2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems

# Tracking Time-Dependent Scalar Fields with Swarms of Mobile Sensors

Joshua Kirby* Marco A. Montes de Oca† Steven Senger† Louis F. Rossi† Chien-Chung Shen*
*Department of Computer and Information Sciences †Department of Mathematical Sciences
University of Delaware, Newark, DE, USA
{kirby,cshen}@cis.udel.edu, {mmontes,senger,rossi}@math.udel.edu

***Abstract*—In previous work, we introduced a novel swarming interpolation framework and validated its effectiveness on static fields. In this paper, we show that a slightly revised version of this framework is able to track fields that translate, rotate, or expand over time, enabling interpolation of both static and dynamic fields. Our framework can be used to control autonomous mobile sensors into flexible spatial arrangements in order to interpolate values of a field in an unknown region. The key advantage to this framework is that the stable sensor distribution can be chosen to resemble a Chebyshev distribution, which can be optimal for certain ideal geometries.**

## I. Introduction

Autonomous monitoring of terrestrial, maritime and atmospheric fields requires the coordination of sensor-equipped mobile nodes. Many tasks, such as monitoring the extent of the Gulf Oil Spill, demand the deployment of large numbers of relatively simple but coordinated devices (i.e., a *swarm*) rather than the use of small numbers of sophisticated devices (e,g, [4]). While there have been many efforts dedicated to determining the boundary of a measured field, it is often the case that investigators need to know the full field, not just the position of the boundary. We refer to this challenge as the *swarm interpolation problem*. In previous work [8], we introduced a framework that allows mobile sensors to distribute themselves into special configurations that let them accurately interpolate the whole field in which they are deployed.

Most previous work (Section II) has been based on the assumption that the field in which the sensors are deployed is static, that is, that it does not change over time. This assumption is valid if the monitored field does not change spatially. For example, if one is interested in monitoring temperature over a specific geographic region, then there is no need to change the location of the sensors. However, if the field's changes are spatial, then the sensors must move with the field in order to track the changes the field is experiencing.

In this paper, we present a study of the tracking ability of a slight modification of the framework introduced in [8] (Sections III–VIII). We also present an analysis of the spatial distributions obtained with this framework when the sensors reach an equilibrium. Together, these results show that interpolation of both static and dynamic fields is possible with swarms of mobile sensors. However, there is no free lunch. Our results suggest that as the swarm size increases, that is, as the accuracy of the interpolation is increased, the tracking performance decreases as measured by the speed at which the equilibrium state is reached. Thus, with fixed size swarms there is a trade-off between interpolation accuracy and speed.

## II. Related Work

While searching for the location of a contaminant field and identifying points of high concentration within that field is a relatively simple and well-studied problem, mapping out the interior of such a field, the swarm interpolation problem, is more challenging. Rather than locating one or more peak points, mapping the interior of a field requires that data be gathered at a variety of positions throughout it, with enough resolution and coverage to make a reasonable estimate of the field as a whole. One approach consists in adapting a population-based search algorithm in order to record measurements along the search paths, and use the recorded measurements to interpolate the field. Turduev [13] et. al. take this approach. Their algorithm, based on particle swarm optimization (PSO) [5], uses a group of nodes to search for areas of peak concentration within a field. Nodes calculate their movement in steps, using the positions of the highest values found by themselves and their neighbors during the previous step to determine their next destination. While the goal of the algorithm is for nodes to settle on peak values of the field, the algorithm also records the sensed values at each step, and uses those values to compute a regression for the field. However, since the PSO algorithm is built for optimization and not for full coverage, there is no guarantee that all significant regions within the field will be represented by the gathered data. Additionally, the data is gathered over time, which means that if the field is changing, then the data will not accurately represent the field at any given point in time.

One way to ensure that the field is fully represented by the data is to employ a coverage algorithm to distribute sensor nodes throughout the region. Cortes [3] presents an algorithm that divides the region into Voronoi partitions based on the current positions of the sensors, and moves the sensors in order to equalize the area covered by the regions. Krause [9] presents an algorithm that distributes sensors based on the concept of mutual information. However, while these algorithms succeed in spreading nodes throughout a region, they are optimized for detecting events within the field and not for gathering data at the locations of the nodes. Moreover, these algorithms rely on the structure of a field being known in advance, rather than adapting to the field on the fly, as would be necessary for a survey of an unknown region.

A third approach consists in dynamically adapting the positions of a group of nodes to fit the shape of the monitored region. An example of this approach is given by Bertozzi et al. [2], who propose an algorithm for tracking the edge of a region by treating the nodes as points along a contour, and

978-0-7695-5129-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SASO.2013.37

evolving those points using a discretization of an image snake algorithm to fit the shape of the contaminated region. This method has the advantage of being able to adapt to a variety of shapes without knowing any information about the area in advance. Unfortunately, it places all of the nodes on the outside edge of the region, limiting the information that can be gained about the interior.

Kalantar [7] manages to solve the aforementioned issue by dividing the sensor nodes into two groups, a boundary set that defines the edge of the region of interest and an interior set that fills out the inside, with a node's set membership being self-determined based on the alignment of its neighbors. Node on the boundary act to position themselves along the desired edge through gradient descent, while nodes on the inside maintain a uniform distribution by being repelled by each other and the boundary. This places all of the nodes in positions where they will be useful for sensing information about the field, and is capable of adapting itself to a variety of shapes.

## III. Chebyshev Distributions and Interpolation

Our work may be seen as an evolution of Kalantar's work. We maintain the basic concept of boundary and interior sets, but focus more on applying additional controls to the behavior of the nodes in the interior set. While a uniform interior is sufficient to reasonably generate a regression of a field, we have shown evidence in [8] in support of the idea that using a non-uniform distribution can improve the predictive accuracy of such a model.

Technically speaking, the problem we tackle with our framework is called *scattered data interpolation* [6]. In a nutshell, given an input set $\{(\boldsymbol{x}_1, \phi_1), (\boldsymbol{x}_2, \phi_2), \ldots, (\boldsymbol{x}_N, \phi_N)\}$, where $N$ is the number of sensors, $\boldsymbol{x}_n \in \mathbb{R}^2$ represents the location of the $n$th sensor, and $\phi_n = f(\boldsymbol{x}_n)$ 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 $g(\boldsymbol{x}) = \phi$ and that the difference between $g$ and $f$ at locations different from $\boldsymbol{x}_n$, $n = 1, \ldots, N$ 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 interpolants, but these produce large oscillations, and therefore large errors, near the boundary of the region (this is known as Runge's phenomenon). One effective distribution is based on the roots of a Chebyshev polynomial. (See [1], [12], and the references contained therein for a general discussion.) The following system yields the roots of the desired Chebyshev polynomial.

$$\begin{aligned} T_0(x) &= 1, \qquad T_1(x) = x, \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \quad (n \geq 1) \end{aligned} \tag{1}$$

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

$$T_n(x) = \cos(n \cos^{-1} x) \quad (n \geq 0) \tag{2}$$

where $n$ is the desired number of roots for the polynomial. See Fig. 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

$$|f(x) - p(x)| \leq \frac{1}{2^n(n+1)!} \max_{|t| \leq 1} |f^{(n+1)}(t)|, \tag{3}$$

where $f(x)$ is the function being interpolated and $p(x)$ is the interpolation polynomial based on the Chebyshev roots. This distribution is optimal for polynomial interpolation. The distribution can be further extended from 1D into a 2D circle. One example is shown in Fig. 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. Typical swarming algorithms will produce meshless uniform distributions of sensors, so in order to achieve a Chebyshev distribution, a special form of swarming will need to be performed, one that is sensitive to the positions of the sensors within the swarm, not just the relative positions of their neighbors. We finesse a standard swarming 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 is extremely close to an appropriate Chebyshev distribution as they swarm.

In this paper, 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.

## IV. The Swarm Interaction Model

The swarming algorithm uses a basic control model

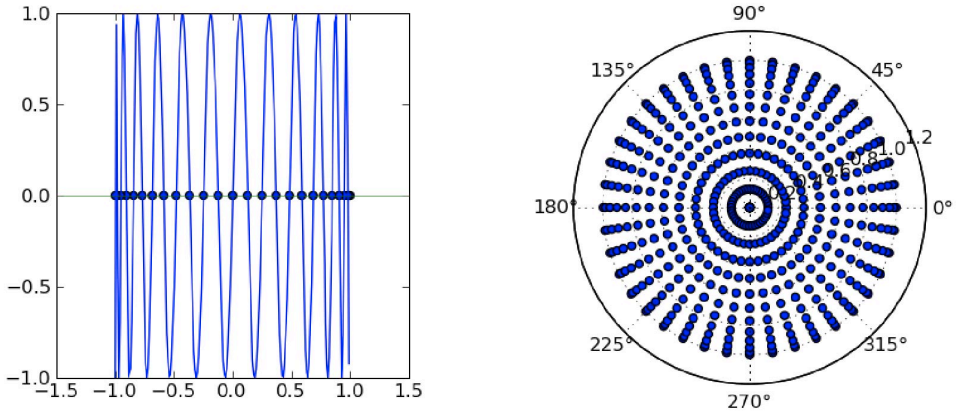
$$\frac{d}{dt}\vec{x}_i = \vec{v}_i \tag{4}$$

$$\frac{d}{dt}\vec{v}_i = \kappa(\vec{d}_i - \vec{v}_i) \tag{5}$$

where $\vec{x}_i$ and $\vec{v}_i$ are the position and velocity vectors of the $i^{\text{th}}$ sensor, $\vec{d}_i$ is the desired velocity vector for the sensor and $\kappa$ is a rate constant. The desired velocity is the sum of pairwise interactions from the other sensors in the swarm:

$$\vec{d}_i = \sum_{j \in N_i} \vec{d}_{ij} \tag{6}$$

where $N_i$ is the set of indices of sensors within communication range of sensor $i$. Each interaction between a pair of sensors represents a balance between mutual repulsion and attraction




(a) Chebyshev polynomial with 25 roots (b) Circular mesh-aligned radial Chebyshev distribution

Fig. 1: Chebyshev distribution: 1D vs. 2D

to greater levels of the sensed field. The specific pairwise interaction model is

$$\vec{d}_{ij} = V\left[\left(\frac{\phi(\vec{x}_i)-\phi_\star}{1-\phi_\star}\right)e^{-k_1 r_{ij}} + a\phi_\star e^{-k_2 r_{ij}}\right]\frac{\vec{r}_{ij}}{r_{ij}} \quad (7)$$

where the function $\phi$ is the sensed value at a given position, $V$ is a characteristic velocity scale for the sensors, $\phi_\star$ is the edge threshold for the sensed region, $a \ll 1$ is a small parameter, $1/k_1$ and $1/k_2$ are characteristic length scales, $\vec{r}_{ij} = \vec{x}_i - \vec{x}_j$ and $r_{ij} = \|\vec{r}_{ij}\|$. Furthermore, we assume the sensed function is normalized so that $0 \leq \phi \leq 1$.

The interaction between two given sensors is driven by two competing objectives. First, in order to interpolate across an entire region, the sensors must occupy that region. Given a lack of other cues, a group of sensors located within the sensed region should spread to occupy that region, and so a repulsive element is necessary. However, the sensors should not spread indefinitely, as they ultimately need to align themselves *within* the region, not *outside* of it. A factor is therefore needed to prevent sensors from spreading beyond the edge of their desired region. To allow the sensors to spread across a region while being constrained by its edge, the sensors behavior depends on the difference between their current sensor readings and a set field strength, with the influence felt by the sensors directly proportional to that difference. Rather than adopting a static edge, our solution was to alter the behavior of sensors based on the difference between the field strength they sense at their location and a set field strength that serves as the edge of the region. In the first term of (7), the influence felt by a sensor is set to be directly proportional to that difference. Sensors within the region try to move away from others nearby, and sensors outside the region move toward nearby sensors. The second term of (7) corresponds to a small field-independent repulsive force to prevent sensors on the outside from overly converging and to prevent sensors directly on the edge from becoming completely locked into position.

This model will yield a uniform spread across the region of interest. The modifications necessary to produce a Chebyshev distribution are described in the next section.

## V. Applying Chebyshev Distribution

The presentation that follows assumes the reader is familiar with our work presented in [8]. Here, we focus on the modifications that make tracking possible. The method to change the swarm's equilibrium distribution from uniform to Chebyshev-like is based on the algorithm for yielding a star-like Chebyshev distribution presented in [8]. This algorithm interprets the region of the swarm as a set of line segments extending from the center of the region to the edges of the swarm, and remaps the coordinates along those line segments so that a Chebyshev distribution along one of those segments appears uniform in virtual coordinates, while the distribution perpendicular to the segments remains unchanged between the real and virtual systems.

Following the assumption that nodes do not have immediate access to the positions of all other nodes in a swarm, the algorithm described in [8] must be modified to incorporate the fact that each node must develop its own estimates for the midpoint and edge distances. The modified version of the virtual coordinate algorithm is thus

$$\overline{r} = \sqrt{(\overline{x} - \overline{x_{\text{mid}}})^2 + (\overline{y} - \overline{y_{\text{mid}}})^2} \quad (8)$$

$$\overline{\theta} = \tan^{-1}\left(\frac{\overline{y} - \overline{y_{\text{mid}}}}{\overline{x} - \overline{x_{\text{mid}}}}\right)^* \quad (9)$$

$$\overline{vr} = \overline{r_{\text{edge}}}\cos^{-1}\left(\frac{\overline{r}}{\overline{r_{\text{edge}}}}\right) \quad (10)$$

$$\overline{v\theta} = \overline{\theta}, \overline{vx} = \overline{vr}\cos(\overline{v\theta}), \overline{vy} = \overline{vr}\sin(\overline{v\theta}) \quad (11)$$

where $\overline{x_{\text{mid}}}$ is an array containing the local estimates for the x coordinate of the midpoint of the swarm, $\overline{y_{\text{mid}}}$ is an array containing the local estimates for the y coordinate of the midpoint of the swarm, and $\overline{r_{\text{edge}}}$ is an array containing the local estimates for the edge distances for each node.

*In our implementation, we use the function `atan2`.

A method for generating these estimates is presented in the next section.

## VI. Edge Discovery

On a basic level, swarming can be carried out simply by using a node's knowledge of it's immediate neighbors. However, in order for nodes to be able react to the edges and centroids of swarms, they must have access to a greater range of information. Since for many nodes, the edges and midpoint will not be directly visible, they must rely on their neighbors to relay that information to them in a time and communication-efficient manner.

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 pass that single approximation along instead of the entire set of node positions. If that approximation is passed as part of the existing neighbor communications, then no additional messages will be required, and the existing message will only need to be lengthened by the position of the midpoint.

Since the swarm exists as a set of nodes, an obvious solution is to chose 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 thus has an estimate of its distance to the middle and edge of the swarm, and thus 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 centralised authority for the swarm, the points that act as edge representatives are self-selected. To determine whether a node should identify itself as an edge, nodes look at the positions of their immediate neighbors, checking whether there is an empty arc of at least 90 degrees between two of them. If such a gap exists, the node assumes that it is viewing a region of empty space, and is thus located on the edge of the swarm.

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 adaject 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.

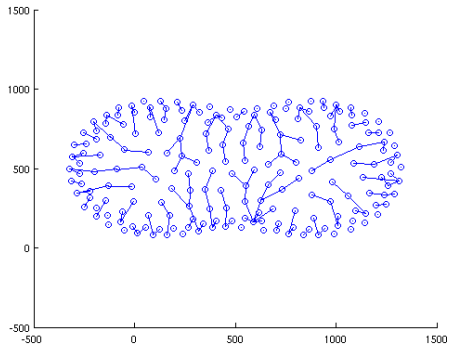
After the depth information has had enough cycles to propagate from the outside edges to the rest of the nodes in the swarm, there will be a set of nodes that cannot see any nodes deeper than themselves, though they may see other nodes at the same depth. These nodes select themselves as deep representatives, and pass their information back towards the outside in a similar manner as the shallow representatives. Nodes look for neighbors that are deeper than them, and store and pass on the position of the deep representative sent by the furthest deeper neighbor.

When determining adjacency for the purposes of this algorithm, ability to communicate is likely to be insufficient. 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 twice the distance of the closest neighbor. When searching for deeper neighbors, the adjacency distance is set to four times the distance of the closest neighbor, in order to minimize the occurance 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 adjustement 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.

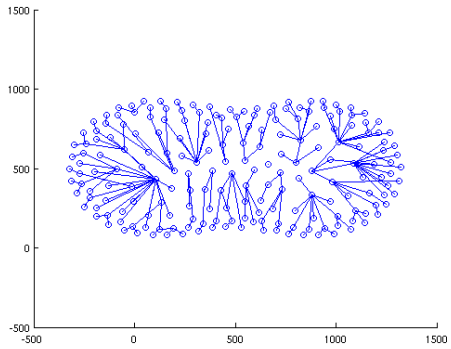
In our system, nodes effectively group themselves into two sets of trees. One set leads from the nodes on the outside edge, and specifies which other nodes will use those nodes' points to represent their closest edge. Another set leads from the deepest nodes, and specifies which other nodes will use 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. 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. 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 representitive 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




(a) Detected paths to edges




(b) Detected paths to middle

Fig. 2: Edge and Deep Trees

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, since it is likely that the edge is actually oriented more perpandicularly towards those points than towards the centroid. 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 [10], which utilizes a similar method for detecting edge distance.

## VII. Experiment Design

For the purposes of testing, the algorithm was implemented within a modified version of the Qualnet simulation platform, which handles actions and communications as discrete events, and simulates delay and signal loss in communications. In order to minimize signal interference, communication and computation for each sensor is handled in a staggered manner, with sensors waking up based on a timer, carrying out the algorithm, and then broadcasting their information, while the others passively listen for the broadcasts of others. The communication range is limited by the parameters of the simulated environment, though the algorithm also has a built-in limit on the distance at which messages will be accepted.

Three sets of experiments were performed with the algorithm, each based on a different sensed field.

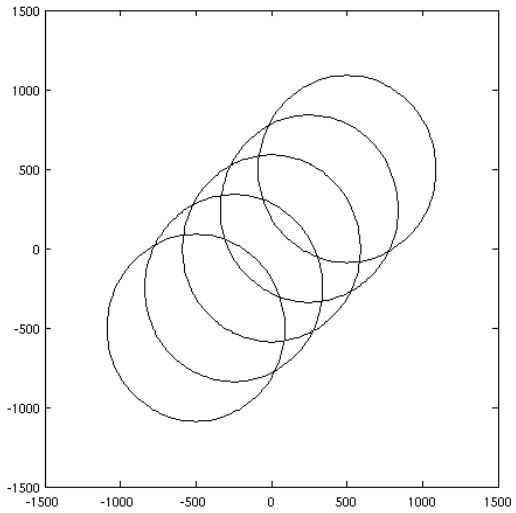



Fig. 3: Translating Circular Field

Each set consisted of multiple experiments, across which the number of nodes varied, with each experiment run using 50, 100, 200, or 400. In addition, the same configurations were used for a version of the algorithm with the virtual coordinate remapping, yielding uniform distributions of sensors across the region, with the same exterior edge but different interior node density. Each set therefore contained four Chebyshev runs and four corresponding uniform runs.

For all of the runs, the sensors were initially placed in a uniform rectangular grid spanning from the coordinates [0,0] to [1000,1000], though they flowed beyond those boundaries during the runs. The parameters used for the swarming algorithm were $F_{\text{factor}} = .01$, $a = 100$, $R_{\text{scale}} = 200$, and $R_{\text{factor}} = .02$. The target edge strength for all fields was edgevalue $= .5$. The scaling factor for acceleration was $\kappa = 1$. The fields used in the experiment were specified by a function of the x and y coordinates, with the coordinates rescaled from the initial range of [0,1000] to [0,1].

The runs consisted of two 4000 second segments. In the first, the field sensed by the nodes evolves over time, with the swarm adapting to fit the changing field. In the second, the evolution of the field ceases, giving the nodes time to settle into stable positions. The fields we worked with are:

1) **Translating Circular Gaussian Field.** This field takes the form of a circular bell shape which begins centered on the coordinates [500,500] and translates over a period of 4000 seconds to a new center of [-500,-500]. It is specified by the equation

$$\phi(x, y, t) = e^{-\left(\frac{1}{2 \cdot 500^2}\right)\left((x-(500-\frac{t}{4}))^2+(y-(500-\frac{t}{4}))^2\right)} \tag{12}$$

as shown in Fig. 3.

2) **Flexing Elliptical Gaussian Field.** This field takes the form of an elliptical bell shape which remains

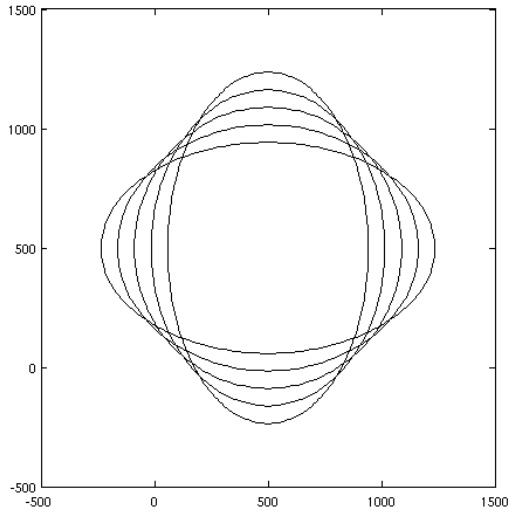



Fig. 4: Flexing Elliptical Field

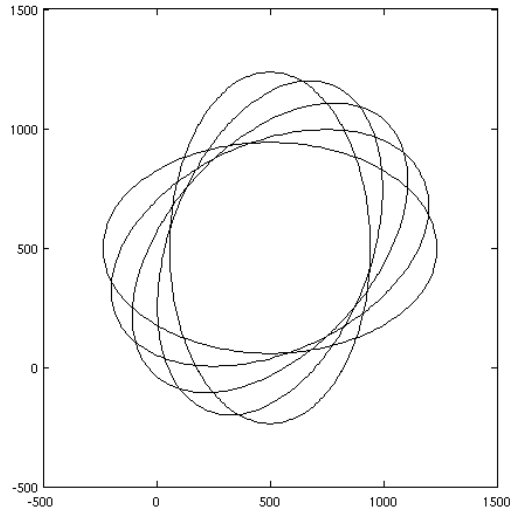



Fig. 5: Rotating Elliptical Field

centered at [500,500] but alters its x and y widths over time. It is specified by the equation

$$\phi(x,y,t) = e^{-\left(\frac{1}{2\cdot 500^2}\right)\left[\left(\frac{x-500}{.75+\frac{t}{8000}}\right)^2+\left(\frac{y-500}{1.25-\frac{t}{8000}}\right)^2\right]} \tag{13}$$

as shown in Fig. 4.

3) **Rotating Elliptical Gaussian Field.** This field takes the form of an elliptical bell shape which remains centered at [500,500] but rotates around that position over time. It is specified by the equations

$$\phi(x,y,t) =$$

$$\phi_0\left(\cos\left(\tfrac{\pi t}{8000}\right)\cdot(x-500) - \sin\left(\tfrac{\pi t}{8000}\right)\cdot(y-500)+500,\right.$$
$$\left.\sin\left(\tfrac{\pi t}{8000}\right)\cdot(x-500) + \cos\left(\tfrac{\pi t}{8000}\right)\cdot(y-500)+500\right),$$

where $\phi_0(x,y)$ is defined by

$$\phi_0(x,y) = e^{-\left(\frac{1}{2\cdot 500^2}\right)\cdot\left(\left(\frac{x-500}{.75}\right)^2+\left(\frac{y-500}{1.25}\right)^2\right)} \tag{14}$$

as shown in Fig. 5.

## VIII. Results

### A. Spatial Distribution at Equilibrium

A tracking swarm will not be of great use for interpolation if the nodes at equilibrium end up arbitrarily distributed. The main advantage of the system described in this paper is that one can control the sensing nodes' distribution at equilibrium. We exploit this feature to generate distributions that resemble ideal Chebyshev distributions. In this section, we address the fundamental question of how much resemblance exists between the spatial distribution obtained with the swarming rules described in this paper (referred to as *swarm distribution*) and an ideal Chebyshev distribution on a circular field. Our hypothesis is that if the swarm distribution is close to an ideal Chebyshev distribution on a circular field, the swarm distribution will also be good for interpolation of noncircular fields, or at least better than a uniformly distributed swarm.

First, we need to establish the ideal Chebyshev distribution on a circular field. The extension of a univariate Chebyshev distribution to more than one dimension depends on the shape of the interpolation domain (see, e.g., [11], [14]), so there is no universal two-dimensional Chebyshev distribution. Consequently, we explore two possibilities for a circular domain. In both cases, we populate with nodes a series of concentric rings with radii equal to the roots of a Chebyshev polynomial of degree $2n$, where $n$ is the number of rings. In case (a), each ring contains the same number of nodes uniformly distributed in the azimuthal direction (see Fig. 1 (b)). In case (b), the number of nodes per unit of length along the azimuthal direction is constant across rings. The result is an increasing number of nodes as the radius of a ring increases (see Fig. 6).

We performed an interpolation experiment with the same number of nodes ($N = 186$) but with different values for the shape parameter ($a \in [1, 10]$ in increments of $0.05$) on a static Gaussian field (Eq. 3) with . The results are shown in Fig. 7. Clearly, the behavior of the second variant of a Chebyshev distribution (case (b)) is superior than the first variant (case (a)). We therefore adopt the second case as the reference Chebyshev distribution for the rest of the paper. Hereafter, we refer to this reference distribution as the ideal Chebyshev distribution.

Let us now turn our attention to comparing the swarm distribution with the Chebyshev distribution. To do this, we work on the polar rectangle $[r \times \theta]$, where $r \in [0, 1]$ and $\theta \in [0, 2\pi]$. In the ideal Chebyshev distribution, a certain number of nodes share the same radial coordinate and only differ in their azimuthal coordinate. With our swarming algorithm, it is practically impossible to obtain exactly equal radial coordinates for the interpolation nodes. Therefore, we bin the radial axis and perform the comparison in terms of densities of

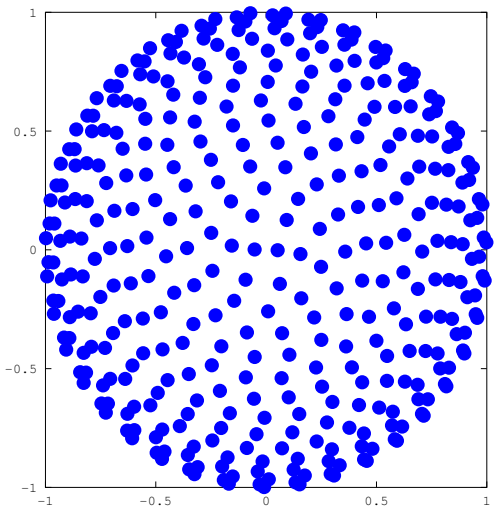


Fig. 6: Chebyshev Distribution with Constant Azimuthal Density of Nodes

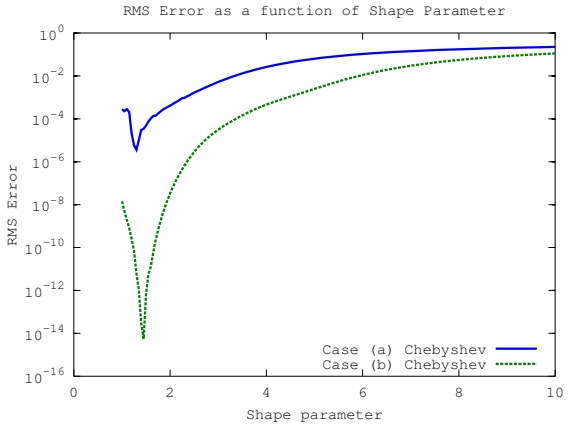



Fig. 7: RMS error obtained with two Chebyshev distributions (see text for details)

points per unit of length in the positive radial direction. Fig. 8 shows histograms that correspond to the "spectra" of the ideal Chebyshev, the uniform and the swarm distributions with 421 nodes, which are distributed across 16 rings. The centers of each bar are the actual locations of the roots of a Chebyshev polynomial of degree 32 ($2 \times 16$) in the range $[0, 1]$.

By centering the data around the exact location of a univariate Chebyshev polynomial, we can measure the deviation of the swarm distribution from the ideal Chebyshev distribution with a single number: the average ratio of nodes per bin (ARNB). Each ratio can be greater than or equal to one. If the ratio is one, then both distributions have the same number of nodes at that bin. Figure shows the ratios per bin of the uniform and swarm distributions with respect to the Chebyshev distribution. The ARNB for the uniform distribution is 1.1754, and the ARNB for the swarm distribution is 1.0683, which shows that the swarm distribution more closely resembles a Chebyshev distribution than a uniform distribution (as expected). In particular, Fig. 8 (d) shows that the swarm distribution approximates the Chebyshev distribution relatively accurately around the center of the radial axis and less accurately toward the extremes. Near the center of the field we do not expect this discrepancy to cause many problems since only a few nodes are in this area. The difference near the edge is more significant; however, as a matter of fact, many of these nodes lie *outside* the region of interpolation and they may be discarded before performing interpolation. The uniform distribution has more nodes in the interior than the Chebyshev distribution and fewer nodes close to the edges. To conclude, although the swarm distribution is not exactly equal to a Chebyshev distribution, it is indeed a good approximation.

### B. Field Tracking

While the internal distribution of the swarm is significant, another important factor is whether the swarm as a whole can successfully move to track the evolving field. The system proposed in this paper is capable of tracking dynamic fields (see Fig. 11) so we consider two means of measuring the performance of the tracking.†

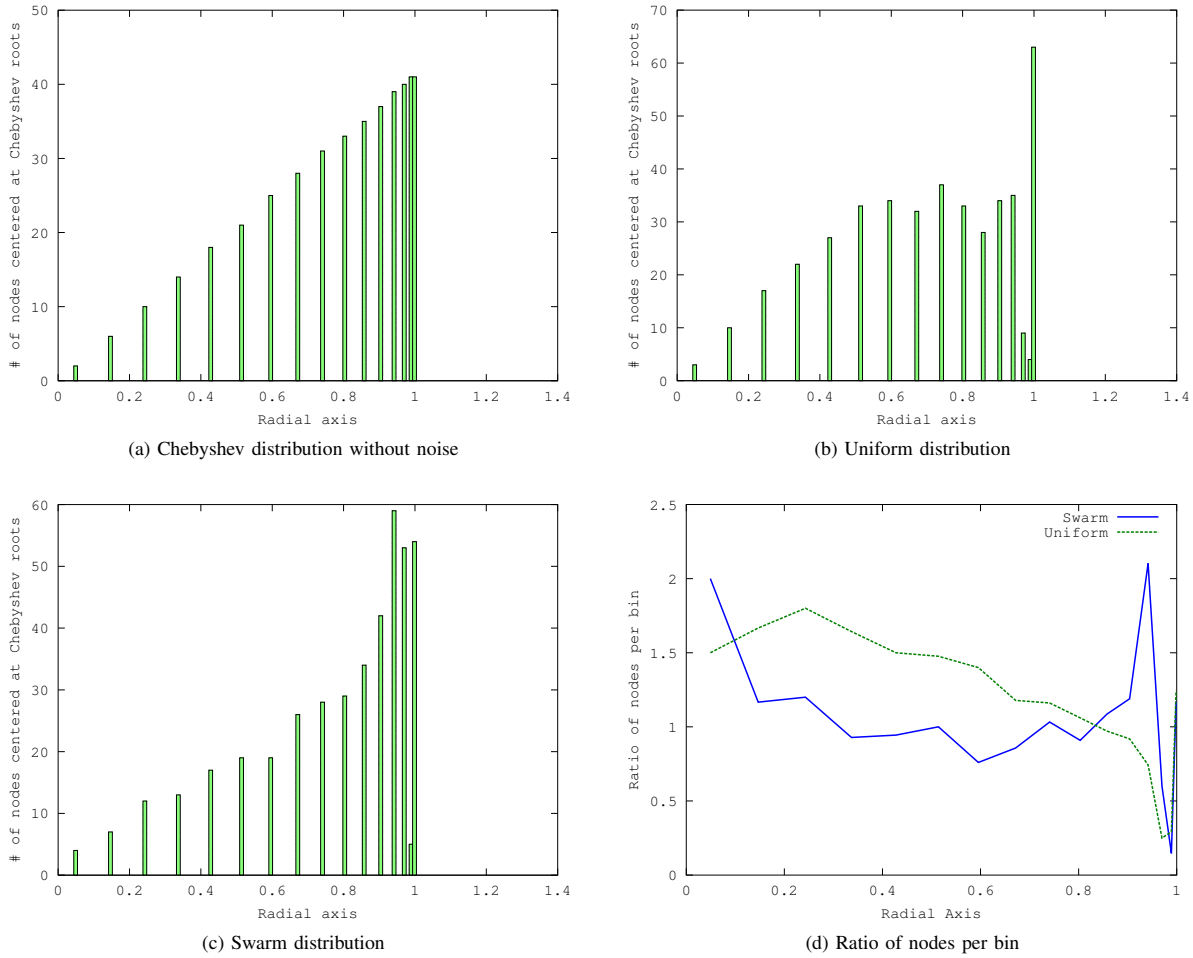
The first measure is based on the distance between the midpoint of the swarm and the midpoint of the two-dimensional bell curve formed by the sensed field. As the field evolves the midpoints of both it and the swarm will shift, so this serves to track the ability of the swarm to keep up with the motion of a field's curve. For cases where the center of the curve does not shift, such as the flexing and rotating fields, this measure will be less relevent, and so was not considered. For the translating fields, the values for all cases are given in Fig. 9.

In all cases, the value for the distance starts near zero, since the nodes were initially placed near the center of the field's curve. As the field evolves, all of the swarms showed issues in keeping up with the motion of the translating field, shown by an increase in the distance until the halfway point of the simulation at 4000 seconds. For most of the swarms, the increase seems to be rapid at first but slowly tapering off, which might be due to less interference between nodes as more of them transition to following the field rather than trying to fill it. Swarms with smaller node counts appear to have superior performance, as to uniform distributions over Chebyshev ones. The 200 and especially the 400 node Chebyshev distributions had particular difficulty, with a massive portion of the nodes falling behind.

After the midpoint of the simulation is reached and the field ceases to evolve, the distance between the centers begins to fall back to zero as the nodes settle into the field. This decrease is initially rapid, but slows as the swarm stabilized. Like before, the smaller and uniform distributions were quicker to settle, with the larger Chebyshev distributions failing to settle by the time the simulation concluded, though they would likely have shown similar behavior to the others given more time.

In general, the distance plots seem to show that while our swarming algorithm is capable of following translating fields to

†Please go to http://www.math.udel.edu/~mmontes/papers/swarmtracking/ to access videos showing the system in action.




(a) Chebyshev distribution without noise

(b) Uniform distribution

(c) Swarm distribution

(d) Ratio of nodes per bin

Fig. 8: Spectra of the ideal Chebyshev distribution, the uniform and the swarm distributions with 421 nodes. Fig. (d) shows the ratio of nodes per bin for the uniform and swarm distributions with respect to the ideal Chebyshev distribution (A value close to 1 is better).

a certain extent, the algorithm cannot be scaled up indefinitely, due to distributions with relatively large node counts showing a tendency to fall behind and collapse on themselves while attempting to follow smaller curves. It is also yet unclear whether smaller swarms that attempt to follow a field for an extended period of time will continue to increase in distance from the center. Fortunately, in many cases swarms seem to be able to successfully recover from tracking problems, though it takes a time scale similar to the one they spent following the moving field.

The second measure is based on the proportion of all nodes in the swarm that fall within a desired region of the field. This measures both the swarm's ability to keep up with motion, which is relevant to translating fields, and the ability to conform to shifting regions, which is relevant to the flexing and rotating fields.

For this evaluation, it may be desirable to adjust the value of the margin determining what lies inside and outside of the region. While the swarming algorithm was configured to transition at an edge value of .5, in practice the swarm tends to extend further beyond, as the non-scaling repulsion factor continues to apply to nodes on the edge, pushing them further outwards. Therefore, a lower cutoff may prove to provide a more useful indication of how well the swarm is conforming to the field. Fig. 10 shows an example of how a uniform group of 400 nodes fit into translating, flexing, and rotating regions given an edge value of .3, which even then proves insufficent to completely capture the full scope at rest.

For the translating fields, the results of the analysis were similar to the analysis of the distances between midpoints. The proportion of nodes within the region falls as the field evolves and the nodes struggle to catch up, but after the midpoint is passed, the nodes rapidly begin to catch up and the proportion rises until it reaches the stable rest value.

For the flexing and rotating fields, the plots seem to show that the swarm is tracking the general shape of the region

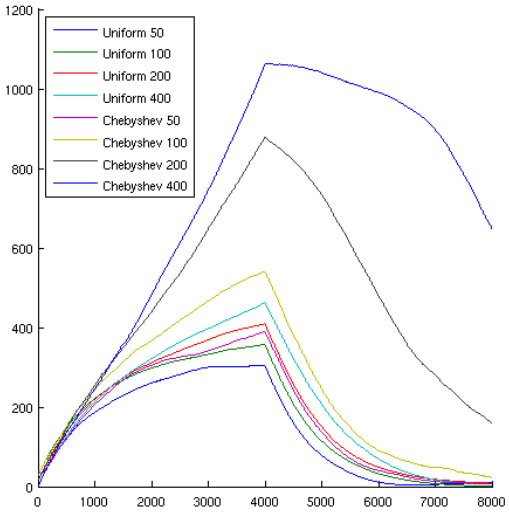



Fig. 9: Distances between the field and swarm midpoints for the translating field

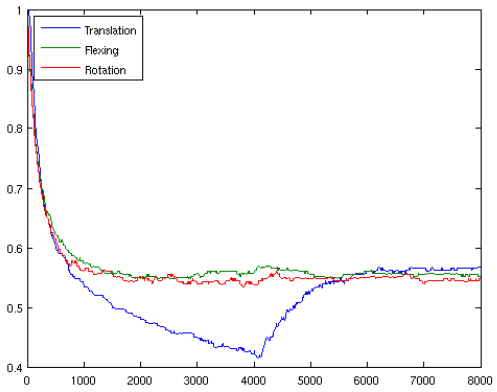



Fig. 10: Proportion of nodes with a sensed value greater than .3 in a 400 count uniform swarm by field type

well. Once the swarm fills out the initial region, changes in the proportion of nodes that remain within the region are relatively tiny, even as the field deforms itself. Larger node counts and Chebyshev distributions seem to have their nodes spread out wider, and thus lower proportions of those nodes within the region, possibly being due to the nodes being packed in more tightly along the edges, and thus spreading out further. One noticeable difference between the flexing and rotating fields was that for many of the simulations involving flexing fields, a minor but rapid increase was detectable right when the field ceased to evolve, while that effect was far less visible for rotating fields.

Overall, flexing and rotating fields seem to maintain their overall shapes well under the applied conditions, while translating fields did much more poorly.

## IX. Conclusions

When dealing with distributed sensing tasks, one has to decide how to spatially distribute the sensor nodes at one's disposal. The chosen spatial arrangement typically depends on the task at hand, and therefore there is no single spatial arrangement of sensors that will result in optimal performance for all possible distributed sensing tasks. Moreover, the environmental conditions in which the sensing task takes place may change over time. Ideally, therefore, sensors should relocate themselves, if they are mobile, or be relocated by the system designer in order to maintain an acceptable performance.

The work presented in this paper is aimed at tackling both of the aforementioned issues: optimal spatial distribution and rearrangement upon change detection. To show the effectiveness of our approach, we focused on the task of scalar field interpolation from scattered field measurements. The first issue is dealt with with an approach based on solid mathematical concepts. In particular, knowing that placing the interpolation nodes at the positions of the roots of a Chebyshev polynomial is the optimal case for polynomial interpolation in one dimension, we devised a set of swarming rules to approximate a Chebyshev distribution in two-dimensions. We showed in Section VIII-A that the resulting distributions approximate well the ideal Chebyshev distribution, especially in the interior of the sensed field. The second issue is tackled directly by the swarming rules, which are first and foremost designed to keep a swarm of mobile sensors together and within a certain range, but that also allow for tracking of fields that translate, expand, contract, and rotate over time. In Section VIII-B, we demonstrated the ability of the system to follow and adapt to these kinds of fields. However, our analysis showed that the system exhibits a trade-off between accuracy and speed that is exacerbated with large numbers of sensors. We believe that future work should be aimed at improving the proposed system's scalability and in extending it to three-dimensional scenarios.

*Acknowledgments.:* The work is supported in part by the National Science Foundation under grant CCF-0916035.

## References

[1] Battles, Z., Trefethen, L.N.: An Extension of MATLAB to Continuous Functions and Operators. SIAM J. Sci. Comput. 25(5) (May 2004), http://dx.doi.org/10.1137/S1064827503430126

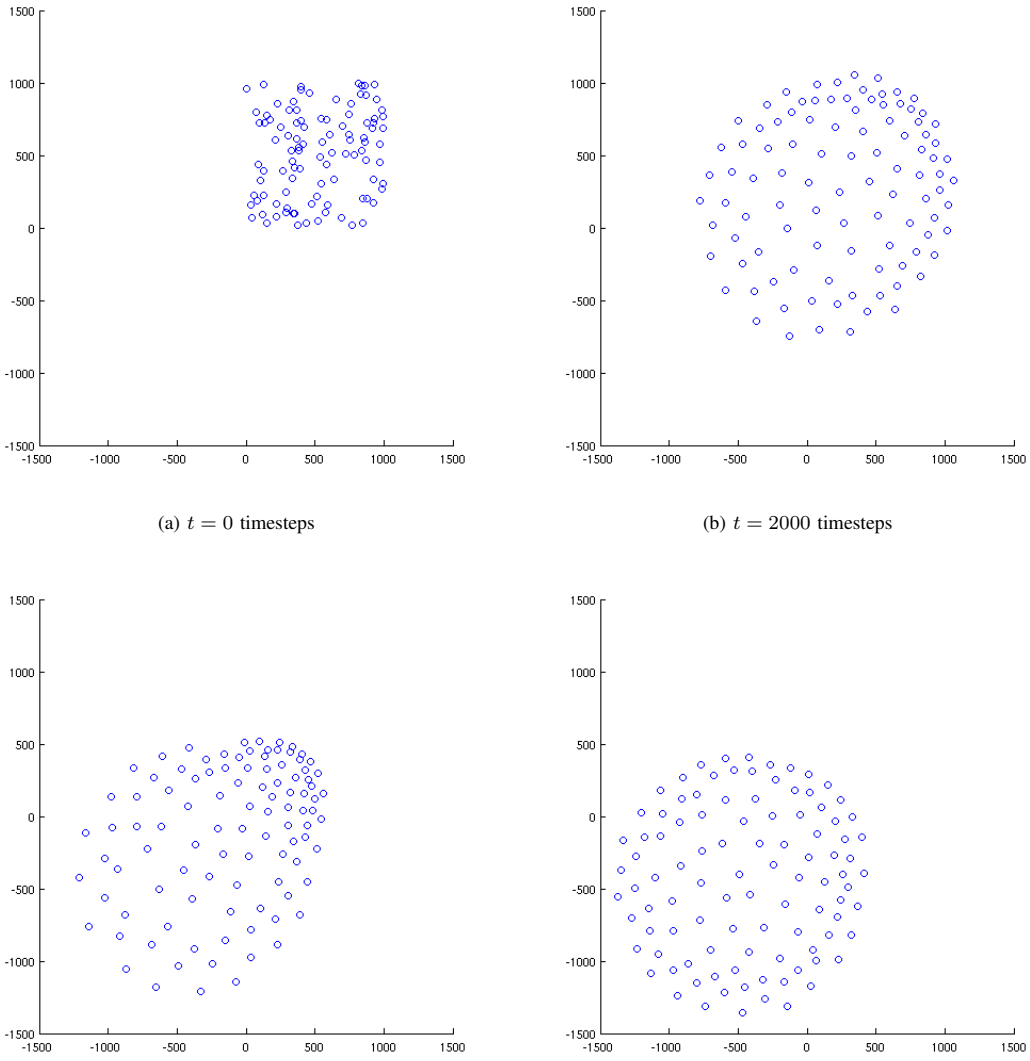
[2] Bertozzi, A., Kemp, M., Marthaler, D.: Determining Environmental Boundaries: Asynchronous Communication and Physical Scales. Lecture Notes in Control and Information Sciences 309, 403–405 (2005)

[3] Cortes, J., Martinez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. Robotics and Automation, IEEE Trans. on 20(2), 243–255 (april 2004)

[4] Cui, X., Hardin, T., Ragade, R., Elmaghraby, A.: A swarm-based fuzzy logic control mobile sensor network for hazardous contaminants localization. In: Mobile Ad-hoc and Sensor Systems, 2004 IEEE Int. Conf. on. pp. 194–203 (oct 2004)

[5] Dorigo, M., Montes de Oca, M.A., Engelbrecht, A.: Particle Swarm Optimization. Scholarpedia 3(11) (2008)

[6] Fasshauer, G.E.: Meshfree Approximation Methods with MATLAB. Interdisciplinary Math. Sci. – Vol. 6, World Scientific Publishing, Singapore (2007)




(a) $t = 0$ timesteps

(b) $t = 2000$ timesteps

(c) $t = 4000$ timesteps

(d) $t = 6000$ timesteps

Fig. 11: Swarm tracking a translating field


[7] Kalantar, S., Zimmer, U.: Distributed shape control of homogeneous swarms of autonomous underwater vehicles. Autonomous Robots 22(1), 37–53 (2007)

[8] Kirby, J., Montes de Oca, M.A., Senger, S., Rossi, L.F., Shen, C.C.: Swarm Interpolation Using an Approximate Chebyshev Distribution. In: Dorigo, M., et al. (eds.) LNCS 7461. Proceedings of the Eighth International Conference on Swarm Intelligence (ANTS 2012), pp. 324–331. Springer, Berlin, Germany (2012)

[9] Krause, A., Singh, A., Guestrin, C.: Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. J. Mach. Learn. Res. 9, 235–284 (Jun 2008), http://dl.acm.org/citation.cfm?id=1390681.1390689

[10] Liu, W., Jiang, H., Bai, X., Tan, G., Wang, C., Liu, W., Cai, K.: Distance transform-based skeleton extraction and its applications in sensor networks. Parallel and Distributed Systems, IEEE Transactions on (2012)

[11] Newman, D.J., Xu, Y.: Tchebycheff Polynomials on a Triangular Region. Constructive Approximation 4(9), 543–546 (1993)

[12] Trefethen, L.: Spectral Methods in MATLAB. SIAM, Philadelphia (2000)

[13] Turduev, M., Atas, Y., Morais de Sousa, P.A., Gazi, V., Marques, L.: Cooperative chemical concentration map building using Decentralized Asynchronous Particle Swarm Optimization based search by mobile robots. In: Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10). pp. 4275–4180. IEEE Press, Pistataway, NJ (2010)

[14] Verde-Star, L.: Symmetric multivariable Chebyshev polynomials. Applied Mathematics and Computation 187(1), 530–534 (2007)