacing match is impossible, the actual test used for this is $s_n > s_m s_{\text{max}}$, where $s_{\text{max}}$ is the reassignment tolerance and the key parameter governing the dynamics of the reassignment algorithm. Even if this condition is met, there is the possibility that other nearby nodes may already be in the process of transitioning from the interior to the boundary, which may be sufficient to correct for the difference on their own. Therefore, $n$ searches for already promoted nodes within the radius $a_c$, where

$$a_c = 2s_n$$

(2.10)
If there is no other node with state promoted within a distance of $a_c$ from $n$, then the promotion is clear to happen, and $m$ is promoted, setting $S_m$ to promoted.

Likewise, if $s_n < s_m$, then the local spacing on the interior is wider than the spacing on the boundary. To correct for this, $n$ may need to be demoted from the boundary to the interior. As a completely perfect spacing match is impossible, the actual test used for this is $s_n < \frac{s_m}{s_{\text{max}}}$. Even if this condition is met, there is the possibility that other nearby nodes may already be in the process of transitioning from the boundary to the interior, which may be sufficient to correct for the difference. Therefore, $n$ searches for already promoted nodes within the radius $a_c$. If there is no other node with state demoted within a distance of $a_c$ from $n$, then the demotion is clear to happen, and $n$ is demoted, setting $S_n$ to demoted.

The algorithm assumes that local information about state changes is instantly communicated, allowing neighbors to immediately act on that information. If communication delay is a factor, then the algorithm might be modified to require that nodes would announce their desire to transition and wait for their neighbors to acknowledge that their states have not changed before actually transitioning.

Demoted boundary nodes move towards the interior of the swarm, while ignoring all other nodes. As the gradient of the field is not assumed to be known, demoted nodes instead orient their motion relative to the nearest boundary. The equations for the motion of a demoted node $n$ are

$$
\vec{v}_n = -v_B \frac{s_n}{\tau} R_{-\frac{\pi}{2}} \frac{\vec{r}_{C_m^{-}, C_m^{+}}}{\|\vec{r}_{C_m^{-}, C_m^{+}}\|}
$$

where $m$ is the closest boundary node to the demoted node, $C_m^{-}$ and $C_m^{+}$ are the counterclockwise and clockwise immediate boundary neighbors of $m$, and $R_{-\frac{\pi}{2}}$ is a rotation matrix.

Demoted nodes continue in this state until they determine that they have sufficiently embedded themselves within the interior nodes. This is done by comparing the distance to the nearest node $m \in N_B$ with its spacing value $s_m$. As the ratios of
Procedure 9 UpdateNodeState

Input: The current state of node \( n \)

Output: The updated state of node \( n \), with updated node state \( S_n \).

1: {If a boundary node has somehow moved into the interior of the swarm without becoming demoted}
2: if \( S_n \) is boundary and \( \gamma_n \) is False then
3: Set \( S_n \) to interior
4: end if
5: {If a promoted node has reached the boundary of the swarm}
6: {Alternatively, if an interior node has somehow moved into the boundary of the swarm without becoming promoted}
7: if \( S_n \) is interior or promoted and \( \gamma_n \) is True then
8: Set \( S_n \) to boundary
9: end if
10: if \( S_n \) is demoted then
11: Set \( m \) to the nearest boundary neighbor of \( n \)
12: {If a demoted node is farther from the nearest boundary node than the demotion factor \( D_{dist} \) times the boundary spacing}
13: if \( \| r_{nm} \| \geq D_{dist}s_m \) then
14: Set \( S_n \) to interior
15: end if
16: end if
17: {Stable boundary nodes check whether promotion or demotion is necessary}
18: if \( S_n \) is boundary and \( \alpha_n^\perp \) is True and \( \alpha_n^\parallel \) is True then
19: Set temporary variable \( m \) to the nearest interior neighbor of \( n \)
20: {If the interior is denser than the boundary, promote an interior node}
21: if \( s_R > s_{max} \) then
22: Set temporary variable \( \eta \) to the greater value of \( \frac{0.5}{s_m} \) or \( 2s_n \)
23: if There is no promoted node within a range of \( 0.5\eta \) of \( n \) then
24: Send a promotion request to \( m \)
25: end if
26: end if
27: end if
28: {If the boundary is denser than the interior, demote the boundary node}
29: if \( s_R < s_{max}^1 \) then
30: Set temporary variable \( \eta \) to the greater value of \( \frac{0.5}{s_m} \) or \( 2s_n \)
31: if There is no demoted node within a range of \( 0.5\eta \) of \( n \) then
32: Set \( S_n \) to demoted
33: end if
34: end if
35: end if
local boundary and interior spacings are used to determine whether subsequent pro-
motions or demotions are necessary, the temporary increases in interior density caused
by demoted nodes changing to interior nodes can lead to immediate re-promotions.
Therefore, it is imperative that demoted nodes travel a sufficient distance within the
swarm as not temporarily disrupt the spacing measurements of other boundary-facing
interior nodes. If \( \| \vec{r}_{n,m} \| > s_m D_{\text{dist}} \), where \( D_{\text{dist}} \) is a parameter controlling the relative
travel distance for demotions, then the demoted node \( n \) is determined to have traveled
sufficiently far into the interior, and \( S_n \) is set to interior.

**Procedure 10 GetDemotedInteractions**

**Input:** The current state of node \( n \)

**Output:** The updated state of node \( n \), with \( \vec{v}_n \) set to the net interaction with all
neighbors

1: Set temporary variable \( m \) to the nearest Boundary neighbor of \( n \)
2: Set temporary variable \( \vec{n}_n \) to the normalized vector of \( \vec{r}_{C_m} - m \) rotated by \( -\frac{\pi}{2} \)
3: \{Move at factor of spacing away from nearest boundary neighbor and its two ad-
   jacent neighbors, towards interior\}
4: Set \( \vec{v}_n \) to \( -v_B \frac{s_n}{\tau} \vec{n}_n \)

Promoted interior nodes move towards the boundary of the swarm, while avoid-
ing boundary or other promoted nodes. As the gradient of the field is not assumed to
be known, promoted nodes instead orient their motion relative to the nearest boundary.
The equations for the motion of a promoted node \( n \) are

\[
\vec{v}_n = \vec{v}_n^\perp + \vec{v}_n^\parallel
\]

\[
\vec{v}_n^\perp = v_B \frac{s_n}{\tau} \frac{\vec{r}_{C_m} - C_m^+}{\| \vec{r}_{C_m} - C_m^+ \|} R(\vec{r}_{C_m} - m, C_m + m) \quad (2.13)
\]

\[
\vec{v}_n^\parallel = -A_B \frac{s_n}{\tau} \left( \sum_{m \in N_n, \| \vec{r}_{nm} \| < s_n R_s} \exp \left( -\frac{k_B}{s_n \| \vec{r}_{nm} \|} \right) \frac{\vec{r}_{C_m} - C_m^+}{\| \vec{r}_{C_m} - C_m^+ \|} \right) \quad (2.14)
\]

Promoted nodes continue in this state until they have reached the boundary of
the region, determined by their sensed field value transitioning from greater than to
lower than the boundary threshold. When this occurs, a promoted node changes its
state from promoted to boundary.
Procedure 11 GetPromotedInteractions

Input: The current state of node \( n \)

Output: The updated state of node \( n \), with \( \vec{v}_n \) set to the net interaction with all neighbors

1: Set temporary variable \( m \) to the nearest Boundary neighbor of \( n \)
2: Set temporary variable \( \vec{n}_n \) to the normalized vector of \( \vec{r}_{CC_m,C_m} \) rotated by \(-\frac{\pi}{2}\)
3: \{Move at factor of spacing towards nearest boundary neighbor and its two adjacent neighbors\}
4: Set \( \vec{v}_n \) to \( v_B\vec{n}_n \)
5: \{Feel exponentially decaying repulsion from boundary neighbors perpendicular to outward motion\}
6: for all Boundary or Promoted nodes \( m \) with \( \vec{r}_{nm} < R_s \) do
7: Set temporary variable \( \vec{n}_m \) to the normalized vector of \( \vec{r}_{nm} \)
8: Set temporary variable \( \vec{i}_m \) to \(-A_B\frac{s_m}{r}\exp\left(\frac{k_B}{s_m}\|\vec{r}_{nm}\|\right)\vec{n}_m \)
9: Set \( \vec{v}_n \) to \( \vec{v}_n + \) the component of \( \vec{i}_m \) perpendicular to \( \vec{n}_n \)
10: end for

2.1.7 Theoretical analysis of node interactions

As boundary nodes interact only with other boundary nodes, and nodes buried deep within the interior will only interact with other interior nodes, both configurations will inherently equilibrate to a uniform spacing. However, interior nodes located near the boundary, where both terms of (2.9) are of comparable magnitude, are not guaranteed to have uniform spacing. Instead, there will be a region occupied by interior nodes separated from the boundary by a gap width \( \Delta \) (see Figure 2.7). We refer to the interior nodes located adjacent to the gap as “gap front nodes” or simply “gap nodes.” The key to understanding equilibrium coverage is therefore understanding the behavior of gap nodes. We can gain significant insights into the behavior of the swarm coverage algorithm by analyzing a model configuration. We will explore the functional relationship between the gap width \( \Delta \), the swarming parameters and node density ratios.

We examine the case where the boundary is linear, a useful simplifying assumption for cases where the radius of curvature is large. Without loss of generality, the equilibrium position(s) along which interactions from the interior and boundary nodes balance exactly will be taken to be the \( x \)-axis. For this model system, the horizontal
Figure 2.7: Diagram showing the placements of nodes in the model configuration. The bottom side contains a set of interior nodes placed in a triangular grid of width $s_I$, shown as white circles, centered on a single boundary-facing interior node, shown as a gray circle. The grid extends to a distance of $s_I R_s$ from the central node. The top side contains a continuous arrangement of nodes representing the boundary nodes, shown as black circles, with the density of the field equal to $1/s_B$. There is a gap between the boundary and the beginning of the interior of width $\Delta$. Gap nodes such as the one indicated in the center of the diagram are adjacent to the gap.

components of the interactions will cancel to zero by symmetry.

To understand the interplay between algorithm parameters, we will approximate the interactions due to interior and boundary nodes on a representative gap node. We approximate the interior components of the interactions by assuming that there will be a stable uniform node distribution in the lower half-plane, which is represented as a regular triangular grid of nodes with a spacing of $s_I$ between vertices. As the range over which nodes interact is limited, only positions within a distance of $s_I R_s$ are considered, where $R_s$ is the interaction range measured in spacings. The neighbors within that range form a set $C$, with the position of node $m \in C$ given by $\vec{x}_m = s_I \vec{x}'_m$. Note that for any given $m$, $\vec{x}'_m$ will be constant regardless of the values of any other parameters.

From (2.9), the sum of all interior interactions $F_I$ on the representative gap node is given by

$$\vec{F}_I = \sum_{m \in C} A_I \frac{s_I}{\tau} e^{-k_I \|\vec{x}'_m\|} \frac{\vec{x}'_m}{\|\vec{x}'_m\|}$$

(2.15)

With this rescaling, we see that $k_I$ is the parameter controlling the rate of exponential
decay, independent of node spacing. If we define

$$\vec{F}_I(k_I, R_s) = \sum_{m \in C} e^{-k_I \|\vec{x}_m\|} \frac{\vec{x}_m}{\|\vec{x}_m\|},$$  \hspace{1cm} (2.16)$$
then the equation can be expressed compactly as,

$$\vec{F}_I = A_I \frac{S_I}{\tau} \vec{F}_I(k_I, R_s).$$  \hspace{1cm} (2.17)$$

The boundary components of the interactions are approximated by assuming that the boundary neighbors will be uniformly spaced along a line parallel to the x-axis at a distance of $\Delta$, with the spacing between boundary nodes given by $s_B$. As the algorithm already uses quadrature to approximate interactions with the boundary as if they were continuous, the boundary is simply represented as a continuous line of density $s_B^{-1}$, which is integrated to determine the overall interaction.

If we parametrize the boundary node positions by $\vec{x} = xi + \Delta \vec{j}$, the net boundary interactions on the gap node are given by

$$\vec{F}_B = \frac{1}{s_B} A_I \rho \frac{s_I}{\tau} \int_{-\sqrt{(Rs_{sI})^2 - \Delta^2}}^{\sqrt{(Rs_{sI})^2 - \Delta^2}} e^{-k_I \|\vec{x}\|} \frac{\vec{x}}{\|\vec{x}\|} \, dx. \hspace{1cm} (2.18)$$

Then, (2.18) can be transformed to

$$\vec{F}_B = \frac{2}{s_B} A_I \rho \frac{s_I}{\tau} \int_{0}^{\sqrt{(Rs_{sI})^2 - \Delta^2}} e^{-k_I \sqrt{x^2 + \Delta^2} \frac{\Delta}{\sqrt{\Delta^2 + x^2}}} \, dx, \hspace{1cm} (2.19)$$

where $\vec{F}_B$ is taken to be the vertical interactions only because the horizontal interactions are zero by symmetry.

For the rest of this dissertation, we will focus on the special case when $\Delta$ will be equal to the spacing, $s_I$, between interior nodes$^2$, so we substitute $s_I$ for $\Delta$. Changing

$^2$ While it is natural for there to be a single spacing requirement across the region of interest, it is simple to relax this assumption to allow individually specified interior, boundary and gap spacings.
variables in (2.19) substituting in $\tilde{x} = x/s_I$ and $d\tilde{x} = dx/s_I$ yields

$$\vec{F}_B = \frac{2}{s_B} A_I \rho \frac{s_I^2}{\tau} \left( \frac{R_I^2 - 1}{s_B} \right) \int_0^{P_B(k_I, R_S)} e^{-k_I \sqrt{1 + \tilde{x}^2}} \frac{1}{\sqrt{1 + \tilde{x}^2}} d\tilde{x}. \quad (2.20)$$

There is no simple analytic expression for integral $P_B(k_I, R_S)$, but it can be evaluated quickly numerically:

$$\vec{F}_B = A_I \rho \frac{s_I^2}{s_B \tau} P_B(k_I, R_S). \quad (2.21)$$

Combining interior and boundary interactions, it is possible to predict the effect of various combinations of parameters on the behavior of the gap node. For both the interior and boundary interactions to be balanced, we require that

$$A_I \frac{s_I}{\tau} P_I(k_I, R_s) + A_I \rho \frac{s_I^2}{s_B \tau} P_B(k_I, R_s) = 0. \quad (2.22)$$

Therefore, a requirement for a gap node to be stationary is:

$$\rho = -\frac{s_B}{s_I} \frac{P_I(k_I, R_s)}{P_B(k_I, R_s)} = \frac{s_B}{s_I} P(k_I, R_s). \quad (2.23)$$

For the remainder of this dissertation, we simplify this further and assume the boundary and interior spacing is the same so,

$$\rho = P(k_I, R_s). \quad (2.24)$$

For any given $k_I$ and $R_s$, the value of $\rho = P(k_I, R_s)$ can be precomputed and used in the algorithm.

For example, in Figure 2.8 in which $A_I = 1$, $k_I = 2$, $s_I = s_B$, $R_s = 3$, we can visualize gap node behavior in the $(s_I, \Delta)$-plane. We see that the $(s_I, \rho)$-plane consists of two regions, one where gap nodes move toward the boundary and another where they move toward the interior. Separating the two is a curve corresponding to stationary configurations. Since $F_B$ and $F_I$ are monotonic as a function of distance from the boundary, the stationary configurations are stable.
Figure 2.8: Contour plot of edge node velocities toward the interior when $\rho$ is determined by (2.24). In this case, $s_I = \Delta$ is a stable equilibrium by design.
2.2 Experiments

To evaluate its performance, we implemented the swarming algorithm in Matlab. We update positions using a forward Euler scheme with a small enough timestep to reduce overcorrections. For simplicity, all nodes were updated simultaneously and communication between nodes was assumed to be both instantaneous and perfectly accurate.

Unless otherwise indicated, the following parameters are used in each run. The swarm consisted of a group of 200 nodes. The nodes are initially randomly placed within a circle of radius 250, centered at the origin. The duration of each run is 10 time units, with a time step of $\Delta t = 0.05$. The default parameters used across all experiments are given in Table 2.4.

<table>
<thead>
<tr>
<th>$A_I$</th>
<th>$k_I$</th>
<th>$A_B$</th>
<th>$k_B$</th>
<th>$v_B$</th>
<th>$v_{\text{max}}$</th>
<th>$\theta_{\text{min}}$</th>
<th>$\alpha_{\text{max}} \parallel R_s$</th>
<th>$s_{\text{max}}$</th>
<th>$D_{\text{dist}}$</th>
<th>$F_B$</th>
<th>$\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2</td>
<td>20</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0.75\pi</td>
<td>1</td>
<td>3</td>
<td>1.2</td>
<td>2.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 2.4: Default Experiment Parameters

2.2.1 Gaussian field

As a demonstration of the basic functionality of the algorithm, and a basis for comparison to more complex scenarios, the algorithm was first run on a field shaped like a circular 2D Gaussian, with field strength given by

$$F(x, y) = \exp \left[ -\left( \frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right],$$

where $(x, y)$ are the coordinates of a point within the field centered on $(0, 0)$ and $\sigma_x = 500$ and $\sigma_y = 500$ are the standard deviations of the Gaussian.

Figure 2.9 shows several points of interest within the run. Figure 2.9a shows the state at $t=0$, with the nodes in their starting positions. Figure 2.9b shows the state at $t=1$, where some of the boundary nodes have reached the edge of the region of interest and have begun to seek to equalize their spacings, while others have yet to reach the edge. Figure 2.9c shows the state at $t=1.5$, where the boundary of the swarm is nearly fully established but the node spacings on the interior are still in the
process of establishing uniformity. Finally, Figure 2.9d shows the state at t=3, where
the nodes have reached near uniformity.

The behavior and progress towards stability of the nodes can also be shown
through analysis of their motion over time. Figure 2.10 shows the mean speeds of
the nodes through the run. There are several distinct phases to the behavior of nodes
worth noting. The first, taking place from roughly t=0 to t=0.25, consists of the initial
realignment from the initial random positions, with nodes that happen to begin overly
close strongly repelling each other. From t=0.25 to t=1.5, the spread of the swarm
begins to dominate, with nodes moving increasing rapidly to fully cover the region
of interest, accelerating as their overall spacings increase. From t=1.5 to t=3, nodes
have started to reach the positions needed for coverage, and their speed begins to fall
off as they settle into their final positions. Throughout that process, promotions or
demotions are visible as temporary spikes in the mean speeds.

Another means of analysis involves examining the estimated spacings for all of
the nodes. Ideally, as the swarm approaches stability, these estimates should converge
towards a single value, indicating that the nodes have become uniformly spread across
the region. Figure 2.11 shows the mean spacings across the run for the boundary nodes
and interior nodes, as well as the standard deviations for those groups.

There are several aspects of the data worth noting. The first is that the boundary
spacings converged to a value slightly greater than the final value for the interior
spacings. This is reasonable and expected, as the algorithm has set tolerance $s_{\text{max}}$
built into the selection criteria for node promotions and demotions. Reducing this
tolerance will reduce this gap, at the cost of possibly increasing the time needed for
the swarm to stabilize, or for sufficiently low tolerances preventing the swarm from
stabilizing at all.

The second is that the boundary node spacings all managed to converge to a
single value, while even when the swarm has stabilized, the interior node spacings still
show a range of values. This is also reasonable, due to the differences in difficulties
of the tasks of the two types of nodes. Achieving uniformity across the boundary
Figure 2.9: Node positions at various points in time during the simulation on a circular region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.
Figure 2.10: Mean node speeds during the circular simulation.
is a one-dimensional problem. Achieving uniformity in the interior requires nodes to balance their spacings across multiple directions in a two-dimensional plane. In addition, the accuracy of interior spacings is also more difficult to measure, due to the difficulty of determining adjacency in a gridless swarm. The variance in the interior spacings is therefore likely due to both actual variance and a degree of measurement error. Ultimately, the swarm converges to a state where the interior node spacings and boundary node spacings are the same to within a controllable tolerance $s_{\text{max}}$.

### 2.2.2 Interaction decay rate ($k_I$ and $k_B$)

In order to demonstrate the influence that the interaction decay rate parameters $k_I$ and $k_B$ have on the behavior of the algorithm, a series of tests were run in which the values of those parameters were varied. The algorithm was run on the Gaussian in Equation 2.25, with $\sigma_x = 1000$ and $\sigma_y = 1000$. In total, 25 runs were carried out, in sets of 5 with the values of $k_I$ and $k_B$ equal within a set, varying over the range of $(1, 1.5, 2, 2.5, 3)$. 

Figure 2.11: Mean node spacings during the circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.
To control our study of the impact of decay rate on swarming behavior, we normalize $A_I$ and $A_B$ so that the interactions at one node spacing, $s_n$, will be independent of $k_I$ and $k_B$. To fix this interaction strength, we set $A_I$ and $A_B$ to be $A_I = 16 \exp(k_I - 2)$ and $A_B = 20 \exp(k_I - 2)$. The primary effect of $k_I$ and $k_B$ thus becomes to control the degree to which interactions change when nodes are not at the ideal distance of $s_n$, with higher values causing interaction strengths to increase or decrease more rapidly. For low values of $k_I$, the interactions felt from nodes closer than $s_n$ are fairly similar to the interactions felt from nodes further than $s_n$, meaning that nodes that are too close will only generate slightly stronger repulsions, while nodes that are too far will continue to generate significant opposition to movement to fill in the gap. For higher values of $k_I$, closer nodes will start to generate much more extreme repulsions, while distant nodes will effectively vanish.

Figure 2.12 shows the mean speeds of the nodes throughout each of the runs, with the results from each set of runs with the same values of $k_I$ averaged together. Overall the majority of the runs showed similar behavior, with lower values of $k_I$ yielding slightly greater peak node speeds. While it appears at a glance that the runs with $k_I = 1.5$ take longer to resolve, it can be seen that by $t = 4$, the performance of those runs has become nearly identical to that of the others. This implies that the resolution of a uniform alignment is not being delayed as much as nodes are simply moving faster during it. As decreasing the value of $k_I$ will increase the significance of more distant neighbors, it appears that non-uniformity of those neighbors will begin to have a significant impact on node behavior at lower values of $k_I$. As this results in faster node motion while not actually improving the time required to achieve stability, the ideal behavior would likely be achieved for $k_I > 2.5$, where further increases do not result in further reductions in node speed. While this implies that greater values of $k_I$ will be inherently superior, there is a limit to that benefit, as increasing the speed of nodes will cause the algorithm to react more strongly to any change, meaning that minor or temporary disruptions could cause disproportionately strong reactions across the entire swarm, while a lower value of $k_I$ would allow the change to be incorporated.
Figure 2.12: Mean node speeds based on modifying $k_I = k_B$. 
smoothly.

The runs with $k_I = 1$ were also a notable exception to the performance of the algorithm. In this case, the failure to resolve to a stable alignment is not due to the interactions failing to achieve uniformity, but to issues with how demotions are resolved. With $k_I = 1$ and the default value of $D_{dist} = 2.5$, nodes that have recently transitioned from demoted to interior will have a range of significant influence wide enough to disrupt the behavior of gap nodes. This causes a temporary drop in detected spacings, strong enough to meet the algorithm’s conditions for node promotion. As the promoted nodes are being added to a boundary that just triggered a valid demotion, this creates a cycle of promotions and demotion, preventing stabilization. This could be resolved by altering $D_{dist}$ to send demoted nodes even deeper, but that would mean that demotion would take longer to completely resolve. Making $D_{dist}$ too large could also limit the sizes of the regions on which the algorithm can operate, since regions that are small or thin may have areas where demoted nodes may find it difficult or impossible to reach a distance of $D_{dist}$ from the boundary of the swarm. Based on observation from the experiments, the most practical solution to avoid this issue appears to be to limit $k_I$ to values of roughly 1.5 or greater.

2.2.3 Interior node strength and boundary seeking velocity ($A_I$ and $v_B$)

In order to demonstrate the relative influence of the interior node strength $A_I$ and the boundary seeking velocity $v_B$, a series of tests were run in which the relative values of those parameters were varied, with the value of $A_I$ varying over the range of $(8, 16, 32, 64, 128)$ and the value of $v_B$ remaining constant at $v_B = 4$. Keeping $v_B$ consistent across all runs is reasonable due to the relationship that $A_I$ and $v_B$ have with the time scale $\tau$, with altering those parameters (along with $A_B$, which is restricted to motion along the boundary and thus has only a minor effect on overall swarm evolution) while keeping their relative proportions constant yielding the same effect as inversely altering $\tau$. Because of this, only the ratio between $A_I$ and $v_B$ will influence the actual behavior of the swarm.
We explored the algorithm’s performance on the Gaussian (2.25), with $\sigma_x = 1000$ and $\sigma_y = 1000$. In total, 25 realizations were carried out, in sets of 5 with 5 different the values of $A_I$. Figure 2.13 shows the mean speeds of the nodes throughout each of the runs, with the results from each set of runs with the same values of $A_I$ averaged together. As would be expected, changing $A_I$ alone does not significantly alter the behavior of nodes on the boundary. However, it does have a significant impact on the behavior of nodes in the interior of the swarm, with greater values of $A_I$ causing interior nodes to move more rapidly while a swarm is expanding and beginning to settle into a uniform distribution, but ultimately to reach a stable formation much sooner. This increase in mean node speed is not directly proportional to $A_I$, which indicates that the expansion rate of the boundary imposes a significant constraint on the motion of interior nodes.

While greater values of $A_I$ will cause a swarm to approach a stable formation faster, this comes at the cost of significantly faster node motion within the interior. The ideal value for $A_I$ will therefore depend on whether a particular application favors
faster resolutions or minimal node motion. The highest reasonable value for $A_I$ is ultimately limited by the timestep, $\Delta t$.

### 2.2.4 Maximum interaction range

In order to demonstrate the influence of the maximum interaction distance $r_{\text{max}}$ on the behavior of the algorithm, a series of tests were run with the value of $r_{\text{max}}$ equal to $(2, 3, 4, 5, 6, 7)$. $r_{\text{max}}$ controls the maximum range, measured in spacings $s_n$, at which nodes will interact. Greater values of $r_{\text{max}}$ will therefore allow interaction with more distant neighbors. Of course, the impact of greater values of $r_{\text{max}}$ decays exponentially, but lower values could begin to omit neighbors that would otherwise have a significant impact.

We explored the algorithm performance on a Gaussian region (2.25), with $\sigma_x = 1000$ and $\sigma_y = 1000$ with results averaged over 5 realizations for each value of $r_{\text{max}}$. Figure 2.14 shows the mean node speeds throughout each of the runs, with the results from each set of runs with the same values of $r_{\text{max}}$ averaged together. While the runs exhibited similar behavior, there were a few notable differences. The peak mean speed is greater for higher values of $r_{\text{max}}$, implying that the inclusion of more distant neighbors makes a minor difference on the way that nodes interact on a smaller scale while the swarm is still expanding. Once the configuration settles, there is less variation implying that there is no long-term impact. The most significant difference between the runs came as the nodes settled into their final positions. When the interaction range is too small, $r_{\text{max}} = 2$, the swarm will fail to settle into a stationary configuration. These results indicate that including more distant nodes is unlikely to destabilize a stationary configuration, while having too few interacting nodes will.

### 2.2.5 Node reassignment tolerance

In order to demonstrate the influence of the node reassignment tolerance $s_{\text{max}}$ on the behavior of the swarming algorithm, a series of tests were run with the value of $s_{\text{max}}$ equal to $(1.1, 1.15, 1.2, 1.25, 1.3)$. $s_{\text{max}}$ controls the maximum relative spacing difference
Figure 2.14: Mean node speeds based on modifying $r_{\text{max}}$. 
for interior and boundary nodes, with any local relative differences that exceed that value triggering a promotion or demotion. As the value of $s_{\text{max}}$ increases, the algorithm will accept increasingly greater errors in the relative spacing of the swarm. While greater values will result in a less uniform distribution, they also allow a swarm to reach stability faster due to requiring fewer adjustments to reach an acceptable ratio and having a greater tolerance for minor fluctuations in spacing while nodes settle. If the value of $s_{\text{max}}$ is sufficiently low, then the disruptions generated by promotions or demotions may be enough to significantly delay or even prevent the swarm from reaching stability.

We explored algorithm performance on a Gaussian 2.25 with $\sigma_x = 1000$ and $\sigma_y = 1000$ as before with 5 realizations for each value of $s_{\text{max}}$ equal. Figure 2.15 show the output of each of the runs, with the results from each set of runs with the same values of $s_{\text{max}}$ averaged together. There are two ways by which the accuracy of the spacing is measured. The first, shown in Figure 2.15a, compares the ratio of the mean...
of the spacing of all of the boundary nodes over the mean of the spacing of all of
the interior nodes. Once a swarm achieves equilibrium, the value will be lower than
\( s_{\text{max}} \) because promotion and demotion dynamically reassigns spacings if they exceed
this value. When \( s_{\text{max}} = 1.15 \), we see the swarm fails to achieve equilibrium, but still
fluctuates near a low spacing ratio in a quasi-stationary state. When \( s_{\text{max}} = 1.1 \) the
algorithm does not achieve an equilibrium.

The issues with these low values of \( s_{\text{max}} \) likely stem from the fixed search radius
of \( a_c = 2s_n \) used to identify conflicts when promoting or demoting nodes. For values
of \( s_{\text{max}} \) that are too low, legitimate conflicts may exist at a distance greater than \( a_c \),
leading to an oscillating cycle of over-correcting state changes. For \( s_{\text{max}} = 1.15 \) the state
changes settle towards a stable equilibrium, while for \( s_{\text{max}} = 1.1 \) the error is significant
enough for the swarm to remain unstable. It appears that to avoid instability, the
ideal error tolerance for \( a_c = 2s_n \) is therefore roughly \( s_{\text{max}} = 1.2 \), which is the value
used for the other experiments. Lower tolerances may be achievable by increasing \( a_c \),
but this comes at a cost of increasing the distance over which nodes are required to
communicate.

The second plot in Figure 2.15b compares the proportion of nodes that fall
within the interior \( N_I \) to the total number of nodes \( N \). If interior nodes are as-
sumed to ideally fall evenly within a circle of radius \( r \), while boundary nodes are
assumed to ideally fall evenly along the circumference of that circle, then the rela-
tionship between \( N_I \) and \( N \) can be expressed as \( \sqrt{\pi r^2/N_I} = 2\pi r/(N - N_I) \), or
\( N_I = N - 2\sqrt{\pi N + \pi + 2\pi} \), indicating equal spacing for the boundary and inter-
ior. As the runs used a node count of \( N = 200 \), the ideal count of interior nodes is
therefore \( N_I = 200 - \sqrt{\pi \sqrt{200 + \pi + 2\pi}} = 155.76 \), for a ideal ratio of approximately
.78. The results are similar to the first analysis, with lower values of \( s_{\text{max}} \) resulting
in proportions closer to the ideal ratio. The realizations with \( s_{\text{max}} = 1.1 \) showed sig-
nificantly lower counts of interior nodes than would be expected due to the swarms’
continued instability. Nodes that would otherwise be counted as interior nodes are tied
up within promotions and demotions instead. Similar behavior can be observed for a
short time when $s_{\text{max}} = 1.15$ though the ratio returns to a more normal value as the swarm stabilizes.

2.2.6 Adaptation to region size

In this section, we explore how the swarming algorithm adapts to regions of different sizes. Again we use Gaussian regions but with different variances. Some areas are larger than the initial size of the swarm while others are smaller. The swarm will need to grow or shrink to match the region, while dynamically rescaling its behavior to adapt to its new size.

2.2.6.1 A larger region

In order to test the ability of the swarming algorithm to handle situations requiring the swarm to expand to cover regions larger than their original area, the algorithm was run on a region of larger size than the original run. The algorithm was run on the Gaussian $2.25$ with $\sigma_x = 1000$ and $\sigma_y = 1000$. (In the previous section, we used 500.) Note that all swarm algorithm parameters remain the same, so the swarm has no apriori knowledge of the region.

Figure 2.16a shows the mean node speeds throughout the run, while Figure 2.16b shows the mean node spacings. Like for the original circular region, there is a noticeable period of initial adjustment during the beginning of the run. Note that since the distances are greater, the nodes move faster.

2.2.6.2 A smaller region

We also explore how the algorithm performs when the region of interest is smaller than the initial configuration of nodes. We tested the algorithm on a Gaussian (2.25) with $\sigma_x = 125$ and $\sigma_y = 125$ resulting in a smaller region than the initial node distribution within the same 250 radius circle. To cover the region, the boundary of the swarm needs to contract, compressing the interior nodes along with it.

Figure 2.17a shows the mean node speeds throughout the run, while Figure 2.17b shows the mean node spacings. Having the swarm actually shrink to match the
Figure 2.16: Mean node speeds and spacings during the larger circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.

region changes the behavior even more. The rapid reorganization at the start of the run is still visible, but unlike in cases where the region is larger than the initial area covered by the swarm, the mean node speeds continuously decrease rather than forming a peak. This is due to inherently scaling behavior of the algorithm, with the speeds of nodes being inversely proportional to the increasing density of the swarm.

2.2.7 Adaptation to region shape

These test cases are intended to measure how well the algorithm can adapt to regions of different shapes other than the simple circles that have been used previously.

2.2.7.1 Square

In order to test the ability of the swarming algorithm to adapt to regions containing either straight edges or sharp corners, the algorithm was run on a square-shaped region. The field was similar in nature to that of the circular Gaussian, with the exception that the field strength was determined by the greater of the differences in $x$
or $y$ positions from the midpoint of the region, rather than both simultaneously. The equation for the field strength is given by

$$F(x, y) = \exp \left[ -\max \left( \frac{x^2}{2\sigma_x^2}, \frac{y^2}{2\sigma_y^2} \right) \right]$$  \hspace{1cm} (2.26)

where $(x, y)$ are the coordinates of a point within the field centered on $(0, 0)$ and $\sigma_x = 500$ and $\sigma_y = 500$ are parameters controlling the width of the Gaussian.

Figure 2.18 shows the final results of the run at $t=10$. Figure 2.19a shows the mean node speeds throughout the run and Figure 2.19b shows the mean node spacings. Node behavior during the earlier phases of the run was fairly similar to that of the original circle, indicating that the algorithm had little difficulty expanding the swarm to fill the square region. However, it had more difficulty during the later stages of evolution, where the boundary and interior equilibrated. As indicated by the irregularity of the mean node speeds, a large number of promotions or demotions were performed. Much of this can be attributed to the corners, as interior nodes found
Figure 2.18: Node positions at the end of the square simulation.
Figure 2.19: Mean node speeds and spacings during the square simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.

it more difficult to spread into them, and the spacing estimates for interior nodes extending away from the body of the swarm might have been less accurate.

2.2.7.2 Crescent

In order to test the ability of the swarming algorithm to adapt to regions containing concavities, the algorithm was run on a crescent-shaped region. The field was composed of a combination of two circular Gaussians, with a wider curve providing the basic body of the region and a narrower negative curve removing a smaller section. The equation for the field strength is given by

\[ F(x, y) = \exp \left[ - \left( \frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right] - 2\exp \left[ - \left( \frac{x^2}{2\sigma_x'^2} + \frac{(y - 200)^2}{2\sigma_y'^2} \right) \right] \]  

(2.27)

where \( \sigma_x = 500 \) and \( \sigma_y = 500 \) are parameters controlling the width of the region, and \( \sigma_x' = 175 \) and \( \sigma_y' = 175 \) control the width of the removed area.

Figure 2.20 shows the final results of the run at \( t=10 \). Figure 2.21a shows the mean node speeds throughout the run, while Figure 2.21b shows the mean node speed.

\[ \text{(a) Mean Speeds} \quad \text{(b) Mean Spacings} \]
Figure 2.20: Node positions at the end of the crescent simulation.
spacings. The nodes were able to cover most of the region, but the 'horned' features where the curvature is greatest challenged the algorithm. Some of the nodes never equilibrate but rather continuously undergo promotion and demotion. This is reflected both in the speeds and spacings, with the mean speed never reaching zero.

The cause is an inherent issue with the boundary repulsion algorithm when dealing with concave regions. Normally, interior nodes near the boundary of the swarm are held in place by repulsion from both the boundary and their interior neighbors. However, in the case of the outermost nodes in the horns of the crescent, the concave boundary leads to much of the boundary repulsion canceling itself out, while there are no interior neighbors to constrain the node from moving towards the end of the horn. Those nodes drift out of position until they are close enough to the boundary to be reclassified as boundary nodes themselves. This leads to an immediate local spacing disparity between the now denser boundary and the interior, which is corrected through demotion. Other interior nodes will continue to drift out of position and cause a continuing cycle of demotions.

As this issue is primarily caused by the angles of convex boundaries making repulsion less efficient, this issue will not be resolved by altering parameters. Rather, there is an inherent limit on the curvature that the algorithm is capable of handling relative to the node spacing. One way to address the problem would be to add additional nodes to the swarm.

2.2.8 Adaptation to region motion

These test cases are intended to measure how well the swarming algorithm can adapt to regions that are moving or deforming over time.

2.2.8.1 Translating circle

In order to test the ability of the swarming algorithm to handle regions that change position over time, the algorithm was run on a translating circular region. The
Figure 2.21: Mean node speeds and spacings during the crescent simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.

The field was based on the Gaussian equation

\[ F(x, y, t) = \exp \left[ - \left( \frac{(x - x_0(t))^2}{2\sigma_x^2} + \frac{(y - y_0(t))^2}{2\sigma_y^2} \right) \right] \]  

(2.28)

with fixed spread parameters \( \sigma_x = 1000 \) and \( \sigma_y = 1000 \), and \( x_0(t) \) and \( y_0(t) \) representing the horizontal and vertical positions of the midpoint of the region at time \( t \). The simulation was run for 25 time units, with the midpoint of the region remaining at \((0, 0)\) from \( t = 0 \) to \( t = 3 \). From \( t = 3 \) to \( t = 18 \), the midpoint of the swarm moved \( \vec{v} = 100\vec{i} + 100\vec{j} \), finishing at \((1500, 1500)\) at \( t = 18 \). From \( t = 18 \) to \( t = 25 \), the midpoint remained stationary. The number of nodes used for this particular run was 100, rather than the 200 count used for the others.

Figure 2.22 shows several points of interest within the run. Figure 2.22a shows the state at \( t = 3 \), where the translation is just about to begin. Figure 2.22b shows the state at \( t = 10 \), midway through the translation. Note that the nodes in the trailing side of the swarm have become compressed together. This is due to the motion of the swarm being driven not by matching velocities, as would be the case for swarming...
algorithms designed primarily to move a swarm to a distant target, but by expansion
along the front, with interior nodes moving to fill the empty space, and compression
along the rear, with interior nodes being squeezed together. Some distortion is thus
inevitable. Figure 2.22c shows the state at \( t=18 \), where the translation has just ended.
Note that the formation is similar to the one at \( t = 10 \), indicating that it would have
likely remained stable for any duration of region motion at that same speed. Finally,
Figure 2.22d shows the state at \( t=25 \), where the nodes have settled back into a uniform
alignment.

Figure 2.23a shows the mean node speeds throughout the run, while Figure
2.23b shows the mean node spacings. As would be expected, the motion of the swarm
is similar to the behavior of the swarm in a static region right up until \( t = 3 \), when
the region begins to translate. From there, the nodes begins to accelerate toward the
speed of the region \( (s = 141) \) actually reaching it at roughly \( t = 5 \). The differences in
node densities in the front and rear of the swarm lead to greater variances in spacings,
though the overall mean of the spacings remains fairly constant. Once the region ceases
motion at \( t = 18 \), it is able to settle into a more uniform distribution fairly quickly,
though the differences in densities across the region tend to cause state promotions
and demotions as the swarm settles.

2.2.8.2 Deforming region

In order to test the ability of the swarming algorithm to handle regions that
deform their boundaries over time, the algorithm was run on a region consisting of
multiple 2D Gaussians that varied in intensity over time. The field was based on the
Figure 2.22: Node positions at various points in time during the simulation on a circular region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.
Figure 2.23: Mean node speeds and spacings during the translating circular simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.

equation

\[
F(x, y, t) = (1 - B(t)) \exp \left[ -\left( \frac{(x - 600)^2}{2400^2} + \frac{y^2}{2400^2} \right) \right] \\
+ (1 - B(t)) \exp \left[ -\left( \frac{(x + 600)^2}{2400^2} + \frac{y^2}{2400^2} \right) \right] \\
+ B(t) \exp \left[ -\left( \frac{x^2}{2400^2} + \frac{(y - 600)^2}{2400^2} \right) \right] \\
+ B(t) \exp \left[ -\left( \frac{x^2}{2400^2} + \frac{(y + 600)^2}{2400^2} \right) \right],
\]  

(2.29)

where

\[
B(t) = \begin{cases} 
0, & 0 \leq t < 2, \\
(t - 2)/5, & 2 \leq t < 7, \\
1, & 7 \leq t \leq 10 
\end{cases}
\]

(2.30)

This field is a time-dependent superposition of two horizontally aligned Gaussians and two vertically aligned Gaussians.
Figure 2.24 shows several points of interest within the run. Figure 2.24a shows the state at $t=2$, where the deformation is about to begin. The swarm has not equilibrated into the starting shape. Figure 2.24b shows the state at $t=4.5$, at which point the four Gaussians are of equal strength. Similar the translating circle, the nodes near the contracting edges have become compressed relative to the nodes near the expanding edges. Figure 2.24c shows the state at $t=7$, where the deformation has ended. The nodes are relatively close to their final positions, despite the horizontal compression that has occurred. Finally, Figure 2.24d shows the state at $t=10$, where the nodes are close to their final equilibrium.

Figure 2.25a shows the mean node speeds throughout the run, while Figure 2.25b shows the mean node spacings. While the deformation began at $t = 2$, the interior nodes had not begun to approach uniformity by that time, so their speeds and spacings remained similar to that of a static region until $t = 3$. At that point the motion required by adaptation to the deforming boundary began to dominate, with the mean speeds and the spacing variances rising. Unlike for the translating region, the boundary of the region did not change at a constant rate, with the greatest changes being concentrated around the period where both the horizontal and vertical curves were of similar magnitude, centered around $t = 4.5$. Once motion ceased the swarm was able to equilibrate, though it can be seen from the mean speeds and spacing variance at $t = 10$ that the process had not fully completed by the end of the simulation.

2.2.9 Adaptation and scale

In order to demonstrate the homogeneity of the algorithm over length and time scales, a set of experiments were performed in which the length $L$ and time scales $\tau$ were altered, but all other parameters related to the swarm including the initial relative node positions remained the same.

In all cases, the nodes were initially randomly placed within a $250L$ radius circle, centered on $(0,0)$, with the seed for the random placement remaining the same across all
Figure 2.24: Node positions at various points in time during the simulation on a deforming region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.
Figure 2.25: Mean node speeds and spacings during the deforming shape simulation. The middle curve shows the mean, while the upper and lower curves show one standard deviation from it.

runs. The algorithm was run for $10\tau$ time units, with updates occurring with spacing $\Delta t = 0.05\tau$.

The region was based on Equation (2.28), with spread parameters $\sigma_x = 1000L$ and $\sigma_y = 1000L$. The midpoint of the region remained at $(0,0)$ from $t = 0$ to $t = \tau$. From $t = \tau$ to $t = 6\tau$, the midpoint of the swarm moved a distance of $(100L\tau^{-1},100L\tau^{-1})$ every time unit, or at a speed of $\sqrt{100^2 + 100^2}L\tau^{-1} \approx 141.4L\tau^{-1}$, finishing at $(500L,500L)$ at $t = 6\tau$. From $t = 6\tau$ to $t = 10\tau$, the midpoint remained constant. Note that very little adjustment needed to be made for the length scale $L$ other than altering the initial node positions and the field equation because the algorithm scaling node interactions based on perceived spacing. The algorithm cannot inherently account for time scale, however, meaning that many parameters had to be adjusted to account for any changes.

Three runs were performed in total, using identically seeded initial node positions within a circle of radius $500L$. The first used the scaling parameter values $L = 1$
and $\tau = 1$, making the algorithm’s behavior the same as for the previously described translating circle. The second used the scaling parameter values $L = 2$ and $\tau = 1$, resulting in a region that was twice as large and moved twice as fast to a position twice as far away over the same period of time as the first. The third used the scaling parameters $L = 1$ and $\tau = .5$, resulting in a region that was the same size as the first, but moved twice as fast, ending its run in half the time.

Figure 2.26 shows the positions of the nodes for the $L = 1$ and $\tau = 1$ run at $t = 0$ when the run has just begun, at $t = 1$ when the field begins to move, at $t = 6$ where the field halts, and at $t = 10$ when the run has ended. The other two runs showed almost identical behavior over their dimensional scales. There were variances in the positions of individual nodes over time, but these were likely due to computation error magnified by the significance of small variations in deciding node classifications, rather than any real deviation across dimensional scales.

As a verification of the correct scaling, Figure 2.27 shows the mean speeds across the run for the baseline, larger, and faster runs. With the axes adjusted to be non-dimensionalized, it can be seen that the behavior for each of the cases is nearly identical.

2.3 Conclusions

We have demonstrated that our swarming algorithm is capable of uniformly distributing nodes across a region of interest under a variety of conditions, including irregularly shaped regions and regions that are evolving over time. Under the basic assumption that the region is simply connected and can be covered by nodes (e.g., the problem is solvable), the swarm robustly and dynamically adapts itself to the scale of its environment using only local interactions. The key elements of the algorithm include classic notions of repulsive interactions together with a procedure for nodes to locally assign themselves to one of two states, boundary or interior, and a dynamic procedure for nodes to change their state under certain circumstances. Furthermore, the
Figure 2.26: Node positions at various points in time during the simulation on the baseline scaling region of interest. Blue circles represent interior nodes, red circles represent boundary nodes, empty blue circles represent promoted nodes, and empty red circles represent demoted nodes. The edge of the region of interest is indicated by the dotted line.
Figure 2.27: Mean speeds for the three dimensional rescaling runs, scaled based on $L$ and $\tau$. 
algorithm, based on our local analysis, balances interior node spacings and boundary node spacings to achieve uniform coverage of spatially complex and moving regions.

There are many parameters in swarming algorithms in general, and this particular algorithm is no exception. We have taken care to identify those specific parameters on which the algorithm is sensitive. For these, we have provided detailed analysis and demonstrated their impacts on the performance of the algorithm. For others such as $\eta$, $v_{\text{max}}$, $D_{\text{dist}}$, there is much more flexibility, and altering them by small amounts does not lead to a qualitative difference in performance.
Chapter 3
INTERPOLATION

In this chapter, we present a system for carrying out the task of swarm interpolation, the use of sampled data from a swarm of mobile sensors to construct a function that approximates a sensed field. Assuming that the region is initially unknown, sensors will be required to coordinate among themselves to move to positions that are ideal for producing accurate interpolations.

Building on the uniform formation algorithm presented in Chapter 2, we present a swarming algorithm designed to produce non-uniform sensor formations based on an adaptation of the Chebyshev distribution into two dimensions. This algorithm uses a technique called virtual coordinate mapping to reshape the positions of sensors into virtual coordinates, and carries out swarm interactions in that virtual space. As sensors converge to a stable uniform formation in virtual space, those positions are mapped into real space as a stable Chebyshev formation.

We analyze the capabilities of both the uniform and Chebyshev distributions generated by our algorithm to accurately interpolate unknown regions of a variety of shapes and complexities, using a system based on radial basis functions.

3.1 Chebyshev distributions and interpolation

The problem we tackle with our framework is called scattered data interpolation [6]. In a nutshell, given an input set \{((\vec{x}_1, \phi_1), (\vec{x}_2, \phi_2), \ldots, (\vec{x}_N, \phi_N))\}, where \(N\) is the number of sensors, \(\vec{x}_n \in \mathbb{R}^2\) represents the location of the \(n\)th sensor, and \(\phi_n = f(\vec{x}_n)\) is the \(n\)th sensor’s measurement of the variable of interest (represented by the evaluation of the function \(f\), whose definition is not known). Our goal is to find
a function $g$ such that $g(\vec{x}_i) = \phi_i$ for all $i$ and that the difference between $g$ and $f$ at locations different from $\vec{x}_n$, $n = 1, \ldots, N$ is as small as possible.

When interpolating fields with a large number of measurements, the distribution of interpolating nodes is crucial for minimizing error. One option is to use low order splines. Another is to use higher order polynomial interpolants, but these might produce large oscillations near the boundary of the region, an effect which is known as Runge’s phenomenon. It has been shown that this effect can be reduced by controlling the arrangement of data points used for the interpolation. One effective distribution is based on the roots of a Chebyshev polynomial [19] (See [1], [23], and the references contained therein for a general discussion). The following system yields the roots of the desired Chebyshev polynomial:

$$T_0(x) = 1, \quad T_1(x) = x,$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (n \geq 1) \quad (3.1)$$

Restricting the domain to $[-1,1]$, for instance, Chebyshev polynomials can be specified by

$$T_n(x) = \cos(n \cos^{-1} x) \quad (n \geq 0) \quad (3.2)$$
where $n$ is the desired number of roots for the polynomial. See Fig. 3.1(a) for an example.

Using the positions of the roots of a Chebyshev polynomial as interpolation nodes for a 1D field leads to an error formula of

$$
|f(x) - p(x)| \leq \frac{1}{2^n(n+1)!} \max_{|t| \leq 1} |f^{(n+1)}(t)|,
$$

(3.3)

where $f(x)$ is the function being interpolated and $p(x)$ is the interpolation polynomial based on the Chebyshev roots. The distribution can be further extended from 1D into a 2D circle. One example is shown in Fig. 3.1(b), where the points along the radial axis are distributed according to the roots of a Chebyshev polynomial and along the angular axis they are uniformly distributed.

While a mesh-aligned Chebyshev distribution, with all of the data points placed on a polar grid, is traditionally used for polynomial interpolation, aligning a swarm to a grid is a difficult problem when the area to be covered is not a priori known. However, generating a similar but meshless distribution, with sensors arranged in an arbitrary alignment but a similar overall density, is a simpler matter. In Chapter 2 we present an algorithm that produces meshless uniform distributions, which we can use as a base for an algorithm that generates non-uniform distributions instead. In order to achieve a Chebyshev distribution, a special form of swarming will need to be performed, one that is sensitive to the absolute positions of the sensors within the swarm, not just the relative positions of their neighbors. We finesse our previous algorithm by altering the distances measured between sensors. If the perceived positions of the sensors are transformed such that a Chebyshev distribution appears to the sensors to be a uniform distribution, then the sensors will naturally settle into an arrangement which close to an appropriate Chebyshev distribution as they swarm. This method is similar to the coordinate remapping algorithm to control formations of small groups of agents described by Sabbatini [20], but expands its scope, applying it to hundreds of agents simultaneously.
In this chapter, we present a model for achieving this distribution by means of the previously mentioned coordinate transformations. The system is specifically intended for achieving Chebyshev-like distributions, but the framework is flexible enough to also allow for alternate distributions to be included. This could be particularly useful if certain features require more attention in the sensed region, such as a specific value range or subregion. This framework will also work well with various base swarming algorithms that may or may not produce uniform distributions.

3.2 Radial basis function interpolation

In the kinds of applications we are envisioning, all the data we will have at our disposal are measurements at the robots' locations. Thus, our input is a set \( \{(\vec{x}_1, \phi_1), (\vec{x}_2, \phi_2), \ldots, (\vec{x}_N, \phi_N)\} \), where \( N \) is the number of robots, \( \vec{x}_n \in \mathbb{R}^2 \) represents the location of the \( n \)th robot, and \( \phi_n = f(\vec{x}_n) \) is the \( n \)th robot's measurement of the variable of interest (represented by the evaluation of the function \( f \), whose definition is not known). Our goal is to find a function \( g \) such that \( g(\vec{x}) = \phi \) and that the difference between \( g \) and \( f \) at locations different from \( \vec{x}_n, n = 1, \ldots, N \) is as small as possible. This problem is known as scattered data interpolation \([6]\).

We tackle this problem using radial basis function interpolation. Our goal is to find the values of the coefficients \( c_k, k = 1, \ldots, N \) such that

\[
g(\vec{x}) = \sum_{i=1}^{N} c_i \varphi(\|\vec{x} - \vec{x}_i\|),
\]

where \( \varphi \) is a radial basis function, and \( \| \cdot \|_2 \) is the Euclidean norm. The radial basis functions used in our experiments are Gaussians of the form

\[
\varphi(r) = e^{-(ar)^2},
\]

where \( a \) is a parameter called shape parameter. By enforcing the condition \( g(\vec{x}_i) = y_i \), the coefficients \( c_i, i = 1, \ldots, N \) can be found by solving the linear system \( A\vec{c} = \vec{\phi} \) where the entries \( A_{jk} \) of the matrix \( A \) are equal to \( \varphi(\|\vec{x}_j - \vec{x}_k\|_2) \), \( j, k = 1, \ldots, N \), \( \vec{c} = [c_1, c_2, \ldots, c_N]^T \), and \( \vec{\phi} = [\phi_1, \phi_2, \ldots, \phi_N]^T \).
Radial basis function interpolation has a significant advantage over polynomial interpolation in that it allows for meshless interpolation, while polynomial interpolation requires that data points fall in a grid. However, it maintains many of the properties of polynomial interpolation. It has been demonstrated that for low shape parameters, 1D RBF interpolation is equivalent to 1D polynomial interpolation [4]. RBF interpolation for higher dimensions is less well studied, but we hypothesize that similar properties should hold, including the impact of node distributions on the magnitude of Runge's phenomenon.

We use two error measures to determine the accuracy of a particular interpolation. The first measure is the root-mean-square error (RMS error), which represents the accuracy of the interpolation over the region as a whole. Minor errors that extend over a large area will have a significant effect on this value, while errors that are stronger but more localized will have less influence. The RMS error is computed as follows:

\[
\|g - f\|_2 = \sqrt{\frac{1}{M} \sum_{j=1}^{M} (g(\vec{E}_j) - f(\vec{E}_j))^2},
\]

where \(\vec{E}_j\), \(j = 1, \ldots, M\) are the evaluation points.

The second measure is the maximum error, which represents the greatest error that the interpolation experiences. This value will be dependent on the strength of localized effects such as Runge’s phenomenon, while minor errors will have no influence. The maximum error is computed as follows:

\[
\|g - f\|_\infty = \max\{|g(\vec{E}_j) - f(\vec{E}_j)|\}.
\]

As the ideal value for the shape parameter \(a\) is initially unknown, the interpolation algorithm is performed on a range of possible values for \(a\), with the value that produces the lowest RMS error or maximum error selected as the ideal case for any given instance.
As part of the goal of this chapter is to evaluate the relative capabilities of uniform and Chebyshev distributions in providing accurate estimates, the relative errors for the two types of formations on any given field are also of interest. This is determined by dividing the errors for the Chebyshev formations by the errors for the corresponding uniform formations. A value less than 1 indicates that the Chebyshev algorithm is more accurate for that field, while a value greater than 1 indicates that the uniform algorithm is more accurate.

### 3.3 Forming Chebyshev distribution

In order to move from a uniform distribution to a Chebyshev distribution, the coordinates can be remapped in such a way that they appear uniform when the nodes are properly distributed.

The simplest expression of a Chebyshev distribution is with robots positioned along a line segment, with the robots clustered densely towards the ends and sparsely in the middle. This distribution can be obtained from a uniform distribution by normalizing the points, taking their cosines, and reversing the normalization. In order for a formation that is already a Chebyshev distribution to appear as if it is uniform, this operation must be inverted. The formula for doing so for a node \( n \), assuming that the points lie along the x axis, is

\[
\vec{v}x_n = 0.5(x_{\text{max}} - x_{\text{min}}) \left( 1 - \arccos \left( \frac{\vec{x}_n - 0.5(x_{\text{max}} + x_{\text{min}})}{0.5(x_{\text{max}} - x_{\text{min}})} \right) \right) + \vec{x}_n
\]

where \( x_{\text{max}} \) is the high endpoint of the region, \( x_{\text{min}} \) is the low endpoint of the region, \( \vec{x}_n \) is the real position of \( n \), and \( \vec{v}x_n \) is the calculated virtual position of \( n \).

Extending a Chebyshev distribution from a line into a circle requires a shift in the way coordinates are handled, above and beyond simply adding an additional dimension. The same basic density distribution is present, but rather than simply remapping both the \( x \) and \( y \) axes, the coordinates need to be remapped along every line passing through the midpoint of the region.

This can be accomplished by converting the coordinates of the robots from Cartesian to polar, centered on the midpoint of the region. Once this is done, the 1D
form of the Chebyshev distribution remapping will occur parallel to the radial axis, based on the radial positions of the elements. The equation for the virtual position $\vec{v}x_n$ of a node $n$ is

$$\vec{v}x_n = r_{edge} \left( 1 - \arccos \left( \frac{\|\vec{r}_{nmid}\|}{r_{edge}} \right) / \frac{\pi}{2} \right) \frac{\vec{r}_{nmid}}{\|\vec{r}_{nmid}\|} + \vec{x}_{mid}$$

(3.9)

where $\vec{x}_{mid}$ is the position of the midpoint of the region, $\vec{r}_{nmid}$ is the displacement of the position of node $n$ from the midpoint, and $r_{edge}$ is the distance from the midpoint to the edge of the region.

While this is sufficient to generate a Chebyshev-like distribution across a circular region, it can be further extended to more general regions, such as a slight or strong deformation of a circular region. The key to achieving this is the recognition that the Chebyshev distribution relies on the distance between the midpoint and the edges of the region, that is, the maximum position on the radial axis that falls within the region. The formula used for a circular distribution relies on the assumption that this value, $r_{edge}$, is constant across every position on the azimuthal axis, but they can be generalized by discarding this assumption and allowing every angle its own edge distance. As a result, each robot will have its own idea of how its distribution should work based on its angular position. The formula for this is

$$\vec{v}x_n = r_{nlocaledge} \left( 1 - \arccos \left( \frac{\|\vec{r}_{nmid}\|}{r_{nlocaledge}} \right) / \frac{\pi}{2} \right) \frac{\vec{r}_{nmid}}{\|\vec{r}_{nmid}\|} + \vec{x}_{mid}$$

(3.10)

where $r_{nlocaledge}$ is the length of a line segment between the midpoint and edge of the region, passing through $x_n$.

Since the algorithm relies on locally available information, $\vec{x}_{mid}$ and $r_{nlocaledge}$ are not directly available to $n$ and must be estimated based on information shared by $n$’s neighbors. A method for doing this is presented in the next section.

### 3.3.1 Determining relative positions using chaining

For swarming algorithms that depend solely on local context, a node’s knowledge of its immediate neighbors will be sufficient to determine its behavior. However, in
order for nodes to be able to react to the edges and centroids of swarms as our algorithm requires, they must have access to a greater range of information. For many nodes, the edges and midpoint will not be directly visible, so they must rely on their neighbors to relay that information to them.

While the centroid of a swarm can be exactly determined by averaging the positions of all the nodes, this method has definite disadvantages when applied in real time. In order for nodes to be aware of each others’ positions, they must be constantly broadcasting their positions across the swarm, as well as relaying the broadcasts sent by other nodes. This requires either an excessive number of broadcasts if messages are immediately relayed or excessively long messages if positions are stored for later broadcast, in an environment where communication windows are likely to be short. An alternative is to generate an approximation of the center, and share that single approximation across the swarm. If that approximation is passed as part of the existing neighbor communications, then no additional messages will be required, and the existing messages will only need to be lengthened by the position of the midpoint.

As part of our solution, we propose to choose one or more of those nodes to represent the center of the swarm. Likewise, nodes on the outside edge can represent points along the edge of the swarm. If a node knows its nearest center representative, and its nearest edge representative, it has an estimate of its distance to the middle and edge of the swarm, and knows how to determine its virtual coordinates. However, a way is needed to determine which nodes to choose and how to propagate those choices without relying on costly global information.

Since there is no centralized authority for the swarm, the points that act as edge representatives are self-selected. The coverage algorithm from Chapter 2 already classifies nodes based on their positions within the swarm, so nodes in the boundary state can be used as representatives.

Once the nodes on the outside edge of the swarm have been identified, a hierarchy of depth can then be established, with the nodes on the outside edge of the swarm having the smallest depth, and the depth increasing as the distance to the edge
increases. The middle of the swarm will then be the set of nodes with the greatest depth. If the edge points are assigned a depth of 0, the depth of every other node can be determined by looking at the depths of all adjacent neighbors, which can be sent as part of their messages, and choosing a depth that is 1 greater than the shallowest adjacent depth seen. The closest edge node can be propagated through this method as well, with every node choosing its closest edge to match the closest edge chosen by the closest shallow neighbor.

After the depth information has had enough cycles to propagate from the outside edges to the rest of the nodes in the swarm, there will be a set of nodes that cannot see any nodes deeper than themselves, though they may see other nodes at the same depth. These nodes select themselves as deep representatives and estimate the midpoint of the swarm as the mean of the positions of themselves and all other visible neighbors of the same depth. They pass their information back towards the outside in a similar manner as the shallow representatives, with other nodes looking for neighbors that are deeper than them, and storing and passing on the position of the deep representative sent by the closest deeper neighbor.

One possibility for determining neighbor adjacency is to simply declare all nodes within single-hop communication range to be adjacent, but this does not guarantee that the depth estimates generated in this manner will have sufficient resolution to be useful. For example, if the radios of the nodes are sufficiently powerful that nodes can communicate with others at least halfway across the swarm, then every inside node will have a hop count of 1, which would render the algorithm useless. Therefore, to ensure that the structures formed this way are reasonable, the algorithm sets limits on the maximum distances other nodes can have and be considered adjacent. When searching for shallower neighbors, the adjacency distance is set to the control parameter $R_{shal} = 2$ times the local spacing of the node. When searching for deeper neighbors, the adjacency distance is set to the control parameter $R_{deep} = 3$ times the local spacing of the node, in order to minimize the occurrence of cases where a deeper node exists but is outside of the adjacency range. While the presence of an unusually close neighbor can
cause this adjustment to fail, these cases are temporary due to the repulsive effects of the swarming algorithm and do not appear to last long enough to significantly distort the behavior of the swarm. Being based on neighbor distance rather than a fixed parameter, this allows the algorithm to automatically adjust itself to varying node densities, including those produced through the Chebyshev-like distribution.

In our system, nodes effectively group themselves into two sets of trees. One set leads from the nodes on the edge and specifies which other nodes will use those nodes’ points to represent their closest edge. Another set leads from the deepest nodes and specifies which other nodes will use those nodes’ points to represent the center of the swarm. Both of these trees are able to handle changes in the structure of the swarm; if an edge node ceases to be on the edge, it will adjust its depth to 1 in the next cycle based on its visibility of other edge nodes, and any nodes that took their depth from it will either increase theirs as well or switch to another edge node as a source. If a central node reduces its depth or a deeper neighbor appears, it will choose a new neighbor as its center representative in the next cycle, and begin to relay that node’s position instead of its own as the deepest node it knows. An example of the edge and deep trees formed by a single set of nodes can be seen in Fig. 3.2.

All of this can be done using the local communication already used for determining the positions of neighbors. The messages just need to have the node’s hop count, the last known position of the edge representative, the last known position of the center representative, and the hop count of the deep representative added on, which is not excessive.

One interesting effect of the algorithm worth noting is that while it will produce representative midpoints near the true centroid of the region for circular regions, this is not always the case. For some region shapes, there may be multiple areas that are equidistant from the edge enough that there will be multiple midpoints, or even a band of midpoints. While this algorithm will not return the true centroid of the region, this is not necessarily a bad thing in many cases, as such positions may correspond to local peaks or other significant features that dominate the shape of the field near them. The
algorithm may therefore serve to handle those types of regions more accurately than a global average would. While more detailed testing remains to be done, the results would likely resemble those found in [16], which utilizes a similar method for detecting edge distance.

3.4 Alterations to Uniform Algorithm

The algorithm used for virtual coordinate-based swarming coverage is similar to the one used for real coordinates in Chapter 2, with a few noteworthy modifications.

One change is the addition of the procedure to update the position of a node in virtual space, given in Procedure 13, into the main loop. The new main loop is presented in Procedure 12.

Another is the modification of the algorithm to swarm based on virtual coordinates rather than real ones. The nodes now determine their velocities based on virtual coordinates, but still base their movement and stability on their real positions. In addition, communication ranges are also still based on real distances.
In general, this involves replacing instances of real positions $\vec{x}_n$ with virtual positions $\vec{v}_n$, as well as real displacements $\vec{r}_{nm}$ with the virtual displacements $\vec{v}_{nm}$. The exception to this is the sensing range condition $\vec{r}_{nm} < s_n R_s$, which is left in real coordinates so as to represent physical limitations in communication ranges (though it is modified in a different way, which will be described later). $\vec{v}_n$ is left untouched, meaning that behavioral decisions will be based on virtual coordinates, but the actual action of movement is left in normal space. The procedures EvolveNode, GetPerpendicularStability, and GetParallelStability are also left out, since those parts of the algorithm also depend on the physical motion of a node.

Another change involves the maximum ranges over which nodes are allowed to sense each other and communicate. Rather than having nodes search for neighbors within a range based on a factor of their spacing, nodes instead broadcast to other nearby nodes within range based on a factor of their spacing. This means that sparsely-spaced nodes will be able to be seen over a comparably longer range, preventing issues where a tight cluster of nodes becomes trapped in a state where their range to too short to view anything other than themselves.

This is necessary due to how virtual positions are determined. In the non-virtual form, such clusters can form but are essentially harmless. If a cluster sees itself as isolated, it will start behaving as if it was its own full swarm, rapidly expanding out its boundary nodes and becoming more spread out until eventually the rest of the swarm is visible, at which point the boundary nodes will simply demote themselves and the swarm will behave as if there was never any issue.

However, when working with virtual coordinates, accurate identification becomes critical. If a node in the middle of the swarm incorrectly identifies itself as being on the boundary, it will begin to serve as an anchor for boundary positions and hop counts, leading the node and its neighbors to miscalculate what their virtual positions should be relative to the rest of the swarm. Other nodes will see those false positions and try to react to them, creating further disruptions. In some cases, this can lead to the swarm destabilizing.
By allowing more distant nodes to be detected by those tight clusters, the algorithm ensures that they’ll be aware that they are not actually on the boundary, and give them accurate positions to align themselves against.

In effect, neighbor visibility is no longer based on sensing power, but instead on broadcasting power, with more widely spaced nodes communicating more loudly. This is implemented in the algorithm by replacing all instances of $\vec{r}_{nm} < s_{n}R_{s}$ with $\vec{r}_{nm} < s_{m}R_{s}$.

### 3.5 Experiments

As in Chapter 2, we implemented the swarming algorithm in Matlab. We update positions using a forward Euler scheme with a small enough timestep to resolve the calculation. All nodes were updated simultaneously. For simplicity, communication between nodes was assumed to be both instantaneous and perfectly accurate.

Unless otherwise indicated, the following parameters are used in each run. The nodes are initially randomly placed within a circle of radius 250, centered at the origin. The duration of each run is 10 time units, with a time step of $\Delta t = 0.05$. The default parameters used across all experiments are given in Table 3.1.

#### 3.5.1 Achieving Chebyshev Distribution

In order to demonstrate the ability of the algorithm to accurately generate both uniform and 2D Chebyshev formations, the algorithm was run on a circular domain, with field strength given by

$$F(x, y) = \exp \left[ -\left( \frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right],$$

where $(x, y)$ are the coordinates of a point within the field centered on $(0, 0)$ and $\sigma_x = 500$ and $\sigma_y = 500$ are the standard deviations of the Gaussian. Two runs were

<table>
<thead>
<tr>
<th>$A_I$</th>
<th>$k_I$</th>
<th>$A_B$</th>
<th>$k_B$</th>
<th>$v_B$</th>
<th>$v_{max}$</th>
<th>$\theta_{min}$</th>
<th>$\alpha_{max}$</th>
<th>$R_s$</th>
<th>$R_{shal}$</th>
<th>$R_{deep}$</th>
<th>$s_{max}$</th>
<th>$D_{dist}$</th>
<th>$F_B$</th>
<th>$\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2</td>
<td>20</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0.75\pi</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1.2</td>
<td>2.5</td>
<td>0.5</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3.1: Default Experiment Parameters
**Procedure 12** Algorithm for node behavior - Virtual Coordinates

1: **ON INITIALIZATION:**
2:  Set node state to interior
3:  Set node spacing to maximum possible sensing range
4:  Set node depth to 0 and set interior and boundary estimates to current node position \{Requires multiple cycles for information to become accurate\}
5:  Immediately trigger update event
6: 
7: **ON UPDATE EVENT:**
8:  **if** \( n \) has received a promotion request since the last update **then**
9:     Set \( n \)'s state to promoted
10: **end if**
11:  \{Update node state\}
12:  Run GetNodeSpacings
13:  Run GetNodeBoundaryState
14:  Run UpdateNodeVirtualPositions
15:  Run UpdateNodeState
16:  Run GetNodeSpacings
17:  \{Update node velocity\}
18:  **switch** (State of node)
19:    **case** interior:
20:       Run GetInteriorInteractions
21:    **case** boundary:
22:       Run GetNodeBoundaryNeighbors
23:       Run GetBoundaryInteractions
24:       Run GetParallelStability
25:       Run GetPerpendicularStability
26:       **if** \( \alpha_n^\perp \) is true **then**
27:           Set \( \vec{v}_n \) to \( \vec{v}_B^\perp \)
28:       **end if**
29:    **case** promoted:
30:       Run GetPromotedInteractions
31:    **case** demoted:
32:       Run GetDemotedInteractions
33: **end switch**
34:  **if** \( ||\vec{v}_n|| > \frac{\omega_n}{\tau} v_{max} \) **then**
35:      Rescale \( \vec{v}_n \) so that \( ||\vec{v}_n|| = \frac{\omega_n}{\tau} v_{max} \)
36:  **end if**
37:  Broadcast current position and state of node to neighbors
38:  Broadcast node depth, estimated interior position, and estimated boundary position to neighbors
39:  Begin moving at velocity \( \vec{v}_n \)
40:  Set timer to trigger update event in \( \Delta t \) time units
Procedure 13 UpdateNodeVirtualPositions - Part 1

**Input:** The current state of node $n$

**Output:** The updated state of node $n$, with $\vec{v}x_n$ containing updated virtual coordinates, $\vec{b}x_n$ containing the estimated position of the nearest point on the boundary, $\vec{i}x_n$ containing the estimated position of the deepest point in the interior, $h_n$ containing the number of neighbor connections between the node and the closest boundary, and $hmax_n$ containing the number of connections between the deepest point and the boundary.

1: {Build connection to the boundary}
2: Set temporary variable $N_n$ to the set of all nodes $m$ where $s_m R_{shal} < r_{nm}$
3: if $\gamma_n$ is boundary or $N_n$ is an empty set then
4: {Node is located on the boundary of the swarm, so use own position as the boundary position and set hop count to 0}
5: Set $h_n$ to 0
6: Set temporary variable $d_{min}$ to $\vec{x}_n$
7: else
8: {Find the nearest shallowest neighbor within range, and set the boundary position to that neighbor’s boundary position and the hop count to 1 greater than the neighbor’s hop count}
9: Set temporary variable $sh$ to $+\infty$
10: Set temporary variable $d_{min}$ to 0
11: for all nodes $m \in N_n$ do
12: if $h_m < sh$ then
13: Set $sh$ to $h_m$
14: Set $d_{min}$ to $||\vec{r}_{nm}||$
15: Set $h_n$ to $h_m + 1$
16: Set $\vec{b}x_n$ to $\vec{b}x_m$
17: else if $h_m = sh$ and $||\vec{r}_{nm}|| < d_{min}$ then
18: Set $d_{min}$ to $||\vec{r}_{nm}||$
19: Set $\vec{b}x_n$ to $\vec{b}x_m$
20: end if
21: end for
22: end if
Procedure 14 UpdateNodeVirtualPositions - Part 2

1: {Build connection to the midpoint}
2: Set temporary variable \( N_n \) to the set of all nodes \( m \) where \( s_m R_{deep} < r_{nm} \)
3: Set temporary variable \( h_{max} \) to \( h_n \)
4: Set \( \tilde{x}_n \) to \( \bar{x}_n \)
5: Set temporary variable \( d_{min} \) to 0
6: for all nodes \( m \in N_n \) do
7: \{Find the nearest deepest neighbor within range, and set the midpoint position to that neighbor’s midpoint position\}
8: if \( h_m > h_{max} \) then
9: Set \( d_{min} \) to \( \| \vec{r}_{nm} \| \)
10: Set \( h_{max} \) to \( h_m \)
11: Set \( \tilde{x}_n \) to \( \tilde{x}_m \)
12: else if \( h_m = h_{max} \) and \( \| \vec{r}_{nm} \| < d_{min} \) then
13: Set \( d_{min} \) to \( \| \vec{r}_{nm} \| \)
14: Set \( \tilde{x}_n \) to \( \tilde{x}_m \)
15: end if
16: end for
17: if \( h_{max} = h_n \) then
18: \{If no deeper neighbors visible, use mean of the positions of equally deep neighbors as midpoint instead\}
19: Set \( \tilde{x}_n \) to the mean position of all nodes \( m \in N_n \) where \( h_m = h_n \)
20: end if
21: {Map real position to virtual position}
22: Set temporary variable \( \vec{r}_{ni} \) to \( \bar{x}_n - \tilde{x}_n \)
23: Set temporary variable \( \vec{r}_{bi} \) to \( \bar{x}_n - \tilde{x}_n \)
24: if \( \| \vec{r}_{ni} \| < \| \vec{r}_{bi} \| \) then
25: Set temporary variable \( \tilde{t}_n \) to \( \vec{r}_{bi} \ast (1 - (\arccos(\| \vec{r}_{ni} \| / 2)) / \pi) \| \vec{r}_{ni} \| / \| \vec{r}_{ni} \| + \tilde{x}_n \)
26: else if \( \| \vec{r}_{ni} \| < 3 \| \vec{r}_{bi} \| \) then
27: Set temporary variable \( \tilde{t}_n \) to \( \vec{r}_{bi} \ast (1 + (\arccos(2 - \| \vec{r}_{ni} \| / \| \vec{r}_{ni} \|) / 2)) \| \vec{r}_{ni} \| / \| \vec{r}_{ni} \| + \tilde{x}_n \)
28: else
29: Set temporary variable \( \tilde{t}_n \) to \( \tilde{x}_n \)
30: end if
31: Set temporary variable \( \vec{t}_{ln} \) to \( \vec{t}_n - v \tilde{x}_n \)
32: if \( \| \vec{t}_{ln} \| > v_{max} s_n \Delta t \) then
33: Rescale \( v \tilde{x}_n \) so that \( \| v \tilde{x}_n \| = v_{max} s_n \Delta t \)
34: end if
35: Set \( \vec{v} \tilde{x}_n \) to \( v \tilde{x}_n + \vec{t}_{ln} \)
done in total, one which used real coordinates to produce a uniform distribution, and one which used virtual coordinates to produce a Chebyshev distribution. A node count of 400 was used for both runs, and the durations of the runs were 10 time units. In order to provide a basis for comparison, sets of ideal distributions were also generated, by means of randomly placing 1000 nodes across the region, with their radial densities either being uniform or remapped to a Chebyshev-like distribution using Equation 3.10.

The distributions were compared by sorting nodes into groups based on their distance from the midpoint of the region at the end of the simulations, with each group representing a ring with a width of 100 units. The percentage of nodes that fell within each of the groups are shown in Figure 3.3, with Figure 3.3a comparing the uniform formation produced by the non-virtual coordinate algorithm with the random uniform distribution, and Figure 3.3b comparing the Chebyshev-like formation produced by the virtual coordinate algorithm with the random Chebyshev-like distribution. In both cases the node distributions were very similar, indicating that the swarming algorithms...
have successfully produced their desired distributions.

### 3.5.2 Interpolation Quality

In order to demonstrate the ability of the algorithm to create formations that accurately measure a region, formations generated by the algorithm were used to generate interpolations estimating the value of positions within the region not covered by the sensors.

To generate the formations used for sampling the regions, both the uniform and Chebyshev versions of the algorithm were run across simple regions with node counts of 100, 200, and 400.

Interpolations were performed on perturbed versions of the fields used for the simulations. The perturbations were generated though constructing 2D fields of normally distributed random noise, with side ranges of -1000 to 1000 distance units, deltas of 10 distance units, and standard deviations of .25. These fields were run through a low-pass filter in the Fourier domain based on cutoff wave numbers $k = [2, 4, 6]$, resulting in perturbation fields of varying complexity [18]. The fields used for interpolation were generated by summing these perturbation fields with the fields used to generate the original formations. In effect, the field strength can be represented as

$$F'(x, y) = F(x, y) + \sum_{m=-k+1}^{k} \sum_{n=-k+1}^{k} a_{mn} \exp(i(\frac{2\pi m}{L} x_i + \frac{2\pi n}{L} y_j)) \quad (3.12)$$

where $F'(x, y)$ is the resulting perturbed field, $F(x, y)$ is the original unperturbed field, $k$ is the maximum wave number used, and $a$ is a set of random normally-distributed values with a mean of 0 and a standard deviation of .25. An example of how perturbations alter the shape of a sensed field is given in Figure 3.4, which shows the base form of a circular Gaussian field along with the effects of perturbing it with different complexities.
3.5.2.1 Circle

In order to demonstrate the ability of the algorithm to accurately estimate the values of measured data across a circular region, the algorithm was first run on a field shaped like a circular 2D Gaussian, with unperturbed field strength given by

\[
F(x, y) = \exp \left[ - \left( \frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right],
\]

where \((x, y)\) are the coordinates of a point within the field centered on \((0, 0)\) and \(\sigma_x = 500\) and \(\sigma_y = 500\) are the standard deviations of the Gaussian.

A series of runs were done in which the number of sensor nodes present was varied between 100, 200, or 400, and the number of elements used for the perturbation was varied between 0 (representing no perturbation), 2, 4, or 6. For each combination of parameters, five pairs were run, with each pair consisting of a run using the uniform algorithm and a run using the Chebyshev algorithm, with both runs being carried out on the same perturbed field. Examples of the final node positions on runs on an unperturbed field are given in Figure 3.5.

The final states of each individual run were analyzed through the radial basis function method presented in Section 3.2, yielding minimal root mean squared (RMS) errors and maximum errors for that run. An example of the interpolation error for a pair of runs with node count of 200 and perturbation cutoff of 4 is given in Figure

Figure 3.4: Effect of perturbation with maximum wave number \(k\) on a circular Gaussian field.
Table 3.2: Error Ratios for Circular Regions

3.6. For each set of five Chebyshev or uniform runs for a given parameter, the RMS and maximum errors for the Chebyshev run were divided by the respective RMS and maximum errors for the uniform run, yielding relative error ratios for each set. The values across the five sets were then averaged in order to yield mean error ratios for each combination of parameters. The results are presented in Table 3.2, with values less than 1 indicating that the Chebyshev interpolations were more accurate.

As is shown in Figure 3.6, interpolations using the distributions generated by the uniform algorithm tended to concentrate error near the edges of the region, consistent with Runge’s phenomenon, while Chebyshev-based interpolations tended to distribute lesser degrees of error throughout the region. In general, the distributions generated by the Chebyshev algorithm resulted in significantly greater interpolation accuracy than those generated by the uniform algorithm, though there were two noteworthy exceptions. The first occurred when there was no perturbation and either 200 or 400 sensors. In those cases, the field was simple enough relative to the number of data points that the data was exhaustively sampled regardless of the formation used. The second occurred when there was a maximum wave number of 6 and 100 nodes. In this case, the field appears to be undersampled by the sensors. As a result, the Chebyshev interpolation was overfitted to measurement noise near the boundary, leading to greater error than the more widely distributed uniform interpolation.

As long as there are enough sensors to accurately capture the overall shape of the variations within the region, Chebyshev formations appear to show superior performance to uniform formations. However, if the sensors are insufficiently dense to generate an accurate interpolation, uniform formations are likely to generate results that are less inaccurate.
Figure 3.5: Formations generated by running algorithm on an unperturbed circular region.
Figure 3.6: Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4

3.5.2.2 Square

In order to test the ability of the swarming algorithm to accurately estimate the values of measured data across regions containing either straight edges or sharp corners, the algorithm was run on a square-shaped region. The field was similar in nature to that of the circular Gaussian, with the exception that the field strength was determined by the greater of the differences in $x$ or $y$ positions from the midpoint of the region, rather than both simultaneously. The equation for the unperturbed field strength is given by

$$F(x, y) = \exp \left[ -\max \left( \frac{x^2}{2\sigma_x^2}, \frac{y^2}{2\sigma_y^2} \right) \right]$$

(3.14)

where $(x, y)$ are the coordinates of a point within the field centered on $(0, 0)$ and $\sigma_x = 500$ and $\sigma_y = 500$ are parameters controlling the width of the Gaussian.

A series of runs were done in which the number of sensor nodes present was varied between 100, 200, or 400, and the number of elements used for the perturbation was varied between 0 (representing no perturbation), 2, 4, or 6. For each combination of parameters, five pairs were run, with each pair consisting of a run using the uniform
algorithm and a run using the Chebyshev algorithm, with both runs being carried out on the same perturbed field. Examples of the final node positions on runs on an unperturbed field are given in Figure 3.7.

The final states of each individual run were analyzed through the radial basis function method presented in Section 3.2, yielding minimal root mean squared (RMS) errors and maximum errors for that run. An example of the interpolation error for a pair of runs with node count of 200 and perturbation cutoff of 4 is given in Figure 3.8. For each set of five Chebyshev or uniform runs for a given parameter, the RMS and maximum errors for the Chebyshev run were divided by the respective RMS and maximum errors for the uniform run, yielding relative error ratios for each set. The values across the five sets were then averaged in order to yield mean error ratios for each combination of parameters. The results are presented in Table 3.3, with values less than 1 indicating that the Chebyshev interpolations were more accurate.

In this case, the nature of the shape of the field played a significant role in the nature of the interpolation errors. It can be seen from Figure 3.7 that the algorithm had difficulty settling the nodes into stable Chebyshev-like distributions, with the corners still actively in motion when the runs terminated. This appears to be related to how the position of the boundary is determined, with the convexity of the corners causing nodes near the corners to see the edges as closer and incorrectly compute their virtual positions. While this may have reduced the accuracy of the Chebyshev distribution, the effect of this on interpolation error was overshadowed by a more significant problem related to the field within the region.

Looking at Figure 3.8, it can be seen that for both Chebyshev and uniform formations, errors tended to cluster around the diagonals between the corners and the center. These diagonals are where the strength of the field transitions between being controlled by the horizontal position or the vertical position, leading to discontinuity in the slope of the field, similar to the edges of a pyramid.

As RBFs are inherently continuous, they appear to have difficulty accurately interpolating such regions. While the interpolations from both the Chebyshev and
uniform distributions were relatively inaccurate compared to those for the circular region, the uniform distributions were generally more accurate than the Chebyshev distributions. This appears to be due to the uniform distributions providing better coverage of the diagonals, with the uniform distributions’ greater errors in the corners being outweighed by the Chebyshev distributions’ greater errors along the interior diagonals.

For regions that contain discontinuities that are difficult to accurately interpolate, it appears that formations that provide better coverage of those discontinuities will generate more accurate interpolations. In this case the uniform formations provided more of that coverage than the Chebyshev formations. A formation that concentrated nodes even more densely along the diagonals might produce even greater accuracy, but such a formation would be difficult to implement by a swarm on an unknown region.

### 3.5.2.3 Multiple peaks

In order to test the ability of the swarming algorithm to accurately estimate the values of measured data across regions containing multiple local maxima, the algorithm was run on a region consisting of a combination of two circular Gaussians. The equation for the unperturbed field strength is given by

\[
F(x, y) = \exp \left[ -\left( \frac{(x - 300)^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right] + \exp \left[ -\left( \frac{(x + 300)^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right] \quad (3.15)
\]

where \((x, y)\) are the coordinates of a point within the field centered on \((0, 0)\) and \(\sigma_x = 200\) and \(\sigma_y = 200\) are parameters controlling the width of the Gaussians.

<table>
<thead>
<tr>
<th>Wave Cutoff</th>
<th>RMS Error</th>
<th>MAX Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 sensors</td>
<td>200 sensors</td>
</tr>
<tr>
<td>0</td>
<td>1.4137</td>
<td>1.2758</td>
</tr>
<tr>
<td>2</td>
<td>1.2281</td>
<td>1.1240</td>
</tr>
<tr>
<td>4</td>
<td>1.0963</td>
<td>1.2460</td>
</tr>
<tr>
<td>6</td>
<td>1.2069</td>
<td>1.1332</td>
</tr>
</tbody>
</table>

Table 3.3: Error Ratios for Square Regions
Figure 3.7: Formations generated by running algorithm on a unperturbed square region.
Figure 3.8: Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4

A series of runs were done in which the number of sensor nodes present was varied between 100, 200, or 400, and the number of elements used for the perturbation was varied between 0 (representing no perturbation), 2, 4, or 6. For each combination of parameters, five pairs were run, with each pair consisting of a run using the uniform algorithm and a run using the Chebyshev algorithm, with both runs being carried out on the same perturbed field. Examples of the final node positions on runs on an unperturbed field are given in Figure 3.9

The final states of each individual run were analyzed through the radial basis function method presented in Section 3.2, yielding minimal root mean squared (RMS) errors and maximum errors for that run. An example of the interpolation error for a pair of runs with node count of 200 and perturbation cutoff of 4 is given in Figure 3.10. For each set of five Chebyshev or uniform runs for a given parameter, the RMS and maximum errors for the Chebyshev run were divided by the respective RMS and maximum errors for the uniform run, yielding relative error ratios for each set. The values across the five sets were then averaged in order to yield mean error ratios for
each combination of parameters. The results are presented in Table 3.4, with values less than 1 indicating that the Chebyshev interpolations were more accurate.

The results for the multiple peak regions are very similar to the ones for the circular regions. As before, the uniform formations concentrate error near the boundaries of the region, while the Chebyshev formations distribute it more evenly. The error ratios for the cases with no perturbation are again irregular, which is likely an artifact of how inherently accurate they are. The rest of the results show superior accuracy for Chebyshev formations, with the exception of the cases with 100 nodes and a maximum wave number of 6, where the region begins to be undersampled.

From these results, it appears that mild curvature, whether concave or convex, does not significantly affect the ability of swarm formations to accurately interpolate a region, or the relative dominance of Chebyshev formations for doing so.

### 3.5.2.4 Crescent

In order to test the ability of the swarming algorithm to accurately estimate the values of measured data across regions containing high convexity, the algorithm was run on a crescent-shaped region. The field was composed of a combination of two circular Gaussians, with a wider curve providing the basic body of the region and a narrower negative curve removing a smaller section. The equation for the unperturbed field strength is given by

$$F(x, y) = \exp \left[ - \left( \frac{x^2}{2\sigma_x^2} + \frac{(y - 100)^2}{2\sigma_y^2} \right) \right] - 2 \exp \left[ - \left( \frac{x^2}{2\sigma_x'^2} + \frac{(y - 300)^2}{2\sigma_y'^2} \right) \right]$$

(3.16)

where $\sigma_x = 500$ and $\sigma_y = 500$ are parameters controlling the width of the region, and $\sigma_x' = 175$ and $\sigma_y' = 175$ control the width of the removed area.

Table 3.4: Error Ratios for Multiple Peak Regions
Figure 3.9: Formations generated by running algorithm on an unperturbed multiple peak region.
A series of runs were done in which the number of sensor nodes present was varied between 100, 200, or 400, and the number of elements used for the perturbation was varied between 0 (representing no perturbation), 2, 4, or 6. For each combination of parameters, five pairs were run, with each pair consisting of a run using the uniform algorithm and a run using the Chebyshev algorithm, with both runs being carried out on the same perturbed field. Examples of the final node positions on runs on an unperturbed field are given in Figure 3.11.

The final states of each individual run were analyzed through the radial basis function method presented in Section 3.2, yielding minimal root mean squared (RMS) errors and maximum errors for that run. An example of the interpolation error for a pair of runs with node count of 200 and perturbation cutoff of 4 is given in Figure 3.12. For each set of five Chebyshev or uniform runs for a given parameter, the RMS and maximum errors for the Chebyshev run were divided by the respective RMS and maximum errors for the uniform run, yielding relative error ratios for each set. The values across the five sets were then averaged in order to yield mean error ratios for
Table 3.5: Error Ratios for Crescent Regions

<table>
<thead>
<tr>
<th>Wave Cutoff</th>
<th>RMS Error</th>
<th>MAX Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 sensors</td>
<td>200 sensors</td>
</tr>
<tr>
<td>0</td>
<td>1.2276</td>
<td>0.7074</td>
</tr>
<tr>
<td>2</td>
<td>1.4428</td>
<td>0.3761</td>
</tr>
<tr>
<td>4</td>
<td>1.0385</td>
<td>0.7236</td>
</tr>
<tr>
<td>6</td>
<td>1.6952</td>
<td>0.7416</td>
</tr>
</tbody>
</table>

The results are presented in Table 3.5, with values less than 1 indicating that the Chebyshev interpolations were more accurate.

It is notable that for cases involving 100 sensors, the Chebyshev algorithm had difficulty achieving accurate coverage of the region of interest, never managing to achieve a stable Chebyshev formation. As Figure 3.11(b) shows, the formations at run termination tended to leave some nodes sitting outside of the region, while the interior fails to fill out the horns of the crescent. This appears to be due to an inherent weakness of the virtual coordinate algorithm, with the strong convexity of the horns of the crescent interfering with the ability to form stable pathways between the boundary and interior, leading to the calculated virtual positions of the sensors becoming unstable.

As a result, the interpolations generated by the Chebyshev formations with 100 nodes were significantly inferior to the uniform interpolations. As the node count increases to 200 or 400, the Chebyshev algorithm is able to settle the swarms into the desired formations, and the relative errors become similar to those seen for the circular and multiple peak regions, favoring Chebyshev distributions instead.

This shows that while Chebyshev distributions are likely to be superior in many cases, the choice of algorithm also needs to be somewhat tailored to the region. For regions with relatively extreme boundary convexity, a virtual coordinate algorithm may fail to produce ideal formations, making a uniform algorithm the superior choice. For regions with lower convexity, or where the sensor count is sufficient to provide significant depth, Chebyshev distributions will likely remain superior.
Figure 3.11: Formations generated by running algorithm on a unperturbed crescent-shaped region.
Figure 3.12: Interpolation errors for a pair of Chebyshev and uniform runs with 200 nodes and a perturbation cutoff of 4

3.6 Conclusions

In this chapter, we have presented a swarming algorithm designed to position a swarm of sensor nodes into a Chebyshev-like formation across an unknown region of interest. This algorithm is built as an extension of a swarming algorithm that yields uniform distributions across such a region of interest, modifying it so that node interactions are based on virtual positions rather than real ones. These virtual positions are based off of deformations of space within the region such that a Chebyshev-like formation in real space appears uniform in virtual space. By operating the uniform algorithm in virtual space, Chebyshev-like formations will thus be generated in real space.

As part of the virtual coordinate algorithm, we have developed a technique for allowing nodes within a swarm to determine their relative positions within that swarm based only on limited-range communications. By building trees of pathways between the boundary and center of the swarm based on the minimal number of connections needed to reach the boundary, and sharing information about the positions of the
boundary and center along those trees, nodes can gain an estimate of the positions of the closest points on the boundary and the center, and thus an estimate of their own relative position within a swarm.

We hypothesized that formations generated by our Chebyshev swarming algorithm would have superior interpolation accuracy over formations over our earlier uniform swarming algorithm, by minimizing the effect of Runge’s phenomenon. We tested this hypothesis by running both swarming algorithms on a variety of base region shapes and a range of perturbations, and using the positions of the nodes at the end of the runs to generate interpolations of the entire region using radial basis functions.

For the majority of the experiments, the hypothesis held, with the interpolations produced by the Chebyshev swarming algorithm yielding lower errors than the uniform interpolations. Notable exceptions occurred where complexity was focused in smaller areas such as first-order cusps on corners, or where the general perturbations were complex relative to the number of sample points. This indicates that while the Chebyshev regions were indeed minimizing Runge’s phenomenon, this may not be the primary source of interpolation error in all cases. For cases where the sensor count is small relative to the expected complexity of the region, or where complexity is expected to be focused on smaller sections of the region as a whole, then the Chebyshev swarming algorithm might not be ideal.

The Chebyshev swarming algorithm has also been observed to have difficulty achieving stability in some cases, more so than the similar uniform swarming algorithm. Node motion will cause pathways to boundary and interior to change, but changes in pathways will cause a node’s estimate of its virtual position to change as well, sometimes moving it a significant distance in virtual space and causing further changes in node interactions. For many cases this will only slow down achieving stability, but for cases with low node counts and high concavities achieving stability may be impossible. Again, Chebyshev distributions may not be ideal for cases where fewer sensors are present in a swarm.
Chapter 4

CONCLUSIONS

4.1 Summary

In this dissertation, we have presented the problem of gathering data across large regions of interest such as fires and oil spills that are large in size, have shapes that are initially unknown or may be changing over time, and may be too dangerous to be directly measured by humans. We described the technique of swarm interpolation, which involves using swarms of mobile sensors to gather data at various points across the region without needing human intervention, and interpolating the measured data to produce a view of the region as a whole.

In Chapter 2, we presented an algorithm designed to achieve uniform coverage of an unknown region of interest with a swarm of mobile sensors. The algorithm functions by dividing the sensors into two groups. One group seeks out the boundary of the region, and distributes itself uniformly along it. The second group spreads itself out uniformly within the area of the region, with repulsive interactions with the boundary group preventing it from spreading outside the region. Uniformity within the swarm as a whole is controlled by exchanging sensors between the two groups as necessary, and calibrating the strength of the boundary interactions.

We carried out a detailed analysis of the algorithm, showing the influence that various parameters have on its behavior. We demonstrated the ability of the algorithm to achieve coverage across a variety of regions of interest, including ones that translated or deformed over time.

In Chapter 3, we presented an evolution of the algorithm from Chapter 2, expanding it to achieve coverage of unknown regions with non-uniform distributions of...
sensors. We achieved this through mapping the positions of the sensors into virtual coordinates, and using an adapted version of the uniform algorithm on those virtual coordinates rather than the real ones, encouraging the sensors to settle into formations that appear uniform in virtual space but achieve some other distribution in real space, based on the mapping function used. We adapted the concept of a distribution based on the roots of a Chebyshev polynomial into two dimensions, and applied this technique to build an algorithm that generates 2D Chebyshev-like sensor formations.

As part of our algorithm for mapping the positions of a swarm of sensors into virtual coordinates, we developed a novel technique for determining the relative position of a node within a large swarm, based entirely on locally available information and communications. This is done through building trees of connections to the boundary and middle of the swarm, and passing information about points on the closest boundary and the middle along those trees. This allows each individual node to obtain an estimate of the distance to the closest edge and the center of the swarm, enabling them to estimate their relative positions within it.

We experimentally confirmed that both the uniform and Chebyshev swarming algorithms produce formations that closely correspond to their desired distributions. We also analyzed the capabilities of both of the algorithms to produce accurate interpolations of unknown regions, using a series of experiments that varied both the shape and complexity of the sampled data within the regions. We demonstrated that in the majority of cases the Chebyshev algorithm generated higher-quality interpolations, though the uniform algorithm was superior under certain circumstances.

4.2 Future Work

Our future work could take a variety of paths.

While our algorithms are intended to eventually be applied to real-world applications, up until this point all of our work has been carried out on a highly abstract level. All communications between sensors are assumed to be instantaneous, with no
limitations on range beyond fixed cutoffs and no chance of communication interference. All position and sensor readings are assumed to be completely accurate. All of the parameters are based on arbitrary units, rather than considering the limitations of a physical mobile sensor. The energy costs of sensing, communication, and mobility are not considered. While those elements may not be necessary for establishing higher-level control, if the algorithm is to be implemented in a swarm of physical sensors they must be taken into account.

As our algorithms involve simultaneous interactions among hundreds of nodes, testing it on a physical swarm may be difficult or impossible at this point in time, though this may change as the field of robotics becomes more mature. Other works [14] [25] have involved tests on less than a dozen nodes, but our algorithms are incapable of scaling down to that level, requiring roughly four layers of node depth in order to balance out interactions and provide space for node demotion. A viable alternative would be to implement the algorithms on a simulation platform containing more realistic robotic behavior and communication models.

More analysis could be performed on the methods used for interpolation. Our analysis has involved determining the optimal interpolation parameters for each individual instance, but we have not yet done any work on generalizing those parameters for realistic cases where lack of information makes the true error uncomputable. In order to accurately interpolate unknown regions, the ideal parameters must be estimated in advance, based entirely on information that can be known such as node counts and positions. Our analysis has also been limited to interpolation based on a single category of radial basis functions, and on fixed shape parameters. As described by Fornberg [7], and Driscoll [5], using variable shape parameters could result in higher overall accuracy while still minimizing Runge’s Phenomenon. Other types of radial basis functions, or other interpolation methods such as Delaunay triangulation, may also possibly yield more accurate interpolations.

Work could also be done on further improving the swarming algorithms. While the uniform swarming algorithm has been shown to elegantly handle many types of
regions, there are cases where it has difficulty reaching stability. Of particular note are the edge interactions within heavily convex areas, such as the horns of a crescent, being undesirably weak, and nodes having issues completing demotions in areas where the distance between the boundary and the middle is too shallow. The algorithm might be significantly improved by either detecting and handling such situations, or by altering it in a more fundamental way so that they become less of an issue.

Another issue with the uniform swarming algorithm is related to how it performs length scaling on a local level. While this behavior allows the algorithm to adapt its behavior to regions of any size, it leads to issues when part of the swarm is being compressed, such as when the boundary of the swarm is moving inwards. When a region is compressed, nodes should react by rapidly expanding out of that region into more sparse space. However, as nodes are forced together, their estimates of the length scale decrease, causing them to move more slowly than the rest of the swarm, delaying the dissipation of the compression. If an edge of the swarm is moving rapidly, this can lead to large groups of nodes piling up along that edge. One way to resolve this issue might be to build in a mechanism to detect density differentials and push nodes out of denser areas more quickly. Another solution might be to change the nature of the swarm model to incorporate a velocity matching component. If nodes attempt to match the velocity of their neighbors, then nodes can incorporate the motion of the swarm into their behavior. This allows them to more smoothly adapt to the motion of a nearby boundary, rather than resisting that motion and needing to be forced out through compression.

Being based directly on the uniform swarming algorithm, the Chebyshev swarming algorithm inherits all of its issues, while developing issues of its own. One significant issue that it faces is related to how nodes determine their virtual positions. Since information about the boundary and interior is gathered based on dynamically-generated trees, every time node motion causes the trees to be rebuilt, and the virtual positions of any nodes involved in the change will be altered as well, sometimes causing large jumps in estimated virtual positions. This causes a cascading effect on the neighbors
of moving nodes, delaying or even preventing the swarm from achieving stability.

Resolving this would require changes in how information is shared throughout the swarm, so that minor changes cannot cascade into stronger effects. One way to do this might be to replace the strict tree structures with a fuzzy web of connections, allowing nodes to be influenced by information from multiple neighbors. Minor changes in node positions would only result in minor changes in weights, preventing virtual positions from suddenly jumping. Another possibility could be to build pathways based on a system inspired by slime mold growth, similar to the methods developed by Tero [22] and Li [15].


