Tracking environmental level sets with autonomous vehicles

Daniel Marthaler* and Andrea L. Bertozzi†
Department of Mathematics,
Duke University, Durham, NC 27708 and
Department of Mathematics, UCLA, Los Angeles, CA 90095

April 17, 2003

Abstract

We consider a collective motion algorithm for a swarm of homogeneous agents in a stationary environment. The agents cooperatively locate the boundary of a given environmental function in two space dimensions. Applications include tracking oil slicks, algae blooms, and other environmental indicators that occupy a localized area with a well defined boundary. Individual agents are controlled by local rules based on "snake" algorithms from image segmentation. In a previous paper we introduced a method in which the boundary tracking motion was driven by an edge detector for the concentration boundary. In this paper we consider a simpler method in which the agents find a pre-specified level set of the environmental concentration. We consider performance of the algorithm in the presence of sensor noise.

*daniel@math.duke.edu, www.math.duke.edu/~daniel

†bertozzi@math.duke.edu, www.math.duke.edu/faculty/bertozzi, bertozzi@math.ucla.edu, www.math.ucla.edu/~bertozzi

1 Introduction

Collective motion algorithms for autonomous vehicle platforms present a modern challenge with many applications. Such multi-agent systems have the potential to provide flexible and efficient alternatives to dangerous operations currently performed by humans. De-mining operations [7], mapping and exploration [6], navigation control [1], formation flying [2], and multiple rover planning [3] are a few such examples. In a previous paper [8], we proposed a method for a multi-agent system of autonomous vehicles (agents) to perform the exploration of a static environment. Applications include underwater plume monitoring, surveillance, and ecological sensing. We assume that each agent is equipped with an environmental sensor that measures the concentration of a substance confined to an area with a well-defined perimeter. The agents must collectively find and 'lock-on' to the boundary. Also, we assume that each agent communicates with other agents in a reliable manner. The frequency and range of communication are two parameters considered.

An analogous problem exists in image segmentation. There, the goal is to find the boundary of an object. This may be accomplished with a model known as a snake [5], or energy-minimizing curve. This process simulates elastic material which can dynamically conform to object shapes in response to internal or external forces. The basic snake model is a controlled continuity spline [11] under the influence of internal forces and external constraint forces. The internal forces serve to impose a smoothness constraint on the curve, while the external forces allow specification of the type of attribute we wish the snake to locate (in this case, edges).

Our previous paper showed that the snake algorithm from image processing could be directly adapted for use by autonomous mobile agents to find boundaries. The agents move along a virtual contour following simple rules whose overall dynamics result in convergence onto the boundary of an environmental function. As in the image problem, we use a partial differential equation (PDE) as the main motivation for the boundary tracking part of the algorithm. The snake algorithm moves a curve onto an edge; in our problem, the goal is to move a set of discrete agents onto a boundary.

The algorithm in [8] requires each agent to accurately estimate the gradient of an environmental edge detector function P, which we took to be $P = -|\nabla C|^2$, where C is the concentration of the plume. Thus each agent would have to estimate two derivatives of the concentration profile, which requires very sensitive information locally in space. In this paper, we propose a simpler edge detector method, in which the edge detector function P measures the difference between the actual concentration and the concentration along a level set which we define to be the 'boundary' of the plume, $P = |C_0 - C|^2$ where C_0 is the concentration level that defines the 'boundary' of the plume. The algorithm in this paper is designed to have the robots find where the plume concentration is exactly C_0, and to form an evenly spaced group around the perimeter of the plume, which we define to be the C_0 level set of the concentration function. The original reason for using $P = -|\nabla C|^2$ was, as in imaging, that it allows agents to find the largest jump in the concentration, regardless of the absolute value of the concentration. That is, it can be used to track a front where the concentration changes abruptly while the actual magnitude of the change is not known a priori. To use $P = |C - C_0|^2$, we must have some idea of the magnitude of the concentration C_0 that specifies the location of the boundary, for example a fraction of the concentration in the plume.

The rest of the paper is organized as follows: In Section 2, we briefly review image snakes; Section 3 describes terms unique to autonomous agents. In section 4, we present details of the algorithm and the details of the numerical method used in solving it. Section 5 presents simulations of the cooperative algorithm while section 6 shows that the single agent algorithm

from [8] works with the new P function. Finally, Section 7 shows a simulation that illustrates the algorithm's ability to adapt to noise.

2 Energy Minimizing Curves in Image Processing

We review the motivation for the snake algorithm [4]: The deformable contour model is a mapping

$$\Omega = \mathbb{S}^1 \to \mathbb{R}^2, \quad s \mapsto v(s) = (x(s), y(s))$$

where $\mathbb{S}^1$ is the periodic unit interval. For the purpose of this paper, we assume periodic boundary conditions; other boundary conditions are possible and used in image processing problems [5]. A deformable model is defined to be a space of admissible deformations F and functional E to minimize. This functional represents the energy of the model and has the following form:

$$\begin{aligned} E &: F \to \mathbb{R} \\ E(v) &= \int_\Omega \left[w_1 |v_s(s)|^2 + w_2 |v_{ss}(s)|^2 + P(v(s)) \right] ds \end{aligned}$$

where the subscripts denote differentiation with respect to the Lagrangian parameter s, and P is the potential function associated with the environment. In imaging, P is a function of the image data. For example, if we want the snake to be attracted to boundaries, then P could depend upon the gradient of the image intensity. The mechanical properties of the contour are specified by the functions w_j. Their choice determine the elasticity and rigidity of the curve.

If v is a local minimum of E, it satisfies the associated Euler-Lagrange equation:

$$-\partial_s(w_1 v_s) + \partial_{ss}(w_2 v_{ss}) + \nabla P = 0 \tag{1}$$

with periodic boundary conditions.

Equation (1) can be interpreted physically as a mechanical balance.

- The first two terms of equation (1) impose the regularity of the curve. The functions w_1 and w_2 impose elasticity and rigidity of the interpolated curve through the agents.
- In image segmentation, one may choose P as

  $$P(v) = -|\nabla I(v)|^2. \tag{2}$$

  where I denotes the image intensity function. This choice causes the contour to be attracted to sharp gradients, i.e. boundaries in the image. In [8], we chose $-|\nabla C|^2$ where C is the environmental concentration. Here we choose a different method with $P = |C - C_0|^2$.

One way to achieve a solution to equation (1) is to preform a time dependent gradient flow of the energy E,

$$\partial_t v = \partial_s(w_1 v_s) - \partial_{ss}(w_2 v_{ss}) - \nabla P \tag{3}$$

with steady state solving (1). Equation (3) is the motivation for the boundary tracking feature of the mobile agent algorithm developed in [8].

3 Agent Based Motion via "Virtual" Contours

Equation (3) is the starting point for the mobile agent algorithm in [8]. Additional dynamics address such properties as continual self-propulsion of agents, collision avoidance, and sparsing of the group. Self-propulsion is relevant for mobile agents that lack "hovering" capability, such as torpedo AUVs [12]. The connection between the agent motion algorithm and the deformable contour is that the agents are thought to occupy discrete points along a "virtual" contour.

The algorithm in [8] is constructed in stages. We follow that construction here: denote the two coordinates of the position vector v as $v(s) = (x(s), y(s))$, and assume $w_1 = \alpha$ and $w_2 = \beta$, are both constants. Putting equation (3) in component form, we also modify the ∇P term to have a fixed speed of magnitude λ,

$$\begin{aligned} \partial_t x &= \alpha x_{ss} - \beta x_{ssss} - \lambda \frac{\partial_x P}{\|\nabla P\|} \\ \partial_t y &= \alpha y_{ss} - \beta y_{ssss} - \lambda \frac{\partial_y P}{\|\nabla P\|}. \end{aligned} \tag{4}$$

The function $C(x, y)$ is an environmental concentration, for example algae, oil, or some chemical for which the agents are equipped with accurate sensors. Note that the modification of the form of the ∇P term from [8] allows us to think of that effect as occurring with a fixed velocity λ which could then be tuned to specific vehicular constraints.

The derivatives with respect to the Lagrangian parameter s can be approximated by finite differences:

$$\begin{aligned} \partial_t x_i - \alpha x_{ss} + \beta x_{ssss} &\approx \partial_t x_i - \frac{\alpha}{h^2}(x_{i+1} - 2x_i + x_{i-1}) + \frac{\beta}{h^4}(x_{i-2} - 4x_{i-1} \\ &+ 6x_i - 4x_{i+1} + x_{i+2}) \\ \partial_t y_i - \alpha y_{ss} + \beta y_{ssss} &\approx \partial_t y_i - \frac{\alpha}{h^2}(y_{i+1} - 2y_i + y_{i-1}) + \frac{\beta}{h^4}(y_{i-2} - 4y_{i-1} \\ &+ 6y_i - 4y_{i+1} + y_{i+2}) \end{aligned} \tag{5}$$

where the spacing between grid points in the parameter s is $h = 1/N$, the reciprocal of the number of agents N, and (x_i, y_i) now denote the x and y spatial coordinates of agent i. This means that the agents are thought of as "equally spaced" in the s parameter along the curve. If s were an arc-length parameterization, then the agents would also be equally spaced along the physical contour. The deformable model chosen does not preserve arc-length, although in practice, for sufficiently smooth boundaries, the distance between nodes is observed to vary smoothly. Finally, we mention that the coupled system (5) of ordinary differential equations is very stiff. This leads to restrictions on how frequently and how far each vehicle must communicate with other vehicles in order to result in stable collective motion.

Two additional effects needed are anti-collision (sparsing) and self-propulsion terms. We describe them now.

3.1 Sparsing

As in [8], we use an inflation force for the virtual contour. This is required in regions of constant plume concentration in order to keep the virtual contour from collapsing to a point under the elasticity and rigidity terms. The inflation motion will be implemented by pairwise interactions of vehicles through a repulsive potential of the form $U(v_j, v_i)$. The resulting inflation motion for each vehicle can be described by the ordinary differential equation

$$\partial_t v_i = \sum_{v_j, j \neq i} \nabla U(v_j, v_i), \quad v_i = (x_i, y_i). \tag{6}$$

Here we use the repulsion kernel $U(v,w) = \exp(-|v-w|)$. It is the same form repulsion as in [8], i.e. $U(v,w) = C_r \exp(-|v-w|/l_r)$, with $C_r = l_r = 1.0$.

3.2 Self-Propulsion

The algorithm is adaptable to platforms of vehicles that can not hover, by including self-propulsion of the system along the environmental boundary. This propulsion is implemented in the model by the addition of the following term to the velocity of each agent:

$$\partial_t v_i = \omega \frac{\nabla^\perp C(v)}{\|\nabla^\perp C(v)\|}, \tag{7}$$

where ω is the speed of each agent, and C once again denotes the environmental concentration. This is the same self-propulsion used in [8] for the single agent case. The underlying motion is that each agent moves at a constant speed tangent to the gradient of the concentration. In [8] the multi-agent system had a self propulsion in the direction tangent to the virtual contour.

4 Implementation and Communication

Combining (5), (6) and (7), we have the full algorithm for the motion of the ith agent at position $v = (x_i, y_i)$ in terms of the positions of the other agents and sensor information:

$$\begin{aligned}
\partial_t x_i &= \frac{\alpha}{h^2}(x_{i+1} - 2x_i + x_{i-1}) - \frac{\beta}{h^4}(x_{i-2} - 4x_{i-1}6x_i - 4x_{i+1} + x_{i+2}) \\
&- \partial_x P(x_i,y_i)/\|\nabla P(x_i,y_i)\| - \omega \frac{\partial_y C(x_i,y_i)}{\|\nabla^\perp C(x_i,y_i)\|} + \eta \sum_{j\neq i} \nabla U\left((x_j,y_j),(x_i,y_i)\right) \qquad (8) \\
\partial_t y_i &= \frac{\alpha}{h^2}(y_{i+1} - 2y_i + y_{i-1}) - \frac{\beta}{h^4}(y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}) \\
&- \partial_y P(x_i,y_i)/\|\nabla P(x_i,y_i)\| + \omega \frac{\partial_x C(x_i,y_i)}{\|\nabla^\perp C(x_i,y_i)\|} + \eta \sum_{j\neq i} \nabla U\left((x_j,y_j),(x_i,y_i)\right)
\end{aligned}$$

The parameter η allows us to control the amount of repulsion between agents, depending on both environmental condition and time evolution toward the boundary. The parameter λ denotes the speed at which the boundary is approached and it can be tuned to have the algorithm weigh more heavily the environmental concentration compared to other terms such as self-propulsion tangent to the contour (represented by ω) and the material properties α and β of the virtual contour.

In practice, we implement a split step algorithm in which the group of agents alternates between the dynamics of the virtual contour (elasticity and rigidity), and the individual agent motion with sparsing. To dynamically implement this method, each agent must communicate with other agents to determine their planned motion. In practice this communication occurs in discrete 'time steps' requiring us to replace the time derivative v, $\partial_t v$ by a finite difference

$$\partial_t v_i \approx \frac{v_{i+1} - v_i}{\Delta t}.$$

In practice for the numerics we find it convenient to perform a split step method in which the contour dynamics is advanced over one step, followed by the environmental function, repulsion and self-propulsion on a second step:

- STEP 1 ("PDE" term),

$$v_i^{n+1/2} = v_i^n + \Delta t \left(\frac{\alpha}{h^2} \left(v_{i+1} - 2v_i + v_{i-1} \right) - \frac{\beta}{h^4} \left(v_{i-2} - 4v_{i-1} + 6v_i - 4v_{i+1} + v_{i+2} \right) \right).$$

- STEP 2 ("ODE" term),

$$v_i^{n+1} = v_i^{n+1/2} + \Delta t \left(-\lambda \nabla \frac{P(v_i^{n+1/2})}{\|P(v_i^{n+1/2})\|} + \omega \frac{\nabla^\perp C(v_i^{n+1/2})}{\|\nabla^\perp C(v_i^{n+1/2})\|} + \sum_{j \neq i} \eta \nabla U(v_j^{n+1/2}, v_i^{n+1/2}) \right).$$

The superscript $n+1$ denotes the position at the next time. We use $n+1/2$ to denote the intermediate time for the split step.

Since the algorithm will be performed by agents, communication must occur at each step. Hence it is advantageous to perform the algorithm with as large a time-step as possible. Note that for a localized potential U that cuts off at a finite radius R, communication in step 2 will only be required over distances $d < R$ (which amounts to local coupling). The simulations in section 5 use a potential with infinite extent. The communication required to perform Step 1 depends on whether the finite differences on the RHS are evaluated at the old time (forward Euler method) or the new time (semi-implicit method). In [8], the environmental term was updated in the first step. We shift this term to the second step since we do not envision implicitly updating ∇P. Therefore, we group all of the terms that may be updated implicitly and those that will always be updated explicitly.

Both types of update for the first step are now discussed.

4.1 Forward Euler (explicit) Update

Using an explicit update, step 1 can be solved by the explicit formula

$$v_i^{n+1/2} = v_i^n + \Delta t \left(\frac{\alpha}{h^2} \left(v_{i+1}^n - 2v_i^n + v_{i-1}^n \right) - \frac{\beta}{h^4} \left(v_{i-2}^n - 4v_{i-1}^n + 6v_i^n - 4v_{i+1}^n + v_{i+2}^n \right) \right).$$

Note that agent number i only needs to know the positions of agents $i-2$, $i-1$, $i+1$, and $i+2$ in addition to its own position, in order to move accordingly. Thus, this implementation requires only local communication between agents, not global communication of all agents. However, there is a significant drawback to using this method. Stability of the algorithm requires that the time step $\Delta t < h^4/(8\beta + 2\alpha h^2)$. In particular, as the number of agents N increases, the algorithm must be updated on a timescale that decreases as $1/N^4$ for N large. This result comes from a von Neumann stability analysis of the method, which we do not derive here but is standard for finite difference schemes of linear partial differential equations [10].

4.2 Semi-implicit Update

An implicit update can alternatively be used

$$v_i^{n+1/2} = v_i^n + \Delta t \left(\frac{\alpha}{h^2} \left(v_{i+1}^{n+1/2} - 2v_i^{n+1/2} + v_{i-1}^{n+1/2} \right) - \frac{\beta}{h^4} \left(v_{i-2}^{n+1/2} - 4v_{i-1}^{n+1/2} + 6v_i^{n+1/2} - 4v_{i+1}^{n+1/2} + v_{i+2}^{n+1/2} \right) \right). \quad (9)$$

Update Type	Time step	Communication
Forward Euler (Explicit)	$\Delta t \leq \frac{h^4}{8\beta+2\alpha h^2}$	Local $(i \pm 0, 1, 2)$
Semi-implicit	$0 \leq \Delta t$	Global

Table 1: Communication frequency and range for stable implementation of the forward and backward Euler methods for the boundary tracking part of the algorithm (Step 1).

Note that in this situation, the new positions v_i^{n+1} must be determined implicitly by solving the coupled systems of linear equations above. This requires each agent to perform a linear algebra computation, which is only $O(N)$ because the matrix is pentadiagonal (e.g. a standard LU decomposition algorithm is very straightforward to use). However, in order for each agent to solve for their next position, they must know the positions of *all* the other agents in order to solve the implicit equations. The benefit of this method is that it is unconditionally stable for arbitrarily large time steps Δt. In practice, the time-step will be restricted by other factors such as the implementation of the sparsing and self-propulsion parts of the algorithm and desired convergence to the hybrid dynamics. We summarize the time-step constraints and communication constraints in Table 1.

4.3 Interpretation of Stability

Since there are two options for the algorithm, the question arises, which one should we use? The answer depends upon how often, how far, and with how many other agents each agent can communicate. With the explicit method, there is an upper bound on the amount of time between agent interaction. This bound is proportional to $1/N^4$ where N is the number of agents. On the other hand, the implicit update has no upper bound on the time step. This means that we may wait longer between communications, but at the price of having to communicate with all of the other agents. Note that we use the same explicit evaluation of the concentration function for each proposed version of Step 1. Therefore, in practice when using explicit updates, we may do multiple iterations of the boundary tracking part before taking new sensor readings (corresponding to the evaluation of P) or enforcing the anti-collision (sparsing) term.

5 Cooperative motion simulations

We now show some simulations that illustrate the difference between the algorithm in [8] which uses $P = -|\nabla C|^2$ and the proposed algorithm here which uses $P = |C - C_0|^2$. The following simulations will all have a time-step of 0.001, an implicit integrator for step 1 (as always, step 2 is updated explicitly) and the parameter values: $\alpha = 10^{-2}$, $\beta = 10^{-5}$, $\omega = 2.0$, $\lambda = 1$, $\eta = 1.0$, and 25 agents. The concentration function, $C(v)$, is taken to be the same as that in [8]:

$$\begin{aligned} C(x,y) &= \tanh\left(F(x,y) - \frac{1}{2}\right) \qquad (10) \\ F(x,y) &= \sum_{i=1}^{4} \exp\left(-((x-x_i)^2 + (y-y_i)^2)/\sigma^2\right) \end{aligned}$$

where $(x_i, y_i) = (1,0), (0,1), (-1,0), (0,-1)$, for $i = 1,2,3,4$, and $\sigma = 1.0$.

- **Convergence to level set**: Figure 1 shows the convergence of the algorithm with $P = |C - C_0|^2$. The agents inflate until they are close to the level set $C_0 = .01$, then track it indefinitely. Notice that the agents have converged to the level set by the third panel.

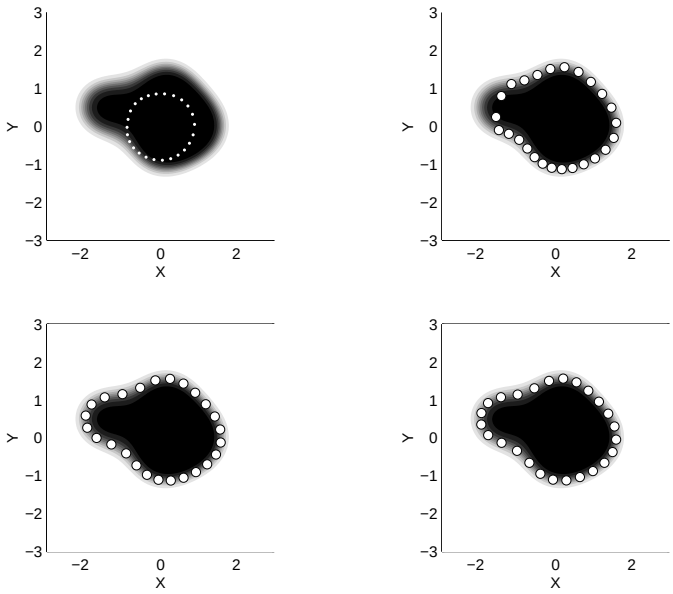

Figure 1: Convergence of the algorithm with $P = |C - .01|^2$. In this simulation, $C_0 = 0.01$, and the agents were initialized on a circle of radius 0.25. The four panels (moving top-left, top-right, bottom-left, bottom-right) show snapshots of the 25 agents at times $t = .05, .25, .75$,and 1.5, respectively.

- **Comparison between gradient and level set methods**: The algorithm from [8] used a different function for P. Figure 2 compares the positions of the agents from the two algorithms using the exact same parameter values, initial conditions and stopping time. One can see that both algorithms are converging to the boundary of the concentration. The gradient algorithm can be made to perform better with a different choice of the repulsion parameter. Both algorithms perform well, however the level set method requires the approximation of one less derivative.
- **Dependence on concentration value**: Changing C_0 makes the agents converge to a new level set of (10). Figure 3 shows the position of agents for two different values of C_0. The concentration function rapidly drops off near the boundary, causing the agents to be close to the boundary in both cases.

6 Boundary tracking without communication

In the case where only a few vehicles remain or are initially available, it makes sense to create an algorithm for single agent use or for small groups of agents with no or little communication.

In [8] we proposed ODE dynamics for a single agent to find and follow along a concentration boundary. Here we revise those dynamics to better match the snake algorithm constructed in

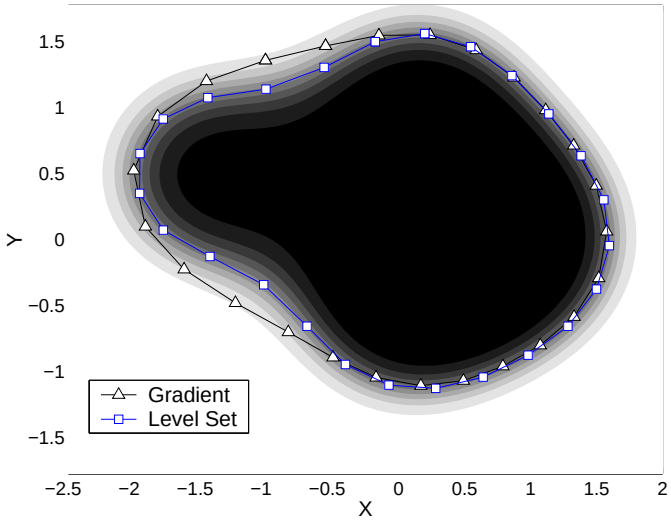

Figure 2: The figure compares the positions of agents at $t = 1.5$ using both the gradient algorithm from [8] and the level set algorithm. The triangles denote the positions using $P = -|\nabla C|^2$, and the squares, the positions when $P = |C - 0.01|^2$. The parameters for the algorithm were identical in both cases.

section 4:

$$\begin{aligned} \frac{dx}{dt} &= -\lambda \frac{\partial P}{\partial x}/|\nabla P| - \omega \frac{\partial C}{\partial y}/|\nabla C| \\ \frac{dy}{dt} &= -\lambda \frac{\partial P}{\partial y}/|\nabla P| + \omega \frac{\partial C}{\partial x}/|\nabla C| \end{aligned} \tag{11}$$

In [8], we used $P(x, y) = -|\nabla C(x, y)|^2$ because the boundary of C lies at a minimum of P, thus the gradient flow will move a single agent to a special point on the boundary, a local minimum of P. There, the agents will move along level curves of C with speed ω.

Figure 4 demonstrates that the single agent algorithm works with P defined in terms of level sets: $P = |C - C_0|^2$ in the above. As in [8], the agents are not spreading themselves out uniformly along the boundary, but instead, are bunching up. This is *not* cooperative behavior. This is due to the agents slowing down in areas of steep gradient along the boundary.

7 Robustness under sensor noise

In [8] we considered robustness of the vehicles to loss of members of the group. We saw that up to 50 percent vehicle loss could occur in a group of 50 agents while the algorithm continued to perform. In this paper, we investigate the effect of noise from the environment.

Robots utilizing (8) need to have the ability to measure local concentrations. These concentration measurements would then be used to approximate the gradients in the algorithm. Field deployed sensors are susceptible to various environmental and man made noise though, so it is important to have an algorithm that continues to perform if less than perfect knowledge of the environment is available. We now demonstrate that the algorithm can still function in the presence of sensor noise.

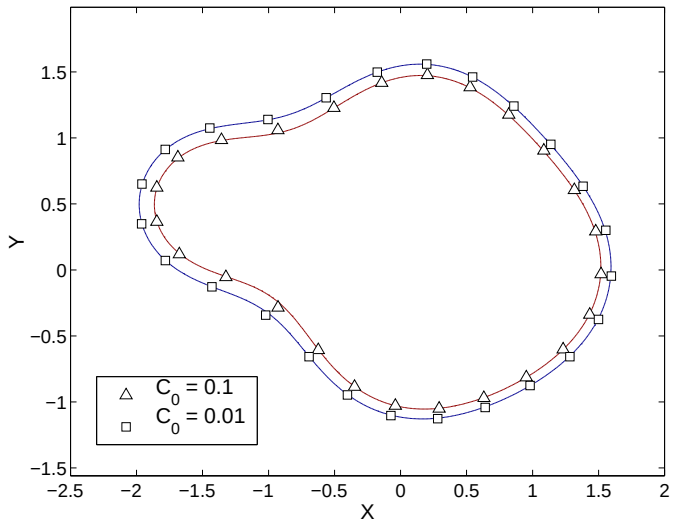

Figure 3: The figure shows the position of agents for two different values of C_0. The squares denote $C_0 = 0.01$, and the triangles, $C_0 = 0.1$. The curves are the actual level curves of $C(v)$.

To model sensor noise, we allow ξ_j to be a Gaussian random variable with zero mean and unit variance. In computing the environmental gradient, noise is added as follows:

$$\begin{aligned} C(x,y) &= \tanh\left(\sum_{i=1}^{4}\exp\left(-\left[(x-x_i)^2+(y-y_i)^2\right]\left[1+\kappa\xi_i\right]\right)-.8\right)\left(1+\kappa\xi_5\right) \\ P(x,y) &= \left(C(x,y)-C_0\right)^2 \\ \nabla P(x,y) &= \begin{pmatrix} \partial_x P(x,y)\left(1+\kappa\xi_6\right) \\ \partial_y P(x,y)\left(1+\kappa\xi_7\right) \end{pmatrix}. \end{aligned} \tag{12}$$

In this model, κ serves as the parameter that determines the amount of noise.

We then define the error as $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(C(v_i)-C_0\right)^2}$, where N is the number of agents, and v_i is the position of the ith agent. We truncated the simulation at $t = 1.5$, so s_i is the position of the ith agent at time 1.5.

To deduce that the algorithm works well in the presence of noise, we contrast the single agent algorithm, (11) using (12) as the environmental function, with the same simulation done for the snake algorithm, (8). Figure 5 shows the results of this contrast. We see that with communication, the noise does not affect the algorithm as much as when no communication is present. The plateau seen for the single agent algorithm demonstrates the complete saturation of the algorithm when $\kappa > 0.6$. An error of 1.0 indicates that the difference between an agent's position and the true level contour is $O(1)$, or that the agent is not accurately locating the boundary.

To see what the agents are actually doing in physical space, we plot the positions of agents for both algorithms with various κ values. Figure 6 shows the agents under the single agent algorithm with four different values of κ. For small values of κ the agents find and track the boundary, but as κ is increased, the agents do not locate the boundary, there is too much noise. Conversely, Figure 7 shows the same simulations with the same κ values. The agents locate the boundary for all four κ values, although, the larger the value of κ, the greater the

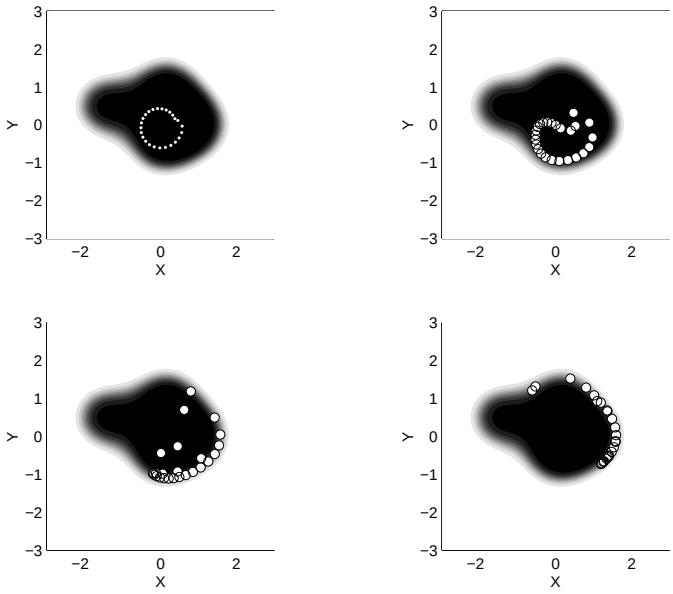

Figure 4: Boundary tracking without communication. The initial conditions for the 25 agents was a circle of radius .25, $\lambda = 2.0$, and $\omega = 2.0$. The four panels show the positions of the agents at times $t = 0.05, 0.25, 0.75$, and 1.5, respectively. The order of the panels are the same as in Figure 1.

difference in the agents' position to that of the true level contour. Nevertheless, the snake algorithm finds the boundary for all four values of κ, showing the benefit of communication for noise compensation/suppression.

8 Conclusions and Future Work

Our first paper [8] constructed a class of a boundary finding algorithms for a group of autonomous mobile robots. The communication required for the algorithm can be either be local or global, depending on frequency of interaction. Explicit update equations require very frequent communication for stability of the method, but only near neighbor interactions are required. Implicit updates require less frequent communication although the entire group must relay their positions for this form of the algorithm. Failure of agents may be overcome, at the cost of accuracy. In the case of extreme failure with only a few agents functioning, a single agent method could take over.

This paper focuses on two main important issues not addressed in the first paper: (1) computing spatial derivatives of the sensor information, and (2) noise or inaccuracy in the sensor data. To address (1), our main idea in this paper was to consider a level set approach for tracking the boundary. Defining the boundary as a set where the concentration dies off enabled us to revise the original algorithm to one in which only first spatial derivatives are needed at the sensor point (i.e. the gradient of the concentration). We also consider Gaussian noise in the sensor signal and its effect on performance of the algorithm.

Problems that are still of interest include topology changes of the plume (i.e. one plume breaks up into two plumes) and time dependent motion of the boundary in general. Also, this paper only addressed the issue of noisy sensor data. In practice, agents' positions will not be known with 100% accuracy. Therefore, the algorithm needs to be shown to work with

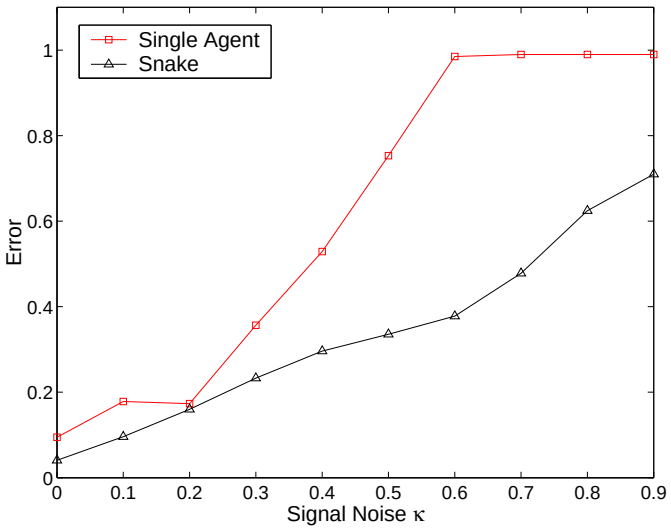

Figure 5: The amount of error as a function of the amount of sensor noise, κ. The left and right panels show the error of the single agent and snake algorithms, respectively, in reaching the 0.01 level set as a function of κ. The parameters for both simulations were $\lambda = 1.0$, and $\omega = 2.0$.

inaccuracies in agent position.

We foresee modeling the topology change analogously to that of t-snakes, or topology changing snakes [9]. We cannot hope to be able to find and track the boundary if the boundary is changing faster than the agents can move, so an initial assumption is that the speed of the boundary be less than that of the slowest agent.

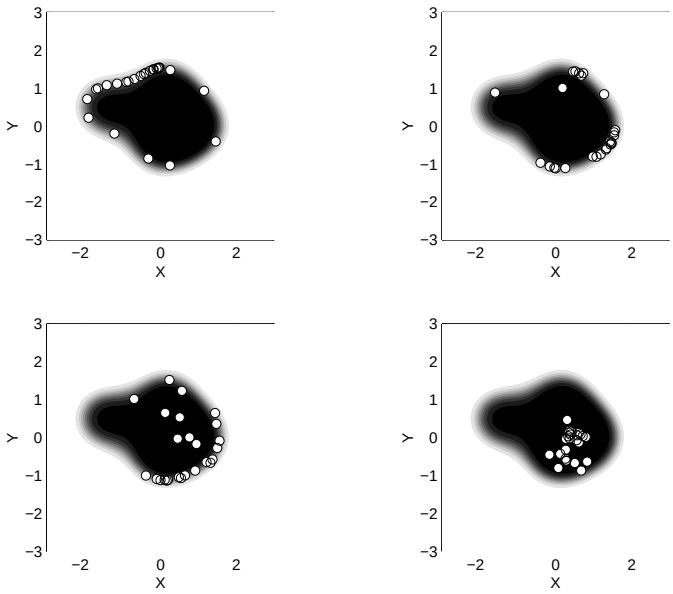

Figure 6: Agent positions for different values of κ for the single agent algorithm with $C_0 = 0.01, \lambda = 1.0$, and $\omega = 2.0$. The panels are arranged as follows: the upper-left panel has $\kappa = 0$, the upper-right panel, $\kappa = 0.2$, the lower-left panel has $\kappa = 0.4$, and the lower-right panel has $\kappa = 0.6$. All of the panels show the positions of the agents when $t = 1.5$.

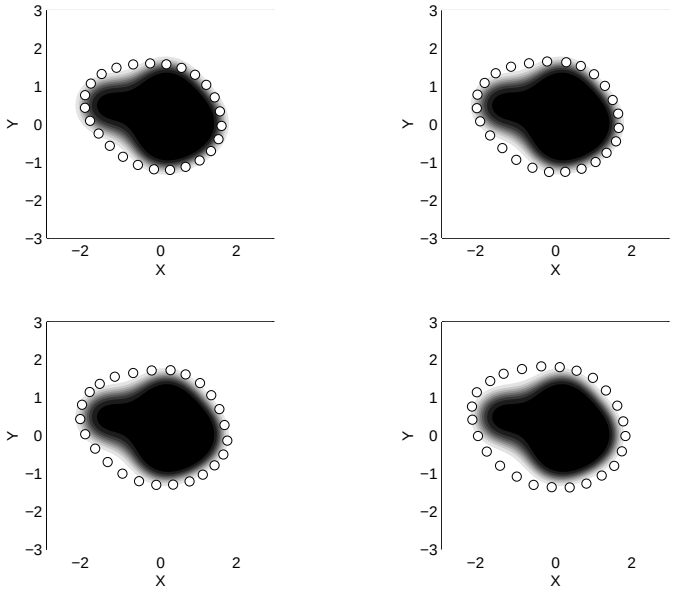

Figure 7: The four panels show the agent positions at $t = 0.5$ for the snake algorithm with different amounts of noise, κ. The upper-left, upper-right, lower-left, and lower-right panels have respective κ values $0, 0.2, 0.4$, and 0.6. Each panel shows the agent positions when $t = 0.5$. The parameters for the simulation were $C_0 = 0.01, \lambda = 1.0$, and $\omega = 2.0$.

Acknowledgments

We thank Matthieu Kemp for useful conversations. This research is supported by ARO grant DAAD19-02-1-0055 and ONR grant N000140310073.

References

[1] J.Desai, V.Kumar, and J.Ostrowski, A Theoretical Framework for Modeling and Controlling Formations of Mobile Robots. IEEE Trans. on Robots and Automations, (under review).

[2] P.Ogren, E.Fiorelli, and N.Leonard, Formations with a Mission: Stable Coordination of Vehicle Group Maneuvers. Proc. 15th Int'l Symposium on Math. Theory of Networks and Systems, (2002).

[3] T.Estlin, G.Rabideau, D.Mutz, and S. Chien, using Continuous Planning Techniques to Coordinate Multiple Rovers. Elec. Trans. on AI, 4:45-57 (2000).

[4] L. Cohen, On Active Contour Models and Balloons. *Comput. Vision, Graphics, and Image Processing: Image Understanding* vol 53 no 2 (1991).

[5] M. Kass, A. Witkin, and D. Terzopoulos, Snakes: Active Contour Models. *International J. of Computer Vision*, 321-331 (1988).

[6] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, Collaborative multi-robot exploration. *Autonomous Robots* vol. 8 no. 3 (2000).

[7] R. Cassinis, Landmine Detection Methods Using Swarms of Simple Robots, *International Conference on Intelligent Autonomous Systems* 6, E. Pagello (Ed.), Venice, Italy (2000).

[8] D. Marthaler and A. Bertozzi, Collective Motion Algorithms for Determining Environmental Boundaries, submitted for publication, 2002.

[9] T. McInerney and D. Terzopoulos, T-Snakes: Topology Adaptive Snakes. medical Image Anal. vol. 4 (2000).

[10] J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Chapman and Hall, New York, 1989.

[11] D. Terzopoulos, Regularization of Inverse Visual Problems Involving Discontinuities. IEEE Transactions PAMI-8, p.413 (1986).

[12] `http://www.nektonresearch.com/`