Adaptive Contour Estimation Using Collaborating Mobile Sensors

Sumana Srinivasan
Advisor: Krithi Ramamritham, KReSIT, IIT Bombay
sumana@it.iitb.ac.in

ABSTRACT
Many real life applications such as tracking of pollutant flows, studying plankton populations in lakes, monitoring ocean currents etc. require real-time detection and tracking of level sets in a dynamic(spatio-temporal) field. In this paper, we examine the problem of estimating a level set or a contour of a particular value within a bounded region of varying field values using a network of mobile sensors. Given the task of estimating a contour, the sensors need to move towards the contour (converge phase) and then trace the contour (coverage phase). In order to minimize the error in estimation, the sensors need to trace all the points on the contour faithfully and to minimize latency, the maximum number of steps taken by the sensors during converge and coverage phases needs to be minimized. In our algorithm called ACE (Adaptive Contour Estimation), we overlap converge and coverage phases to minimize latency. ACE uses the history of movement of sensors to estimate the distance from the contour and strikes a tradeoff between arriving at the contour as quickly as possible and spreading out, compromising on latency initially, but reducing the overall latency by distributing the task of tracing the contour. ACE incorporates a collaborative wall moving algorithm during coverage phase for covering the contour thereby guaranteeing minimization of error in estimation for closed and continuous contours. We demonstrate that, irrespective of the type of initial deployment of sensors, ACE performs well (> 50% reduction in latency) when compared to a previous approach. We present results using both simulated and experimentally measured fields obtained by performing actual measurements of light intensity.

1. INTRODUCTION
Wireless sensor technology has emphasized the importance of in-situ [1] measurements that could potentially reduce the error in measurement and increase our understanding of large scale physical phenomena like contaminant flow [3]. Fine grain sensing can help scientists better understand the spatio-temporal dynamics of the phenomenon and forecast its behavior. Estimating contours corresponding to heavy pollutant concentration helps in containing the contaminants and this may provide an early warning for sensitive areas located in the vicinity of the spill. For such estimation, a wireless sensor network with mobile nodes is deployed to converge onto a specific contour. This forms the first step in the containment process. A user may be interested in estimating a contour corresponding to \( \tau \) which represents a hazardous level of concentration. Sample queries for such a mobile sensor network would be,

- Report all the spatial coordinates for the contour corresponding to pollutant concentration \( \tau \).
- Report spatial coordinates for a contour corresponding to \( \tau \) at times \( t_1 < t_2 < \ldots < t_k \).

There are many advantages of using a mobile sensor network instead of an “in-place” static sensor network. A mobile sensor network

- Improves sampling resolution as they can access those areas in the region which are unreachable for a static sensor network
- Eases deployment since sensors can be dropped off in the region where the phenomenon of interest has occurred and they can “intelligently” move and sample the region
- Adapts to the spatio-temporal dynamics of the phenomenon, while, redeploying a static sensor network to do the same might be prohibitively expensive

However, the main limitations are,

- Higher latency in estimating the contour
- Noise due to odometry errors in addition to actual sensing errors
- Higher energy consumption due to mobility

As mobility consumes most of energy in a mobile sensor network, an energy efficient algorithm for mobile sensors must minimize the maximum number of steps (latency) taken by the sensors in the network to accomplish the task. For any given sensor, the task of estimating the contour comprises of two steps: arriving at the contour and tracing the contour. Depending on its location with respect to the contour, it may be beneficial for the sensors to directly head towards the contour initially and spread out later when the sensor is close enough so that the task of tracing can be distributed amongst sensors. In our approach, we use an adaptive scheme which estimates the distance between the sensor’s current position and the contour and decides when and how much to spread out to minimize the overall latency.

The main features of our Adaptive Contour Estimation algorithm (ACE) are as follows.

1. Overlaps the movement of sensors towards the contour with tracing by the sensors already on the contour thereby reducing the latency of estimation
2. Uses history of its past movement to determine whether to spread out or head towards the contour

3. Uses a collaborative wall moving algorithm that guarantees minimum error in estimation for closed and continuous contours

By experimenting with real and synthesized fields we show an overall > 50% improvement in latency when compared to our previous approach [4] and an improvement of 10 – 27% using the adaptive scheme that dynamically determines whether a sensor should spread or approach the contour when compared to a scheme that does not use adaptation.

The rest of the report is organized as follows. In the next section, we present the problem definition and we follow this up with a discussion of our approach in Section 3. In Section 4 we describe our experimental test-bed and then present the results of our simulations. Related work is presented in Section 5 and we conclude with direction to future work in Section 6.

2. PROBLEM DEFINITION

Let $R$ denote a bounded field, and let $\tau$ denote the field value of the target contour. The goal is to estimate all the points in $R$ whose field values lie in the range $\tau \pm \epsilon$.

The main challenges that need to be addressed by any solution to the problem are due to variation in the field characteristics (continuous or has discontinuities such as obstacles, nonlinear variations in space and time, domain specificity), variation in contour characteristics (continuous or has multiple disjoint segments), different sensor characteristics (noise in measurement, odometry, calibration and saturation errors) and communication parameters (transmission range, delay constraints, noise etc.). Any solution depends upon the assumptions the solution designer makes with respect to the field, contour, sensor and communication characteristics. It depends upon the information the sensor has at every decision making step. We begin by making a few assumptions before we describe our strategy.

2.1 Assumptions

- **Field Characteristics:**
  - Field $R$ is continuous and assumes values $\tau$ at a finite number of points in the grid. Most of the real life, spatial distributions of parameters such as a pollutant concentration, temperature, pressure, humidity, light intensity are continuous in nature.
  - For any given sensor, the entire field $R$ or the function characterizing the entire field is unknown. It only knows the field values at discrete points in its neighborhood where it can sample and measure them. For convenience sake, we approximate the motion of the sensor along a square grid of length $l$. The value of the field is known to the sensor at its current location and at the eight neighborhood grid points (square tessellation) surrounding it. Field $R$ is therefore discretized as a square grid of length $l$.

- **Contour Characteristics:**
  - Contour is continuous and is approximated by discrete points on the grid. If the contour is discontinuous, some segments of the contour may not be traced depending on whether or not sensors have landed on the segments.

- **Sensor Characteristics:**
  - As sensors are noisy, we assume that the sensor measurement at a grid point is actually the expected value of multiple measurements made at that point.
  - Sensors are calibrated so that the error of measurement with respect to each other is determined and adjusted (e.g. via calibration constants).
  - The field values are within the linear operating range of the sensors (no saturation).
  - Sensors can move with a maximum speed much greater than the speed of the contour. For the purpose of estimation, the field is considered to be static.

- **Communication Characteristics:**
  - We propose two schemes, first, centralized or fusion center approach where all the sensors communicate to a fusion center for global information and second, de-centralized or autonomous approach where the sensors exchange messages to obtain global information. Fusion center can be any one of the sensors or a separate base station located in the region.

- **Localization**
  - Sensors are aware of their location and orientation. Sensors can align themselves based on geographic North direction and each sensor can be programmed to move in eight directions w.r.t. the geographic North for a pre-specified amount of time (in our lab experiment, we programmed the mobile sensor to have variable delays for each of the left and right servos in the robot to achieve rotation and linear motion).

- **Information Available**
  - Sensors can sense the field value at the current location as well as at the neighboring locations on the grid (8 points on a square tessellation).
  - Energy consumed for sensing, sensing cost, $k_{sens}$ is a constant
  - Sensors can communicate with other sensors and obtain the location information and field value at their current locations.
  - Sensors have enough buffer size to store a finite number of previously visited locations and their respective field values.

- **Energy Consumption**
- The overall energy consumption for estimation of any given sensor is the sum of energy consumed for sensing, movement, and communication.

\[ E_{\text{sensor}} = \sum_{i=1}^{n_{\text{steps}}} k_{\text{sens}} + k_{\text{mob}} + k_{\text{comm}} \]

- If \( k_{\text{step}} \rightarrow \) Cost of moving one step on grid and \( k_{\text{rot}} \rightarrow \) Cost per rotation of the mobile node, then

\[ k_{\text{step}} \leq k_{\text{mob}} \leq \frac{1}{2} k_{\text{rot}} + 2k_{\text{step}} \]

- If \( k_{\text{txr}} \rightarrow \) Cost of communication for sensors within the transmission range and \( n_{\text{hops}} \rightarrow \) Number of hops for the longest path then

\[ k_{\text{txr}} < k_{\text{comm}} < n_{\text{hops}} k_{\text{txr}} \]

Our problem can be defined as follows: Let \( f : [0,1]^2 \rightarrow [L, U] \) be a two dimensional field with unit grid granularity and bounded field amplitude with upper and lower bounds \( U \) and \( L \) respectively. Let \( C \equiv \{(x,y) \in [0,1]^2 : f(x,y) = \tau \} \) for some \( \tau \in [L, U] \) represent the set of points on the contour with field value \( \tau \). The task is to determine \( C \) with minimum latency and error in estimation.

Before we describe our approach in Section 3, we define performance metrics in the next section.

### 2.2 Metrics

The performance of any mobility strategy depends on the accuracy in estimation and the time taken for estimation. Accuracy is a measure of how well the sensors estimate the contour when compared to the actual contour. We define two metrics for error in estimation. MSE gives the deviation in the field value of the estimated contour and RCE is a measure of the deviation in the number of points on the estimated contour with respect to the actual contour. One way to measure RCE is to define two polygons, one bounding the actual contour and the other bounding the estimated contour and compute the difference between their areas as the error in estimation.

- **Mean Squared Error (MSE)**: If set \( C' \equiv \{(x'_i,y'_i) : 0 \leq i \leq k \} \) represents the estimated contour and if \( T \equiv \{\tau_0, \ldots, \tau_k\} \) represent the field values at these positions, then MSE is given by,

\[ \frac{1}{k} \sum_{i=0}^{k} (\tau_i - \tau)^2 \]  

(1)

Given that the target contour is actually a band \((\tau \pm \epsilon)\), we consider, the difference between \( \tau_i \) and \( \tau \) to be zero if \( \tau - \epsilon \leq \tau_i \leq \tau + \epsilon \).

- **Relative Contour Error (RCE)**: is defined to be the relative difference in the area between the bounding polygons (envelopes) of the actual and the estimated contours. Let, \( A_{\text{act}} \) be the area of the envelope of actual contour and \( A_{\text{est}} \) be the area of the envelope of estimated contour then RCE is given by,

\[ \frac{|A_{\text{est}} - A_{\text{act}}|}{A_{\text{act}}} \]  

(2)

- Latency (\( L \)): is defined as the maximum number of steps taken by any of the sensors to estimate the contour. Since the energy consumed is directly proportional to the distance traveled, latency is a measure of maximum energy consumed by a sensor due to movement. If, \( t_i \) is the number of steps taken by the \( i^{th} \) sensor, if \( N \) represents the number of sensors deployed and \( M \) is the number of sensors converged \((M \leq N)\) then,

\[ L = \arg \max_i (t_i) \text{where}, i = \{1, 2, \ldots, M\} \]  

(3)

The difference between RCE and MSE is that RCE is a measure of “spatial” error between the actual and estimated contours whereas the MSE is a measure of functional error (field values instead of coordinates) in estimate. RCE requires the area of the actual curve to be known to the evaluator whereas to evaluate MSE only the field value of the actual level set is needed. Also, the choice between the two metrics is application dependent. If the task is to just bound a pollutant spill, RCE can be sufficient (we need to find a bounding envelope and we do not care what the field values are at each point on the envelope).

### 3. DETAILS OF ACE

The overall problem of contour estimation can be split up into the following sub-problems.

- **Converge Phase**: Locating the sensors on the contour such that overall latency is minimized (see first figure in Figure 1).

- **Coverage Phase**: Tracing the points on the contour such that error and latency of estimation are minimized (see the second figure in Figure 1).

Any strategy towards solving the problem depends upon the information we have regarding the field, contour and sensor capabilities. If no information is available then, the best approach would be to scan the entire field. On the other hand, if the exact field characteristics are known then the contour points can be determined easily. If only partial information is available and if one can make certain assumptions about the field based on domain knowledge, then, can there be a solution with better performance than a complete scan?

Given that the sensors can explore their neighborhood, **can the sensors choose the direction where the field value decreases to move towards the contour faster?** Also given that the sensors can measure and store the field values during the course of their movement, **can the rate of change of field values over time and space be used to predict how far the sensor is from the contour and use this information to choose a direction that minimizes the latency?** Since sensors arrive at the contour at different times, **can tracing of the contour overlap with the arrival of other sensors onto the contour so as to minimize latency?** When sensors converge and begin to trace the contour, **can the information from the points traced by converged sensors be used to change the course of trapped sensors to move towards the contour?** Finally, **can sensors arrive in such a manner so as to cover the contour and thereby distribute the task of tracing and estimating the contour?** These are some of the questions that arise while designing a solution to the problem. We address each of them when we describe our approach.
deployed in the grid, the task is to locate $M$ as defined in Section 2. Given $S$ stated as an assignment problem. Every $s$ positions of $N$ sensors. Now, the converge problem can be
the sensors need to be located on the contour. Let $P$ to a unique

$\tau \equiv \{ (x, y) \in [0, l]^2 : f(x, y) = \tau \}$ for some $\tau \in [L, U]$ represent the set of points on the contour with field value $\tau \pm \epsilon$ as defined in Section 2. Given $N$ sensors at $(x_i, y_i) \in [0, l]^2$ deployed in the grid, the task is to locate $M \leq N$ sensors on C such that the error in estimation and the maximum number of steps taken by $M$ sensors to complete the estimation is minimized.

Next, we examine why this problem is hard. To begin with, let us assume that we know the points where the sensors need to be located on the contour. Let $P \equiv \{ p_0, \ldots, p_{N-1} \}$ represent points at regular intervals on the contour. Let $S \equiv \{ s_0, \ldots, s_{N-1} \}$ represent the initial positions of $N$ sensors. Now, the converge problem can be stated as an assignment problem. Every $s_i \in S$ is assigned to a unique $p_j \in P$ such that the maximum distance between sensors and the target points is minimized. In order to find the assignment for a given configuration $P$, that leads to minimum latency, $N!$ combinations have to be examined. Also if there are $m$ such configurations, where $m$ is the number of points on the contour, then, the number of configurations that need to be examined becomes $m!N!$. Figure 2 depicts the different assignments possible for $N = 5$ sensors. For large $m$ and $N$, solving the problem computationally becomes difficult. In our case, we have the added difficulty of not knowing the contour points before hand.

There are several approaches to solve the converge phase problem. One way is to divide the entire region into $N$ non-overlapping zones, distribute the sensors one per zone and allow the sensors to measure the field value at every point in their respective zones until they arrive at the contour. Even though, this approach guarantees estimation of the contour, it is not practical as it does not perform well if the field under consideration is large. Also, it does not use any of the information available to the sensor to restrict the area of scan and each sensor needs to be precisely placed one per
zone.

Another approach is to use nonlinear optimization techniques such as gradient descent. Previous research in the area of nonlinear optimization [5] indicates that this technique performs well when sensors are located close to the contour. However, far lying sensors tend to get trapped in local minima. The question then is, can we make use of the the positions of already converged sensors to “attract” the trapped sensors towards the contour? Other approaches in boundary estimation [6] use measurements from all sensors to estimate the gradient. In our scheme we do not estimate the gradient with respect to neighboring sensors but each sensor computes the gradient direction using the target field value and the field value measurement at its current location (the sensor moves in a direction that minimizes the difference between the current field value and the target field value).

In [6], the spread is achieved by each sensor moving away from its neighboring sensors and this requires communication of location at every step. We follow a different approach here based on angular distribution. Suppose we know an interior point inside the contour. Then, we can distribute the sensors equally around the interior point such that each sensor is assigned a target angle $\theta_i$. The entire region is divided into $(\frac{2\pi}{N})$ sections and for each sensor, $\theta_i$ is assigned to be the angle corresponding to the closest section. In our previous work [4], we assumed the knowledge of an interior point. However, in ACE, we start with an initial estimate and refine it as the knowledge of more points on the contour becomes available. The initial estimate of centroid is chosen to be the center of the region for clustered initial deployment (an initial deployment is said to be clustered if the maximum distance between the sensor nodes is less than a minimum threshold) or to be the center of the convex hull formed by the sensors (in regular and random deployments). When the converged sensor(s) trace the contour, the centroid estimate is refined to be the centroid of the points traced by the converged sensors.

Given that our objective is error and latency minimization, let us examine the issues in each of these.

1. Error Minimization — It is enough that at least one sensor land on the contour. However, with multiple sensors on the contour the latency of estimation can be minimized since the task of tracing the contour can be distributed. Also, when real-time deadlines are posed, using multiple sensors decreases error in estimation (by increasing the probability of the number of points being traced within the deadline). Also, when contours are discontinuous, using multiple sensors increases the probability of sensors landing on the the discontinuous portions and tracing them.

2. Latency Minimization — For any given sensor, the over all latency of estimation is the sum of converge and coverage phase latency. In order to minimize converge phase latency, it is of best interest to locate all the $N$ sensors deployed, equidistant from each other along the contour. The trade off here is, does this require increase the converge phase latency thereby increasing the overall latency? If the contour to be traced is of a large perimeter and the sensors are close to the contour, it may be beneficial for the sensors to spread more while they approach the contour. In
contrast, if the perimeter is small and the sensors are far away, it may be beneficial to directly arrive at the contour and then spread when they are close enough. Spreading earlier may result in higher latency in case of small contours located far away from the sensors and not spreading out at all may result in sensors being bunched together on the contour if the contour is large. This decision also depends on the spatial distribution of the sensors with respect to the contour. This intuition leads to interesting questions such as can the distance from the contour for each sensor be estimated? Can the perimeter of the contour be collaboratively estimated? Can the spatial distribution (clustered or uniformly distributed in $R$) be collaboratively estimated? If this is possible, we can use these estimates to decide whether to spread or not thereby minimizing the latency of estimation. In this paper we elaborate on estimating the distance of any given sensor from the contour and the other estimates will be explored in our ongoing work.

Hence our mobility model has the following requirements. It should allow sensors the ability to

1. **R1:** Move directly towards the contour
2. **R2:** Spread themselves around the contour.
3. **R3:** Dynamically choose between R1 and R2.
4. **R4:** Estimate the distance from the contour
5. **R5:** Estimate the perimeter of the contour
6. **R6:** Estimate the spatial distributions of the sensors (clustered vs. uniform)

In our current model, we consider requirements R1 through R4 and will consider R5 and R6 to be part of our ongoing work.

**Cost model to decide direction of movement:** Next, we describe our cost model for movement where we assign a cost to every neighboring point of the sensor on the grid based on two parameters, viz. **field distance** and **angular distance.** The cost $c_i$ for the $i^{th}$ sensor has two components namely, **field distance** component that is responsible for attracting the sensor towards the contour and **angular component** that enables the sensor to approach its target angle. Hence the cost $c_i(x_i, y_i)$ is a function of

- **Field Distance:** Difference between the field value at the current position $(x_i, y_i)$ and the field value at the contour $\tau$.

- **Angular Distance:** Difference between the current angle (angle with respect to the centroid of the hull formed by sensor initial positions at the current position) and the target angle $\theta'_i$.

The **field distance** $a_{cost}$ is given by

$$a_{cost}(x_i, y_i) = \left(1 - \frac{f(x_i, y_i)}{\tau}\right)^2 \tag{4}$$

where $f(x_i, y_i)$ is the pollutant concentration at a given position $(x_i, y_i)$, $\tau$ is the pollutant concentration at the contour, $a_{cost}(x_i, y_i) = 0$ when $f(x_i, y_i) = \tau$.

The **angular distance** $s_{cost}$ is derived as follows. Let, $\theta_i(x_i, y_i)$ be the angle with respect to the centroid for $i^{th}$ sensor at its current position $(x_i, y_i)$, $\theta'_i$ be the target angle of approach for the $i^{th}$ sensor (all angles are measured in radians). If, $\theta_d(x_i, y_i) = \theta_i(x_i, y_i) - \theta'_i$, then $s_{cost}$ is given by

$$s_{cost}(x_i, y_i) = \left(\frac{\theta_d(x_i, y_i)}{2\pi}\right)^2 \tag{5}$$

Note, $s_{cost}(x_i, y_i) = 0$ when $\theta_i(x_i, y_i) = \theta'_i$ at the given point $(x_i, y_i)$.

At every iteration, each of the sensors chooses one amongst the eight neighboring points which either minimizes the field distance or the angular distance with probability $\alpha$, the biasing factor. If $\alpha = 1.0$, the sensor moves only in the direction of minimizing the field distance. If the point has already been visited by the sensor before, the sensor is said to be in a local minimum and moves to the neighboring point that minimizes the distance between the sensor and the centroid of all points traced by the nearest converged sensor. If no sensor has converged, the sensor moves to that neighboring point that minimizes the distance between the sensor and centroid of the hull formed by the sensors. The converged sensors therefore behave as attractors for other non-converged sensors trapped in local minima.

**Effect of reducing $\alpha$ on latency:** In the previous section, we discussed how the decision whether to spread out or not depends on the distance of any given sensor from the contour. In our approach, we estimate this distance using the previous history of each sensor’s movement and adaptively set the value of the biasing factor $\alpha$. If the sensor is close enough (smaller than a minimum threshold) to the contour, then we set $\alpha$ to be low (say $\alpha = 0.5$, biased equally) so that the sensor moves more towards the target angle thereby spreading out. On the contrary, if the sensor is very far (larger than a maximum threshold), then we set $\alpha = 1.0$, the highest value. In order to estimate the distance from the contour, we use a technique that performs multiple non-linear regression using Nelder Mead Simplex optimization technique [15]. At each step, every sensor records its current position $(x_i, y_i)$ and the corresponding observed field value $z_i$. At the end of the $k^{th}$ iteration, we fit $\{(x_0, y_0, z_0), \ldots (x_k, y_k, z_k)\}$ using non-linear regression function of the form

$$z_i = p_0 + p_1 e^{-p_2 x_i} + p_3 * e^{-p_4 y_i}$$

where the coefficients $p_0, \ldots, p_4$ are estimated and the sum of squares error $ss$ is minimized.

$$ss = \sum_{i=0}^{n-1} (\text{observed } z_i - \text{calculated } z_i)^2$$

Once the non-linear functional form is estimated, we need to estimate $(\hat{x}, \hat{y})$ where $\hat{z} = \tau$. For this, we find $y_i$ for fixed values of $x_i \in [0, l]$ such that $y_i \in [0, l]$. We choose the closest $(x_i, y_i)$ pair to be an approximation of $(\hat{x}, \hat{y})$. The euclidean distance between the current position of the sensor $(x_k, y_k)$ and $(\hat{x}, \hat{y})$ is the estimated distance from the contour.

\footnotetext[1]{If $(x_c, y_c)$ represents the location of the centroid, then the current angle is computed as $\theta(x, y) = \tan^{-1} \frac{y-y_c}{x-x_c}$}

\footnotetext[2]{We divide by $2\pi$ in order to normalize}
In our experimentation, each simulation corresponds to an initial deployment of sensors and estimation of the contour by the sensors. At the end of simulation, in order to evaluate the quality of prediction, we define Average Prediction Error (APE), to be the average over the total number of sensors, the root mean square error in estimating the convergence point. If \((x_{act}, y_{act})\) represents the actual converge point and \((\hat{x}, \hat{y})\) the estimated point, then the distance between the two points is measured as the error in estimate. For \(n_{sim}\) simulations, APE is given by

\[
\delta = \sqrt{(x_{act} - \hat{x})^2 + (y_{act} - \hat{y})^2}
\]

\[APE = \frac{1}{n_{sim}} \sum_{n_{sim}} \sqrt{\frac{\delta^2}{N}} \tag{6}\]

Algorithm 1 describes the steps involved in converge phase.

### 3.2 Coverage Phase

Given \(N\) sensors that have converged on to the contour the task is to estimate points on the contour such that the maximum and the minimum field value of the estimated points depicted by \(\gamma_{max}\) and \(\gamma_{min}\) satisfies the following requirements.

- **Accuracy Requirement**: \(\gamma_{max} - \tau \leq \epsilon\) and \(\tau - \gamma_{min} \leq \epsilon\) for some pre-specified threshold \(\epsilon\). i.e. \(MSE \approx 0\).
- **Coverage Requirement**: The set of estimated points \(C'\) should cover the actual contour \(C\), i.e. \(RCE \approx 0\).
- **Latency Requirement**: If \(p\) represents the perimeter of the contour, then the maximum number of steps taken by any given sensor is \(p\) and the minimum number of steps taken by a sensor \(\frac{p}{N}\).

We use the traditional wall following approach [2] (used in solving mazes) to trace the contour. As long as the contour is connected, the wall following algorithm traces any arbitrary shape of the contour using either the right-hand or the left-hand rule. If the contour is disconnected, then the disconnected fragments are traced depending on whether a sensor has landed on the fragment or not. Every sensor that lands on the contour begins tracing the contour in a preset direction (e.g. clockwise). The sensor stops if it encounters a point that has already been traced by any other sensor. In Figure 3, sensor \(S_1\) traces \(p_1\) (\(S_2\) lands on the contour after \(S_2\) traces \(p_2\) and therefore \(S_2\) does not perform any trace as there is no corresponding \(p_3\)) and so on. Since the sensors can sense values only at the neighborhood grid points, the boundary is followed with a wall distance of a unit grid granularity. At the beginning of the trace phase, the sensor moves a unit grid distance away from the boundary. The sensor aligns with the contour such that the contour lies on its “right”. Every sensor is aware of geographic North, has sensory inputs from “right” and “front” neighboring points as shown in Figure 3 and aligns itself accordingly. In Figure 3 we depict two cases where the sensor is aligned to the contour boundary in NE and NW directions. The current position of the sensor along with its “front” and “right” positions are marked by S, F and R respectively. The path the sensor follows is shown by discs marked with “X” while the boundary is represented by black discs. The sensor moves in a clockwise direction. We follow the right-hand rule where

in the sensor maintains the contour on the right of the sensor. The sensor moves forward and turns left if it encounters the contour ahead of it. If the sensor moves too far away from the contour, it moves right again until its “right” sensory input is on the contour. At the end of Trace phase, the sensors aggregate their estimated points and the final estimate is communicated to the user.

Algorithm 2 describes the steps involved in trace phase. We conclude this section with a description of the two

---

**Algorithm 1 Algorithm for Converge Phase**

**Require**: \(N \rightarrow \text{Number of sensors, } S \equiv \{S_1, \ldots, S_N\} \rightarrow \text{Set of sensor IDs, } \tau \pm \epsilon \rightarrow \text{target contour field value, } \alpha \rightarrow \text{biasing factor, } l \rightarrow \text{length of the grid}\)

**Ensure**: A mapping \(A : M \rightarrow C\) where \(M \subseteq N\) and Set \(C \equiv \{(x,y) \in [0,l]^2 : f(x,y) = \tau \pm \epsilon\} \rightarrow \text{Level set or points on the contour with field value } \tau \pm \epsilon\)

Deploy the sensor nodes in the region.

Determine the distance from the contour

Set target angles

end if

end if

if \(\text{alreadyVisited}((x_{lc}, y_{lc}))\) then

Compute the nearest converged sensor \((x_{cen}, y_{cen})\).

end if

Compute the neighbor the point with least field distance cost, \((x_{lc}, y_{lc})\).

end if

Compute the neighbor the point with least angular distance cost, \((x_{lc}, y_{lc})\).

end if

end for

Compute the centroid of the points estimated by the nearest converged sensor \((x_{cen}, y_{cen})\).

Compute neighbor point \((x_{minP}, y_{minP})\) that minimizes the distance from the centroid as computed in the previous step.

end for

end do

end if

end if

end do

end do

end if

end do

end for

end if

end while

end do

end do

end while

end if

end if

end for

end do

end if

end do

end do
schemes for communication.

3.3 Communication

As described in Section 2.1, all sensors are reachable with some cost. Every sensor depends upon two types of information for its movement namely, local information such as field value measurements at its current location and neighboring grid locations and global information such as the centroid of all points traced by the nearest converged sensor (interior point), starting location of other converged sensors and points on the contour. Essentially, there are two communication schemes.

1. Centralized or fusion center approach

2. De-centralized communication

Figure 4 depicts the state transition diagram for a single sensor node in response to the messages and internal events. After deployment, the node waits for the fusion center to request location information from every node, characterized by message FM1 and compute the target angles. The nodes respond by providing their location information in message NM1. The target angle along with the node id is sent to all nodes. This is characterized by message FM2. The node then begins converge phase and is in state NS2 and starts to move to the least cost position characterized by event NE1. If the node gets stuck in local minimum characterized by event NE2, the node requests for information about the estimated interior point from the fusion center. This state is characterized by state NS3. Once the node receives the interior point information from the fusion center as characterized by FM3, the node makes a transition to state NS5 where it starts moving towards the interior point. If the node reaches the contour which is characterized by event NE3 then the node makes a transition to state NS4. The node informs the fusion center that it has converged by message NM3. The node then begins the trace phase which is characterized by event NE4 and transitions to state NS6. The node receives the locations of other converged sensors (required in order to determine the termination condition for wall trace) from the fusion center via message FM4. The node transitions to NS7 when it terminates the wall move when the event NE5 occurs. The node sends all the points traced by it in the coverage phase characterized by message NM4.

In the de-centralized scheme, the global information is sought by the sensors through multi-hop communication. The communication cost is inversely proportional to the square of the distance between the sensors. The cost is assumed to be a constant within the communication range of the sensors. We assume two modes of communication namely sendAll where the sensor tries to reach all the sensors and sendOneHop where the sensor transmits to only its neighbors (within the sensor’s communication range).

4. EVALUATING ACE WITH SYNTHETIC AND REAL DATA

For the field data, we have used two sets of data in our simulations. The first data set was generated by a pollutant flow modeling tool, WQMaP\textsuperscript{T}\textsubscript{M}\textsuperscript{3}. The output generated by WQMaP\textsuperscript{T}\textsubscript{M}\textsuperscript{3} was imported into a GIS tool, GRAM++

\textsuperscript{3}Applied Science Associates Inc., http://www.appsci.com/
and rasterized to generate a two dimensional pollutant concentration field of dimensions (500, 500). Figures 5(a) and 5(b) show the field and contours generated by WQMAP respectively. We use this data set to demonstrate the improvement in latency compared to our previous approach [4]. Two contours of values $\tau = 121$ and $\tau = 86$ are chosen for our comparison. The contour corresponding to $\tau = 121$ is convex in shape and is deeply embedded in the region and $\tau = 86$ has an irregular shape and is spread out in the region.

In order to test our algorithms on a realistic field we generated a real-world light intensity field data. We set up a light field of varying intensity within a 15X15 grid. We measured the light intensity at each of the grid positions using a single Crossbow Mote. We took 10 readings at each position and averaged the values. The field obtained by experimentation was interpolated to a grid of length $l = 140$. Figures 5(c) and 5(d) depict the field and the contours respectively. We use this data set to compare the performance of our algorithm with and without prediction as described in Section 3.1. We use $\tau = 162.0$ and $\tau = 204.0$ for comparison.

To study the feasibility of ACE using real mobile nodes, we used a Rogue robotics Basic Blue base system with a optical sensor mounted on the top. In this experiment we programmed the mobile robot to move on a 15X15 grid within a light field of varying intensity. The mobile node explored each of its eight neighbors and moved in a direction that approached a pre-set level set of light intensity.

Table 1 depict the parameters and their values used in our simulations.

### 4.1 Performance Gain of ACE over MCD (>50%)

In our previous work [4], we explored several strategies of movement where the sensors used gradient as well as an interior point (referred to as centroid) to arrive at the contour when trapped in a local minimum. In our simulations, a novel approach called MCD (Minimizing Centroid Distance) performed well compared to a greedy and the simulated annealing approaches but still exhibited a high latency MCD waited for all sensors to reach the contour before the coverage phase actually began. ACE estimates the interior point using the location information of other sensors and points on the contour traced by the converged sensors. The converge and the coverage phases are overlapped so that we do not compromise latency. ACE uses the points traced by already converged sensors as an estimation of interior point to attract trapped sensors directly towards the contour thereby reducing latency. In our simulations, we study the performance with respect to different types of deployments, different contours with number of sensors, $N = 10$, and show that our proposed algorithm (Adaptive) demonstrated > 50% improvement over our previous approach MCD as shown in (Table 3). We notice that the error in estimation as well as latency improvement is > 50%.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_{\text{sim}})</td>
<td>Number of Simulations</td>
<td>1000</td>
</tr>
<tr>
<td>(n_{\text{tera}})</td>
<td>Number of iterations per simulation</td>
<td>5000</td>
</tr>
<tr>
<td>(\tau)</td>
<td>Target contour field value</td>
<td>0.1 to 1.0</td>
</tr>
<tr>
<td>(l)</td>
<td>Length of grid</td>
<td>0.1 to 1.0</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>Bias</td>
<td>0.1 to 1.0</td>
</tr>
<tr>
<td>MIN thresholds</td>
<td>Minimum distance from the contour</td>
<td>10((\tau = 204.0)), 50((\tau = 150.0))</td>
</tr>
<tr>
<td>NUM samples for regression</td>
<td>No. of samples for regression</td>
<td>100</td>
</tr>
<tr>
<td>Deployment</td>
<td>Random</td>
<td>Uniform ([0 : l], 0 : l])</td>
</tr>
<tr>
<td></td>
<td>Regular</td>
<td>Diameter = (\frac{l}{2})</td>
</tr>
<tr>
<td></td>
<td>Clustered</td>
<td>Near the origin</td>
</tr>
</tbody>
</table>

**Table 1: Simulation Parameters and Values**

<table>
<thead>
<tr>
<th>Deployment</th>
<th>(\tau)</th>
<th>(\alpha = 0.1)</th>
<th>(\alpha = 0.5)</th>
<th>(\alpha = 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>162</td>
<td>173.1[172.4, 173.6]</td>
<td>175.2[174.4, 176]</td>
<td>111.99[111.98, 112.00]</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>Never Converged</td>
<td>Never Converged</td>
<td>71[70.99, 71.01]</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>84.2[81.8, 86.6]</td>
<td>70.4[69.7, 71.8]</td>
<td>57.3[56.3, 58.31]</td>
</tr>
<tr>
<td>Clustered</td>
<td>162</td>
<td>352.8[352.6, 353.1]</td>
<td>420.7[419.1, 422.3]</td>
<td>463.98[463.95, 464.01]</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>Never Converged</td>
<td>132.2[131.7, 132.7]</td>
<td>100.99[100.98, 101]</td>
</tr>
</tbody>
</table>

**Table 2: Latency with pre-set \(\alpha\) for \(N = 10\)**

### 4.2 Performance Gain Using History of Movement Information (10% - 27%)

In this evaluation, we demonstrate the effect of using history of movement to predict the distance of any given sensor from the contour and adaptively change the value of \(\alpha\) to increase the probability of convergence and also to decrease the latency. Table 2 depicts the latency values for different preset \(\alpha\) and deployments. We observe that the lowest latency is achieved for \(\alpha = 1.0\) for regular and random deployments for both the contours. For \(\tau = 162\), clustered deployment displayed lowest latency when \(\alpha = 0.1\). However for the same bias value for contour \(\tau = 204\), the regular and clustered deployments failed to converge. This indicates that presetting \(\alpha\) is not advantageous and as a result \(\alpha\) should be made adaptive depending on the position of the sensor with respect to the contour. We notice that prediction in the clustered deployment case (352.52 in Table 4) achieves the same lowest latency for clustered deployment (\(\alpha = 0.1\)) in the without prediction case (352.8 in Table 2). Since we use wall moving algorithm to perform the coverage phase, the RCE and MSE are always close to zero when sensors converged on to the contour with reduced latency. For distances larger than \(\tau = 204\), since the value of \(\tau\) is changed. This was set to a low value (10) for \(\tau = 162.0\) contour since the sensors’ path length before converging is small and a higher value (50) for \(\tau = 204.0\) since the path length in converge phase is higher. For distances larger than MIN THRESHOLD, \(\alpha = 1.0\) and for values less than or equal to MIN THRESHOLD we chose \(\alpha = 0.5\). The results along with the confidence intervals specified in parentheses are in Table 4.

For prediction, the sensor nodes performed a nonlinear fit on 10 sample data points. The MIN THRESHOLD refers to the minimum distance from the contour at which the bias is changed. This was set to a low value (10) for \(\tau = 162.0\) contour since the sensors’ path length before converging is small and a higher value (50) for \(\tau = 204.0\) since the path length in converge phase is higher. For distances larger than MIN THRESHOLD, \(\alpha = 1.0\) and for values less than or equal to MIN THRESHOLD we chose \(\alpha = 0.5\). The results along with the confidence intervals specified in parentheses are in Table 4.

From Table 4 for \(\tau = 162.0\), we observe that the random deployment produces the least latency. Using prediction does not seem to have any effect on latency for this case. The smaller values of latency for random case is obtained for \(\alpha = 1.0\) implying that it is beneficial to head directly towards the contour. However, for the clustered deployment case, we notice that prediction indeed results in lesser latency.

From Table 4 for \(\tau = 204\) we observe that using prediction indeed results in a lower latency for all deployments. For the prediction case, bias was set to \(\alpha = 1.0\) initially. When any given sensor’s estimated distance from the contour was less than or equal to MIN THRESHOLD, the bias was reset to \(\alpha = 0.5\). Setting the bias to very low values such as \(\alpha = 0.1\) resulted in non-convergence of sensors. For this case, random deployment produced the lowest latency. Since starting bias of \(\alpha = 1.0\) provided the smallest latency, we have not provided the results for lower values of starting bias.

The percentage improvement in latency for \(\tau = 204.0\) case was 10%, 13% and 27% respectively for regular, random and clustered deployments and the percentage improvement in latency for \(\tau = 162.0\) case clustered deployment was 22%. This indicates that when the sensors are deployed far away from the contour, prediction does better than when the sensors are deployed near the contour (regular/random deployments for \(\tau = 162.0\)).

### 4.3 Accuracy of Regression Fit

Figure 6 depicts a nonlinear regression fit for a sensor for \(\tau = 162.0\). We also calculated the average prediction error for each deployment. For the \(\tau = 162\) with random deployment and \(N = 10\) sensors, \(APE = 4.56\) and for \(\tau = 204\) with random deployment and \(N = 10\) sensors, \(APE = 7.46\). In \(\tau = 204\) case, ACE extrapolates to find out \(\hat{x}, \hat{y}\) where \(z = 204\). Since the value of \(\tau = 204\) is higher than \(\tau = 162\), the point \(\hat{x}, \hat{y}\) for \(\tau = 204\) is further distance than the point \(\hat{x}, \hat{y}\) for \(\tau = 162\). We believe that as the distance of the point to be estimated increases, the prediction error also increases.

In summary, we observe that, overlapping converge and coverage phases and using history of movement information for adjusting the value of \(\alpha\) increases the probability of sensors converging on to the contour with reduced latency. For random and regular deployments, when sensors are deployed very close to the contour, it is beneficial to approach the con-
Algorithm | Deployment | RCE(τ = 121) | Latency (τ = 121) | RCE(τ = 86) | Latency (τ = 86)
--- | --- | --- | --- | --- | ---
MCD | Random | 0.024 | 648.6 | 0.175 | 792.146
 | Regular | 0.01 | 626 | 0.10 | 837
ACE | Random | 0.001 | 207.13 | 0.058 | 276.1
 | Regular | 0.0 | 206.99 | 0.0 | 297.99

Table 3: Performance Comparison Between MCD and Adaptive Algorithms

<table>
<thead>
<tr>
<th>Deployment</th>
<th>τ</th>
<th>Latency (Adaptive α)</th>
<th>Latency (Preset α = 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>162</td>
<td>184.76(184.38, 185.14)</td>
<td>111.99(111.98, 112.00)</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>63.5[63.29, 63.7]</td>
<td>71.00[70.99, 71.01]</td>
</tr>
<tr>
<td>Random</td>
<td>162</td>
<td>156.40[153.01, 159.79]</td>
<td>155.43[151.91, 158.94]</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>49.83[48, 51]</td>
<td>57.3[56.3, 58.31]</td>
</tr>
<tr>
<td>Clustered</td>
<td>162</td>
<td>352.52[352.31, 352.74]</td>
<td>463.98[463.95, 464.01]</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>73.12[73.09, 73.18]</td>
<td>100.99[100.98, 101]</td>
</tr>
</tbody>
</table>

Table 4: Latency with adaptive α and preset α = 1.0

Figure 6: Observed Vs. Calculated Scatter Plot - Nonlinear Regression for Sensor 0

5. EXPERIMENTAL TESTBED

In order to validate ACE we are in the process of building a testbed with a fleet of robots Fire Bird 2, each equipped with

- **Microcontroller** — ATMEGA 128
- **Sensors** — 3 White line sensors, 2 Shaft encoders, 1 Light sensor mounted on a servo
- **Communication** — 2.4 GHz CDMA, Infrared, Wired RS232 serial, Standalone
- **Power** — Onboard Lithium ion battery
- **Motors** — 2 Ultra low power geared DC motors

A white-line grid of dimension (8 x 8) and a single light source is used to create a light field of varying intensity. A single Fire Bird 2 robot is used to test the feasibility. The robot navigates along the grid and sense the light field in eight neighboring locations using the sensor mounted on the rotary arm. All the sensors on board are read using ADC registers by the microcontroller. It is programmed as an interrupt driven system. The on-board timer is programmed to generate a pulse in regular intervals and the left and right servo motor movement and their velocity can be controlled by the microcontroller. The three white-line sensor values are feedback periodically and the robot is programmed to follow the white line. The spacing and the thickness of the grid lines are sensitive to the odometry errors (higher the thickness, larger the grid granularity implies smaller odometry errors but however the resolution of the contour decreases). We have chosen the thickness of the white lines to be 1.5" and the grid granularity to be 7". Shaft encoders are used to control the servo motors for the sensor arm and the left right translational movement. Each sensor can be programmed using the serial port connection and the on-board flash memory stores the program. Figure 7 depicts the test bed with white-line grid and Fire Bird 2.

Figure 8 depicts the values of light intensity threshold at the highest intensity position and the starting position for each experiment. We demonstrate that for each of these starting positions, the robot arrives at the correct field value. Table 5 depicts different starting positions of the robot and stopping position with field value. It also depicts the number of steps taken by the robot. The target contour field value was chosen to be 32 or (0x20h). We observe from the table that the robot does take the shortest path to the level set from its starting point. This confirms our basic premise that gradients do help in converging on to the contour optimally if they are deployed near. In the future we do propose to implement ACE completely and use the on-board communication to collaborate. We also plan to use light sources with dynamically varying light intensity to see how well the robots can track level sets.

Figure 7: Whiteline Grid with Fire Bird 2
Table 5: Fire Bird 2 with different starting positions and orientations

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Initial Position</th>
<th>Initial Orientation</th>
<th>Stopping Position</th>
<th>No. of Steps</th>
<th>No. of Rotations</th>
<th>Field Value</th>
<th>No. of Samplings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,0)</td>
<td>N</td>
<td>(3,3)</td>
<td>7</td>
<td>3</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>(5,0)</td>
<td>N</td>
<td>(3,3)</td>
<td>5</td>
<td>3</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>(6,2)</td>
<td>S</td>
<td>(3,3)</td>
<td>4</td>
<td>3</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>(0,3)</td>
<td>N</td>
<td>(3,4)</td>
<td>4</td>
<td>3</td>
<td>35</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>(5,1)</td>
<td>N</td>
<td>(3,3)</td>
<td>4</td>
<td>3</td>
<td>35</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>(6,4)</td>
<td>S</td>
<td>(3,3)</td>
<td>4</td>
<td>3</td>
<td>34</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>(3,6)</td>
<td>N</td>
<td>(3,3)</td>
<td>5</td>
<td>3</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>(6,6)</td>
<td>W</td>
<td>(3,3)</td>
<td>6</td>
<td>3</td>
<td>35</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 8: Light Field and Starting Positions of Fire Bird 2

6. RELATED WORK

Our work bears a strong resemblance to boundary estimation even though we perform level set estimation. In our case, we have an added knowledge of the value of the level set we are looking for and thereby, we use this information to do the path planning for the mobile nodes. This information is absent in the boundary estimating algorithms for mobile sensors proposed in the literature [6] [8]. In this section we compare our approach with existing boundary estimation approached using mobile nodes. Table 6 depicts the salient differences between ACE and approaches proposed in [6] and [8].

The basis of boundary estimation algorithms is the detection of the edge that separates two different regions [9]. Boundary detection and estimation using a network of static sensors has been studied extensively in the recent past [9]. The authors in [13] derive a theoretical bound on the number of sensors needed in a lattice network of static sensors to achieve a certain accuracy.

In [8], the authors use mobile nodes to adaptively scan the entire region to determine the boundary. Their strategy comprises of placing the mobile sensors uniformly along the bounds of the region and scan the entire field in a raster scan interleaved manner. This approach does not assume or make use of any local gradient information to converge on to the boundary. In contrast, in our approach, we assume that the field is continuous and bounded (needed to define the existence of a level set) and the mobile nodes use the gradient at their current position with respect to the target field value to converge on to the contour. For a single sensor, if it is deployed at the origin of a $[(0,0) : (l, l)]$ grid, and the nearest point on the contour is at location $(k, j)$ and if the granularity of scan is unity, the distance traversed by the sensor is $(k - 1)l + j$. If we assume a field with a uniform gradient towards the contour, the smallest number of steps taken by the sensor to arrive at the contour is $(k + j)$. If the gradients are not towards the contour, then the sensor can be hopelessly lost and can never converge on to the contour. While the adaptive sampling method guarantees convergence it does not exploit information available to the sensor. When the dimension of the field, $l$, is large, sampling the whole field might be prohibitively expensive. In our approach, we exploit local gradient and previous history information to arrive at the contour with smaller latency. Another interesting contrast is in the case of dynamic boundaries. In our approach, if the position of the level set is changing in time, the sensors can adapt since their motion is based on the current measurements. In adaptive sampling, the measurements made at the previous pass may not be valid at the current instant. It is our belief that we should exploit whatever field information we have to do the path planning and resort to scanning only in the case where such information is unavailable.

In [6] the authors propose the active contour algorithm used in image segmentation [14] to arrive at the boundary and trace the boundary by moving the nodes in and out of the boundary. The differences between the approaches are as shown in Table 6. Their approach requires communication at every step to determine gradient. However, in our approach, we know the target value of the level set. This target value is used to compute the gradient at every sensor position. The sensors collaborate only when they are trapped in local minima. Secondly, we achieve the spread by means of moving towards the target angle instead of moving away from the neighboring sensors. Thirdly, in our approach, we overlap the converge and the trace phases. In their approach [7], the agents are allowed to converge and in the trace phase the agents move in and out of the boundary to trace the boundary. The agents interlace the traverse such that they sample different points. The disadvantage of this scheme is that the agents have a higher latency since all of them traverse the entire boundary. However in our approach, we follow the wall moving algorithm and stop tracing when we arrive at a point already traced by another sensor. Table 6 compares the three approaches qualitatively.

In [10] the authors explore the use of mobile sensors to improve the quality of measurement by ensuring that there are enough sensors in a pre-specified critical region. However in our scenario, the position of the points on the contour is not known to the sensors. The work in [11] and [12] use a static sensor network to guide mobile sensors. in [12], the mobile node computes the gradient using the readings from the static sensors. Our algorithm is suitable for use where
<table>
<thead>
<tr>
<th>Approach</th>
<th>Path Length</th>
<th>Accuracy</th>
<th>Communication Overhead</th>
<th>Multiple Passes</th>
<th>Convergence and Coverage guarantees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Sampling[8]</td>
<td>always ≥ 2l</td>
<td>Depends on scan granularity</td>
<td>None (at the end of every pass)</td>
<td>Yes</td>
<td>Always</td>
</tr>
<tr>
<td>Active Contour[6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converge phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converge Phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Qualitative Comparison of Strategies

a static sensor network deployment is not feasible. In [11], the mobility is modeled based on group mobility. The group direction is determined by the static nodes. In our approach, the direction of sensor movement is determined by exploring its neighborhood and do not depend on any static network.

7. CONCLUSION

In summary, ACE demonstrates that overlapping converge and coverage phases, using history of movement to adjust the spread bias \( \alpha \), using already converged sensors as “attractors” to provide course correction for trapped sensors and the use of collaborative wall moving algorithm for tracing results in lowering the latency as well as the error in estimation by 50% when compared to MCD. Also, the performance gain by adapting \( \alpha \) using history of movement was 10%-30% depending on the initial deployment when compared to the case where \( \alpha \) was preset and not adaptive. In addition, we have also made an in-depth qualitative comparison between our proposed approach and other approaches to the problem of contour estimation. Since the problem of mapping sensors to equidistant points on the contour such that the maximum number of steps required is minimized is exponential in nature when the entire field is unknown, our proposed heuristic tries to use all possible information that is currently available to the sensors to estimate the level set so as to minimize the latency and error in estimation.

8. FUTURE WORK

We propose to incorporate estimation of perimeter and the area of the contour and use this information to minimize latency. We plan to implement real-time tracking with minimum latency and error in estimation constraints. The tracking algorithm should be able to deal with splitting and merging contours. We are in the process of implementing a test-bed with multiple communicating mobile robots capable of measuring light intensity in a light field to demonstrate our algorithm in a practical setting.

9. ACKNOWLEDGEMENTS

Our thanks to Parmesh Ramanathan, ECE, University of Wisconsin, Madison for his continued support and invaluable discussions. We thank the staff members of Embedded Systems Lab at KReSIT for their support in the design of the testbed. We thank the ASA and CSRE IIT Bombay for permission to use WQMAP and GRAM++ for generating the simulation data.

10. REFERENCES