Managing the Mobility of a Mobile Sensor Network Using Network Dynamics

Ke Ma, Yanyong Zhang, and Wade Trappe
Wireless Information Network Laboratory (WINLAB)
Rutgers University, 73 Brett Rd., Piscataway, NJ 08854.

Abstract

It's been widely discussed in the literature that the mobility of a mobile sensor network can be used to improve the network's sensing coverage. How to efficiently manage the mobility towards a better coverage, however, remains an unanswered question. In this paper, motivated by classical dynamics that govern the movement of particles, we propose the concept of network dynamics and define the associated potential functions that capture the operational goals of a mobile sensor network. We find that, in the context of sensor mobility, Newton's laws of motion are inefficient because they introduce oscillations, and that instead, the equations of motion need to be formulated using the steepest descent method. In order to apply the network dynamics model to actual sensor networks, we have devised a parallel distributed algorithm that runs on each node to guide its movement based on the network dynamics model. The algorithm thus turns mobile sensor nodes into autonomous entities capable of adjusting the location and layout according to the operational goals and environmental changes. Further, we provide a formal proof of the convergence of this algorithm. In order to demonstrate its effectiveness in improving a sensor network's coverage, we applied the network dynamics model and the proposed algorithm in three domains: changing the layout of a mobile sensor network to maximize its sensing coverage; repositioning a mobile sensor network to chase a moving target; and maintaining a mobile sensor network's coverage in the presence of adversarial jammers.

I. Introduction

Sensor networks usually consist of stationary sensor nodes. Deploying stationary sensor networks, however, is a daunting task. For example, to deploy a sensor network in a hostile environment, e.g. a battlefield, it is infeasible to manually place the nodes. Even if more advanced deployment tools, such as airplanes, are available, various complications like wind or obstacles will likely lead to coverage holes regardless of how many sensors are dropped. Even if a perfect deployment is initially achieved, the network may still lose its coverage over some area as time evolves either due to natural causes such as node failures or due to malicious attacks such as jamming attacks. As a result, there is an urgent need that sensors have mobility so that they can autonomously heal the coverage holes after landing. Other situations where mobile sensor nodes are desired include tracking moving objects whose exact trajectory is unpredictable in a large area. Deploying static sensors all over the area is technically possible; however, it is both inefficient and prohibitively costly. For these applications, engaging a mobile sensor network

capable of chasing the object is a much more viable alternative. As a matter of fact, this type of mobile sensor network has already been tested in commission [1], where a fleet of undersea robots work together without human intervention to make measurements of the ocean.

As sensor mobility is becoming increasingly important and available, it is therefore critical to formulate laws that can govern the mobility of the sensors. In light of this, artificial potential functions and virtual forces were first introduced to guide the motion of a mobile device in [2]. Later, the concept of virtual force and potential energy is also used to improve sensing coverage, such as the techniques proposed in [3]–[11]. Most of these approaches, however, implicitly or explicitly follow Newton's laws of motion, which we argue inefficient because they result in unnecessary oscillations (more discussions in Section II). Instead, we take the viewpoint that we should follow the method of steepest descent to formulate sensor motions. In this paper, we formally define a set of models, which we call *network dynamics*, to describe the virtual forces between network nodes as well as the associated potential energy of the entire network. In addition, we also discuss the ideal simulation framework and convergence of network dynamics models, and we point out that steepest descent formulation is necessary to model realistic sensor network scenarios.

In addition to proposing the concept of network dynamics, we also attempt to manage their movement in an autonomous and energy-efficient fashion using the concept of network dynamics. In this end, we propose a parallel distributed movement algorithm to put the laws of network dynamics into effect. In our algorithm, each node periodically calculates the virtual forces it receives from its neighbors based on the distances with all its neighbors. According to the resulting virtual force, a node determines the movement speed and direction in the next interval. By repeating this process, the potential energy of the network keeps decreasing to its minimum. The virtual force and the corresponding potential energy are defined in such a way that the desired network objective such as a better spatial coverage or the tracking of a mobile object will be realized when the potential energy reaches its minimum. We have also formally proved that our algorithm will converge under realistic assumptions.

To demonstrate the effectiveness and generality of our distributed movement algorithm, we have applied it to three important application domains for sensor networks. The first application domain deals with deploying sensor nodes to cover a large area. In this case study, we propose an efficient deployment method: first dropping all the sensor nodes within a small area, and then letting each sensor node follow our distributed movement algorithm to find its real location. We have designed simulations to model this scenario, the results show that our algorithm can achieve efficient deployment under different circumstances including varying node densities. We have also proposed a variation of the algorithm which can shorten the convergence time without noticeably sacrificing the coverage performance. Finally, we have compared our algorithm with a popular class of method and show that our algorithms can successfully deploy nodes in situations that were impossible.

The second application domain is to track mobile objects. For these applications, our movement algorithm can be used to detect the departure of the target, and to chase the target as it moves. We have

also proposed ways to balance the tradeoff between the coverage of the object and the travel distance of the senor nodes. Finally, the third application domain is to dynamically repair network holes that are caused by malicious jammers. A jammer can cause communication holes in the network by blocking the wireless channel. In the presence of jamming, we first propose an efficient retreat strategy so that all the jammed nodes can escape the jammed area. After escaping, these nodes will follow our movement algorithm so that they will uniformly disperse into the rest of the nodes. More importantly, our algorithm can prevent the network from being partitioned by a jammer that sweeps through the network because the nodes will automatically fill the holes after the jammer leaves an area.

The paper is organized as follows. We begin the discussion of the proposed model, network dynamics, in Section II. In Section III, we present a parallel distributed algorithm that can be used to implement network dynamics in practice. Its convergence is formally proved in Section IV. Then we examine three applications of the network dynamics. The first application, presented in Section V, focuses on using network dynamics to control the sensing coverage of mobile sensors over a particular region. The second application in Section VI studies the feasibility of applying network dynamics to enable a mobile sensor network to chase a moving target. In Section VII, we apply network dynamics to achieve a robust spatial retreat strategy that maintains desirable network connectivity in the presence of jamming attacks. Finally in Section VIII and IX, We present related works and concluding remarks respectively.

II. Network Dynamics

A mobile sensor network (MSN) in this paper is a collection of sensor nodes that have mobility, such as unmanned vehicles equipped with sensors. These sensor nodes are operated on battery power, but we assume their batteries are rather long lasting [12]. A MSN may be viewed as a dynamical system – the positions of sensor nodes change with time. The dynamics of classical mechanics systems are described via underlying laws of motion and laws of force between objects [13]. We propose to apply the concept of forces, the corresponding notion of potential energy, and the laws of motion to manage the movement of a mobile sensor network. Appropriately defining potential functions and forces, yields a general framework that allows one to optimally use mobility to govern the operations of a mobile sensor network.

A. Classical Dynamics and Mobile Sensor Networks

We shall look at a MSN as a dynamical system of N devices subject to the laws of classical mechanics. Each device will be able to communicate with its neighbors through some wireless communication protocol. Since the devices are mobile, we will associate with each device j a position vector $\mathbf{p}_j$ and a momentum vector $\mathbf{q}_j$. We will, for simplicity, assume that all devices are located in two-dimensions and that, for each device, both $\mathbf{p}_j$ and $\mathbf{q}_j$ are two-dimensional vectors. For the purpose of our discussion, since we are looking at the system as a mechanical system, we will arbitrarily assign each network device a mass of 1, thus momentum and velocity are equivalent.

N-body dynamical systems appear commonly in physics in order to model classical mechanical systems, such as from gravitational modeling or in electrostatics. In these problems, the dynamical relationship between position and momentum of these N bodies evolves based upon Newton's second law, which describes the motion of a body in the presence of a field of force.

The forces acting upon a conservative dynamical system arise as the negative gradient of the potential energy function U. This potential function may be comprised of two components: *external* U_{ext} and *internal* U_{int}. External potentials arise from externally applied forces, while internal potentials correspond to the attractive or repulsive forces between bodies of the system. Typically, the internal potentials are restricted to two-body interactions, such as the gravitational pull between two objects.

We may collectively refer to the position vector of each of the N bodies by a $2N$-dimensional position vector $\mathbf{p} = [\mathbf{p}_1, \cdots, \mathbf{p}_N]$, and similarly for the momentum vector. Hence, the potential energy may be viewed as

$$U(\mathbf{p}) = \sum_j U_{ext}(\mathbf{p}_j) + \sum_i \sum_{j>i} U_{int}(\mathbf{p}_i, \mathbf{p}_j). \tag{1}$$

Newton's classical equations of motions give

$$\frac{d}{dt}\mathbf{q}_j = \mathbf{f}_j, \qquad \mathbf{f}_j = -\nabla_j U(\mathbf{p}) \tag{2}$$

where ∇_j is the gradient at the j-th body's position $\mathbf{p}_j$. Applying the relationship between position and velocity, $\frac{d}{dt}\mathbf{p}_j = \mathbf{q}_j$, yields a set of coupled differential equations describing the dynamics of the N bodies.

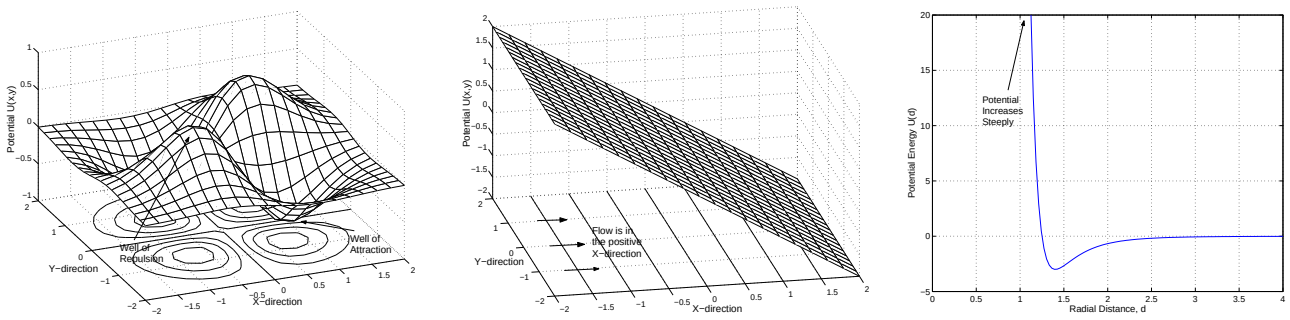
B. Potential Functions for Network Dynamics

Mobility of bodies is governed by the description of the potential function $U(\mathbf{p})$, which in turn yields force and causes motion. We therefore need to define potential functions that suitably capture the need for causing mobile devices to move. We will do this in two parts: first describing possible external potential functions $U_{ext}(\mathbf{p})$, and then describe internal (i.e. pairwise) potential functions $U_{int}(\mathbf{p}_i, \mathbf{p}_j)$.

External potential functions may be viewed as representing factors coming from the environment that should influence the motion of a MSN. For example, suppose that a MSN has been deployed to perform monitoring functions. It may be that there are regions of the network that deserve more attention than other regions, and therefore we might wish for mobile devices to be attracted to these regions. As another example, it might be desirable for a set of monitoring devices to migrate, perhaps to monitor a moving asset or perhaps to sweep through a coverage area. We present such potentials in Figure 1 (a) and (b).

Internal potentials typically correspond to either attractive or repulsive pairwise interactions between entities. An attractive potential is the gravitational potential

$$U(\mathbf{p}_i, \mathbf{p}_j) = \frac{-G}{\|\mathbf{p}_i - \mathbf{p}_j\|} \tag{3}$$



(a) (b) (c)

Fig. 1. (a) The potential function U indicating basins of attraction and repulsion, (b) the potential function U encouraging a flow in the positive x direction, and (c) the Lennard-Jones potential.

where G is the gravitational constant and we have taken the masses to be 1. An example of a repulsive potential is Coulomb's potential from electrostatics:

$$U(\mathbf{p}_i, \mathbf{p}_j) = \frac{Q_i Q_j}{4\pi\epsilon_0 \|\mathbf{p}_i - \mathbf{p}_j\|} \tag{4}$$

where Q_i and Q_j are charges and ϵ_0 is the permittivity of free space. In general, we desire potential functions that are capable of dispersing mobile sensors without causing them to separate too greatly from each other. These potential functions, often known as dispersive potentials, involve repulsive and attractive components. An example of such a potential function is the Lennard-Jones potential from thermophysics [14]

$$U(\mathbf{p}_i, \mathbf{p}_j) = 4\epsilon \left[\left(\frac{\sigma}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right)^{12} - \left(\frac{\sigma}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right)^{6} \right] \tag{5}$$

where σ describes the radial intercept, and ϵ governs the *well depth*, as depicted in Figure 1 (c).

C. Ideal Simulation Framework and Convergence

It is not only unreasonable to expect that a continuous-time representation of system potential function to be available, but it is also generally intractable to explicitly solve an arbitrary system of equations that would describe the system's motion. Therefore, we envision that the use of network dynamics will involve discrete time steps. In the following, we will examine the natural discretization of Newton's equations of motion, and argue that such a formulation leads to mobile devices performing extra mobility. To address this concern, we propose to formulate the equations governing network dynamics using steepest descent optimization methods.

Suppose we have a system of N objects, and consider the evolution of the N objects' position in time n. Here, we will represent time discretely by breaking time into intervals of length T. A typical discretization involves updating the i-th object's position via

$$\mathbf{p}_i(n+1) = \mathbf{p}_i(n) + \mu_p \mathbf{q}_i(n) \tag{6}$$

$$\mathbf{q}_i(n+1) = \mathbf{q}_i(n) + \mu_q \mathbf{f}_i(n), \tag{7}$$

where μ_p and μ_q are step sizes, and the force $\mathbf{f}_i(n) = -\nabla_i U(\mathbf{p}(n))$ may be determined at each time step. The time-evolution of the MSN's motion is then determined by letting the above system evolve.

The classical equations of motion are second-order in time, meaning that the coupled equations tie in position, velocity and acceleration. This is suitable for mechanics, but results in undesirable properties from the point-of-view of network dynamics. Specifically, the coupling between position, velocity and acceleration implies that it is quite likely that, even when the system has reached a configuration with minimal potential energy, the N bodies will still have velocity, and hence kinetic energy. As a result, the system will escape its desirable configuration and have to compensate later by applying forces in the opposite direction. To visualize this, simply consider the classical pendulum, in which the pendulum achieves its minimal potential at the base of the trajectory, the momentum causes the pendulum to swing past and increase potential energy. The increased potential causes the pendulum to reverse direction and oscillate around the point of minimal potential.

From the viewpoint of network dynamics, this behavior is undesirable as it means that the mobile device must waste movement, and hence power, compensating for momentum. We therefore, would like to modify the equations of motion to remove the effect of momentum. This may be accomplished by making the equations first-order, and have the force affect the distance traveled. For example, we may simply create an iterative system

$$\mathbf{p}(n+1) = \mathbf{p}(n) - v\nabla U(\mathbf{p}(n)). \tag{8}$$

Examining this iteration, we see that we are simply making a step of size v in the direction of steepest descent to minimize the potential function U. Now, once the devices have moved into a configuration with minimal potential U, they stop moving and don't move unless a disruption is introduced that necessitates the relocation of them. For suitable choice of v, convergence will follow from the convergence of steepest descent methods. In the following sections, where we discuss distributed versions of network dynamics, we shall use our steepest descent formulation of motion as the starting point for our algorithms, and will discuss the selection of v.

III. Distributed Implementations of Network Dynamics

In the previous section, we presented a generic way of modeling node mobility using network dynamics. In this section, we discuss how to map network dynamics onto an actual network and implement it in a completely distributed fashion with realistic constraints.

A. Overview of Distributed Network Dynamics

In Section II, we discussed the discretization of network dynamics. The straight-forward way to formulate network dynamics involves a centralized controller with knowledge of each device's position and the ability to communicate its directives to each mobile device. In practice, however, such a formulation

is unrealistic as MSNs, by their inherently ad hoc nature, do not have a centralized infrastructure. Consequently, we must devise a set of distributed algorithms whereby each node makes decisions based on its local environment in an attempt to achieve the minimum system potential energy. Although there are different ways of implementing distributed network dynamics, there are some common features amongst these different schemes. A node can only "feel" the forces from the nodes that are within its radio range. Every node j periodically calculates the total force $\mathbf{f}_j$ from all its neighbors. If the magnitude of the total force is above a threshold, i.e. $\|\mathbf{f}_j\| > \delta$, then node j will start moving in the direction of $\mathbf{f}_j$. While moving, the node will broadcast its location information and receive location updates from its neighbors if necessary, thereby allowing each node to periodically calculate its new force and adjust its movement during the next time slice. The system dynamics proceeds until each node achieves a net force below the threshold δ. We note that, though neighboring nodes need to exchange their location information, this may not introduce much extra overhead because a node can piggyback the location information to background heartbeat packets.

System Model: In the effort of formulating and implementing the distributed network dynamics algorithm, we have made the following assumptions regarding the underlying MSN:

- *2D Deployment:* The network is deployed on a 2D plane, and as a result the movement of the nodes are also constrained on the 2D plane.
- *Mobility and its limitations:* Nodes can move, but the mobility is limited both by the maximum speed at which a node can move and by the total distance a node can move because movement in general consumes a large amount of energy.
- *Locations Known:* Every node knows its own location. This can be achieved by devices such as GPS [15], or through various wireless localization algorithms [16].

Performance Metrics: We propose the following metrics to evaluate the proposed distributed algorithms:

- *Movement efficiency:* Sensor nodes will experience different movement trajectories as a result of following the movement algorithm. To measure the efficiency of these trajectories, we add up the total distance travelled by all the sensors; a shorter travel distance indicates a lower energy consumption and hence a better movement efficiency.
- *Convergence time:* When the network dynamics algorithm converges, every node within the network will have a force which is below the specified threshold. At the same time, the system potential energy U will have reached its minimum. In general, a shorter convergence time is preferred.

B. Distributed Network Dynamics Algorithm

A naive way of implementing the distributed network dynamics algorithm is a sequential approach, in which the nodes within a neighborhood of each other allow only a single node to move. While this node is moving, the other neighbor nodes in its radio range must remain stationary. Though easy to

```
Algorithm: Parallel Movement Algorithm
while (1) do
    f = calculateForce (my_location, neighbor_location);
    if (||f|| > δ) then
        calculateNewPos (my_location, f);
        moveToNewPos;
        send(my_location);
    else
        wait(T);
    end
end
```

Algorithm 1: Parallel Distributed Network Dynamics Algorithm.

implement, this approach suffers from inefficiencies due to its sequential nature. Additionally, it limits the amount of devices that may move at any time, which will lead to slow convergence. In order to address these limitations, we adopt a parallel version of network dynamics algorithm, which we call the Parallel Distributed Network Dynamics (PDND) algorithm.

In PDND, any node that intends to move can start movement immediately, and therefore, multiple nodes may move simultaneously. A moving node advertises its location every T time, while a stationary node updates its location information at a much coarser granularity. Every node maintains a neighbor table which records each neighbor's location. Note that this information may not be up-to-date. Based on the neighbor table, each node calculates the total force using its neighbors' positions. If a node's force is greater than the threshold, then that node will move along the direction of the force for T time. After T time, it sends out its new location, re-calculates the force, and determines whether it needs to move in the next time slice, and the direction if it needs to. It stops moving when the total force it receives is below the threshold δ. Every node repeats this process iteratively until all the nodes reach steady state.

The PDND algorithm is summarized in Algorithm 1. For this algorithm, the location update interval T is an important parameter, for a coarse T may make sensors oscillate. However, we argue that since the communication among sensors happens in the scale of microseconds, T can be easily set to a small value that its effect on the convergence is negligible.

IV. The Convergence of the Parallel Distributed Network Dynamics Algorithm

In this section, we provide the formal proof of the convergence of the PDND algorithm. We first prove its convergence on a convex sensing field, and then extend the proof to cases with non-convex sensing fields. We assume that there are totally N sensors, indexed by $1, ..., N$, and that their effective sensing ranges, as well as their communication ranges, are identical discs. In what follows, we use r_s to represent the common sensing radius and r_c to represent the common communication radius. We use $\mathbf{p_i} = [x_i \ y_i]^T$ to represent the coordinates of sensor i and use $\mathbf{p}$ to denote the coordinates of all the sensor nodes, with $\mathbf{p} = [\mathbf{p_1}^T \ \cdots \ \mathbf{p_N}^T]^T$. Further, $\mathbf{P}$ denotes all the feasible choices of $\mathbf{p}$. $\mathbf{f_{ij}} = [f_{ij,x} \ f_{ij,y}]^T$ denotes the

virtual force placed on sensor i by sensor j $(j \neq i)$, and therefore, the total force on sensor i can be formulated as $\mathbf{f_i} = \sum_{j=1, j\neq i}^{N} \mathbf{f_{ij}}$. Finally, we use r_f to denote the maximum distance at which two sensors have a force between them.

As nearby sensors have virtual forces with each other, the whole network possesses a virtual potential energy $U(\mathbf{p})$, and $-\nabla U(\mathbf{p}) = \mathbf{f} = [\mathbf{f_i}^T \ \cdots \ \mathbf{f_N}^T]^T$ according to physical laws. The problem of repositioning sensors, with the help of $U(\mathbf{p})$, can be transformed into the following optimization problem:

$$\begin{aligned} \min \quad & U(\mathbf{p}), \\ \text{subject to} \quad & \mathbf{p} \in \mathbf{P}. \end{aligned}$$

It should be noticed that this optimization problem is not convex. This is because of the fact that given any optimal solution $\mathbf{p}^*$, one can obtain another optimal solution $\mathbf{p}'$ by exchanging the positions of any two sensors in $\mathbf{p}^*$.

The PDND algorithm proposed in Section III is similar to the standard gradient projection algorithm, which is formulated as

$$\mathbf{p}(k+1) = \Big[\mathbf{p}(k) - \gamma(k)\nabla U\Big(\mathbf{p}(k)\Big)\Big]^+, \tag{9}$$

where $\gamma(k)$ is a positive step size at the k-th iteration of the algorithm and $[\mathbf{p}]^+$ is the orthogonal projection (with respect to the Euclidean norm) defined by

$$[\mathbf{p}]^+ = \arg\ \min_{\mathbf{p}' \in \mathbf{P}} \|\mathbf{p}' - \mathbf{p}\|_2. \tag{10}$$

Although the standard gradient projection algorithm is a suitable numerical method that can be used to find solutions for optimization problems, it is inappropriate for our problem for the following two reasons. First, it needs the global information to find out the appropriate step size by carrying out a line search algorithm in each iteration. Second, by adopting a single $\gamma(k)$ for all sensors, it does not take into account the speed limit of the sensors. As a result, during the time interval of the k-th iteration, a sensor i may be asked to travel a distance far beyond its capability. To address the second shortcoming, one can either increase the locomotion capability of the senors or extend the duration of each iteration. Both options, however, have drawbacks: the former is not always realistic while the latter may significantly slow down the convergence of the algorithm. *Instead, we propose a better choice, the PDND algorithm, which modifies the standard gradient projection algorithm by letting each sensor independently choose its own step size based on the local information*. The PDND algorithm is described by the following equation:

$$\mathbf{p}(t+\Delta t) = \Big[\mathbf{p}(t) - \mathbf{diag}(\gamma(t))\nabla U\Big(\mathbf{p}(t)\Big)\Big]^+, \tag{11}$$

where $\gamma(t) = [\gamma_1(t)\ \gamma_1(t)\ \ldots\ \gamma_i(t)\ \gamma_i(t)\ \ldots\ \gamma_N(t)\ \gamma_N(t)]^T \succeq 0$.

Before discussing the convergence of the PDND algorithm, we first present the assumptions we have made about the considered systems. We would like to note that these assumptions, on the other hand, will not make the considered system less realistic.

Assumption 1: $r_f < r_c$.

This assumption eases the implementation of the proposed algorithm because sensor i only needs the position information of its communication neighbors to calculate $\mathbf{f_i}$.

Assumption 2 (Lipschitz Continuity of $\mathbf{f_{ij}}$*):* there exists a constant C such that

$$\|\mathbf{f'_{ij}} - \mathbf{f_{ij}}\|_2 \leq C\left\|[\mathbf{p_i'}^T\ \mathbf{p_j'}^T]^T - [\mathbf{p_i}^T\ \mathbf{p_j}^T]^T\right\|_2. \tag{12}$$

Under this assumption, the force between two sensors is a bounded continuous function of the distance between them. We now present some theoretical results describing the convergence of PDND.

Proposition 1: $U(\mathbf{p})$ is bounded below for every feasible $\mathbf{p}$.

Proof: Given that $-\nabla U(\mathbf{p}) = \mathbf{f}$ and Assumption 2, this is obvious. **Q.E.D.**

Proposition 2 (Lipschitz Continuity of $\nabla U(\mathbf{p})$*):* $U(\mathbf{p})$ is continuously differentiable and there exists a constant K such that

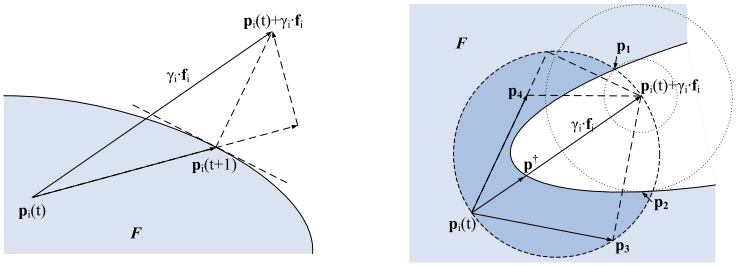
$$\left\|\nabla U(\mathbf{p}') - \nabla U(\mathbf{p})\right\|_2 \leq K\|\mathbf{p}' - \mathbf{p}\|_2, \tag{13}$$

where $\mathbf{p}', \mathbf{p} \in \mathbf{P}$.

Proof: Let $N(i)$ be the set consisting of every sensor j $(j \neq i)$ that satisfies either $\|\mathbf{p_i'} - \mathbf{p_j'}\|_2 \leq r_f$ or $\|\mathbf{p_i} - \mathbf{p_j}\|_2 \leq r_f$, and L be the maximum set size among all the sensors, i.e. $L = \mathbf{max}_i\ |N(i)|$. The fact that $\mathbf{f_{ij}}$ is Lipschitz continuous indicates that there exists a constant C such that $\|\mathbf{f'_{ij}} - \mathbf{f_{ij}}\|_2 \leq C\left\|[\mathbf{p_i'}^T\ \mathbf{p_j'}^T]^T - [\mathbf{p_i}^T\ \mathbf{p_j}^T]^T\right\|_2 \leq C(\triangle d_i + \triangle d_j)$, where $\triangle d_i = \|\mathbf{p'_i} - \mathbf{p_i}\|_2$. Therefore, the constant K can be found in the following way:

$$\begin{aligned}
&\left\|\nabla U(\mathbf{p}') - \nabla U(\mathbf{p})\right\|_2^2 && (14)\\
=\ & \sum_{i=1}^{N}\left[\left(f'_{i,x} - f_{i,x}\right)^2 + \left(f'_{i,y} - f_{i,y}\right)^2\right] && (15)\\
=\ & \sum_{i=1}^{N}\left[\left(\sum_{j\in N(i)}\left(f'_{ij,x} - f_{ij,x}\right)\right)^2 + \left(\sum_{j\in N(i)}\left(f'_{ij,y} - f_{ij,y}\right)\right)^2\right] && (16)\\
\leq\ & L\sum_{i=1}^{N}\left[\sum_{j\in N(i)}\left(f'_{ij,x} - f_{ij,x}\right)^2 + \sum_{j\in N(i)}\left(f'_{ij,y} - f_{ij,y}\right)^2\right] && (17)\\
\leq\ & LC^2\sum_{i=1}^{N}\sum_{j\in N(i)}(\triangle d_i + \triangle d_j)^2 && (18)\\
\leq\ & 2LC^2\sum_{i=1}^{N}\sum_{j\in N(i)}(\triangle d_i^2 + \triangle d_j^2) && (19)\\
\leq\ & 4L^2C^2\sum_{i=1}^{N}\triangle d_i^2 && (20)\\
=\ & 4L^2C^2\|\mathbf{p}' - \mathbf{p}\|_2^2. && (21)
\end{aligned}$$

As a result, $K = 2LC$. **Q.E.D.**



(a) (b)

Fig. 2. (a) Illustration of the orthogonal project on a convex sensing field; (b) Illustration of the moving strategy on a non-convex sensing field.

Lemma 1: $U(\mathbf{p}') \leq U(\mathbf{p}) + (\mathbf{p}'-\mathbf{p})^T\nabla U(\mathbf{p}) + \frac{K}{2}\|\mathbf{p}'-\mathbf{p}\|_2^2,\ \forall \mathbf{p}', \mathbf{p} \in \mathbf{P}$.

Proof: Given in [17]. **Q.E.D.**

Lemma 2 (Properties of the PDND algorithm on a convex set):

1) $U\Big(\mathbf{p}(t+\Delta t)\Big) \leq U\Big(\mathbf{p}(t)\Big) - \Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)^T \mathbf{diag}(1/\gamma(t)-K/2)\Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)$.

2) $\mathbf{p}(t+\Delta t)=\mathbf{p}(t)$ iff $\Big(\mathbf{p}'-\mathbf{p}(t)\Big)^T\nabla U\Big(\mathbf{p}(t)\Big) \geq 0$ for all feasible $\mathbf{p}'$.

3) The mapping $\Big[\mathbf{p}(t)-\mathbf{diag}(\gamma(t))\nabla U\Big(\mathbf{p}(t)\Big)\Big]^+$ is continuous.

Proof: Only the proof of the first property is given here, and the rest is almost identical to that of the standard gradient project algorithm provided in [17].

From the definition of the projection method, we know that for each i (see Fig. 2(a)),

$$\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\gamma_i(t)\mathbf{f_i}(t) \geq \Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big), \tag{22}$$

$$\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\mathbf{f_i}(t) \geq \frac{1}{\gamma_i(t)}\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big). \tag{23}$$

Therefore,

$$-\sum_i\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\mathbf{f_i}(t) \leq -\sum_i\frac{1}{\gamma_i(t)}\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big)^T\Big(\mathbf{p_i}(t+\Delta t)-\mathbf{p_i}(t)\Big), \tag{24}$$

$$\Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)^T\nabla U\Big(\mathbf{p}(t)\Big) \leq -\Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)^T\mathbf{diag}(1/\gamma(t))\Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big). \tag{25}$$

Applying Lemma 1, we get

$$\begin{aligned} U\Big(\mathbf{p}(t+\Delta t)\Big) &\leq U\Big(\mathbf{p}(t)\Big) + \Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)^T\nabla U\Big(\mathbf{p}(t)\Big) + \frac{K}{2}\Big\|\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big\|_2^2 & (26)\\ &\leq U\Big(\mathbf{p}(t)\Big) - \Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big)^T\mathbf{diag}(1/\gamma(t)-K/2)\Big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\Big). & (27)\end{aligned}$$

Q.E.D.

Theorem 3 (Convergence of the PDND Algorithm on a convex set): If $0 < \max_i\ \gamma_i(t) < 2/K$ and $\mathbf{p}^*$ is a limit point of the sequence $\{\mathbf{p}(t)\}$ generated by the PDND algorithm, $(\mathbf{p}-\mathbf{p}^*)^T\nabla U(\mathbf{p}^*) \geq 0$, for all feasible $\mathbf{p}$.

Proof: Let $\Big\{\mathbf{p}(t)\Big\}$ be the sequence generated by the PDND algorithm, where t can be $0, \Delta t, 2\Delta t, \cdots$. We first consider the situation where $\gamma_i(t) > 0$ for all i. The condition, $0 < \max_i \gamma_i(t) < 2/K$, guarantees the matrix $\mathbf{diag}(1/\gamma(t) - K/2)$ is positive definite. Applying this observation to Lemma 2.1, we get the conclusion that the sequence $\Big\{U\Big(\mathbf{p}(t)\Big)\Big\}$ is strictly decreasing unless $\mathbf{p}(t+\Delta t) = \mathbf{p}(t)$. Further, as $U(\mathbf{p})$ is bounded below, this sequence converges.

When at least one $\gamma_i(t) = 0$, we may puncture both $\mathbf{diag}(1/\gamma(t) - K/2)$ by removing all zeros on the diagonal, and $\Big(\mathbf{p}(t+\Delta t) - \mathbf{p}(t)\Big)$ by removing all corresponding zeros. Applying the same reasoning as before completes the proof. **Q.E.D.**

We have proved that under minor assumptions, the PDND algorithm converges on a convex sensing field. Next, we study its convergence in a non-convex area. In such scenarios, the orthogonal projection method can no longer be used to calculate sensor locations. In order to address this difficulty, we propose a simple but efficient movement regulation strategy. When a sensor i approaches the boundary of the sensing field, it regulates its movement in the following way: if the target position $\big(\mathbf{p_i}(t) + \gamma_i(t)\mathbf{f_i}(t)\big)$ is outside the sensing field, sensor i moves to the location within the sensing field that is the closest one in its vicinity to the target location. Specifically, sensor i's movement must be inside or on the circle centered at $\big(\mathbf{p_i}(t) + \gamma_i(t)\mathbf{f_i}(t)/2\big)$ with the radius $\gamma_i(t)\mathbf{f_i}(t)/2$. Figure 2(b) illustrates this strategy. Sensor i first moves toward its target location $\big(\mathbf{p_i}(t) + \gamma_i(t)\mathbf{f_i}(t)\big)$ until it hits the boundary of the sensing field at $p^\dagger$, and then, it randomly picks a direction and moves along the boundary until it reaches a point that is closest to the target location in that particular direction, e.g. $\mathbf{p_1}$ or $\mathbf{p_2}$ in Figure 2(b). Additionally, if it has a priori knowledge of the shape of the boundary, it can calculate the point that is closest to the target position, e.g. $\mathbf{p_1}$ in Figure 2(b), and move to that point following the optimal path.

Lemma 3 (Properties of the PDND algorithm on a non-convex set):
$U\Big(\mathbf{p}(t+\Delta t)\Big) \leq U\Big(\mathbf{p}(t)\Big) - \Big(\mathbf{p}(t+\Delta t) - \mathbf{p}(t)\Big)^T \mathbf{diag}(1/\gamma(t) - K/2)\Big(\mathbf{p}(t+\Delta t) - \mathbf{p}(t)\Big)$.

Proof: Any position that is inside or on the circle centered at $\big(\mathbf{p_i}(t) + \gamma_i(t)\mathbf{f_i}(t)/2\big)$ with the radius $\gamma_i(t)\mathbf{f_i}(t)/2$, e.g. $\mathbf{p_3}$ or $\mathbf{p_4}$ in Figure 2(b), satisfies the following inequality:

$$\Big(\mathbf{p_i}(t+\Delta t) - \mathbf{p_i}(t)\Big)^T \gamma_i(t)\mathbf{f_i}(t) \geq \Big(\mathbf{p_i}(t+\Delta t) - \mathbf{p_i}(t)\Big)^T \Big(\mathbf{p_i}(t+\Delta t) - \mathbf{p_i}(t)\Big). \tag{28}$$

According to the aforementioned movement regulation strategy, $\mathbf{p_i}(t+\Delta t)$ is such a position. Starting from Equation 22, the rest of the proof is the same as that of Lemma 2.1, and hence omitted. **Q.E.D.**

Theorem 4 (Convergence of the PDND Algorithm on a non-convex set): If $0 < \max_i \gamma_i(t) < 2/K$, the PDND algorithm converges.

Proof: Let $\Big\{\mathbf{p}(t)\Big\}$ be the sequence generated by the PDND algorithm. When all $\gamma_i(t)$ are greater than zero, from the condition $0 < \max_i \gamma_i(t) < 2/K$, we find that $\mathbf{diag}(1/\gamma(t) - K/2)$ is positive definite. Applying this observation to Lemma 3, we can conclude that the sequence $\Big\{U\Big(\mathbf{p}(t)\Big)\Big\}$ is strictly decreasing, and since $U(\mathbf{p})$ is bounded below, this sequence converges. When there exists at least

one $\gamma_i(t)$ that is equal to zero, the convergence can be proved using the same strategy in the proof of Theorem 3. **Q.E.D.**

V. Case Study I: Spatial Coverage

In the first case study, we explore the applicability of network dynamics to the problem of maximizing the spatial coverage of a mobile sensor network.

A. Problem Statement

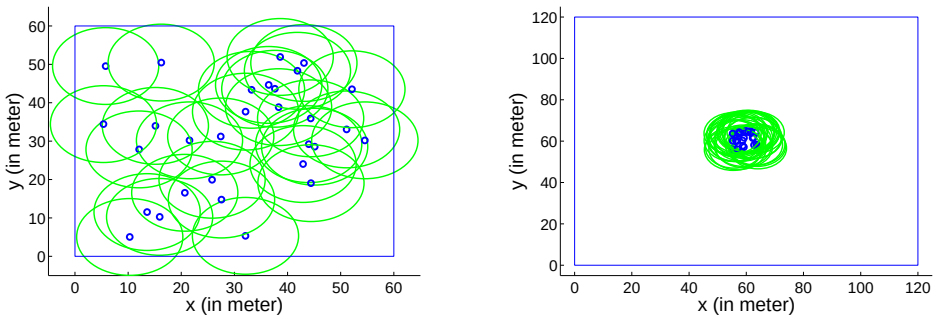
To set up the problem, let us consider the scenario in which a mobile sensor network is deployed to monitor a particular region (sensing field). It is often difficult, if at all possible, to initially deploy the sensors in such a way that the maximum coverage is achieved. On the other hand, the ease of dropping sensors over a smaller region, or randomly over the whole sensing field, suggests that we adopt a two-phase strategy that involves first randomly dumping the sensor nodes and then letting the sensors adjust their positions to better cover the area. Furthermore, even if it is possible to initially place sensors to achieve the maximum coverage, it is still desirable to re-configure their positions on the fly because sensors may fail during the operation.

The proposed PDND algorithm aims to make a mobile sensor network an autonomous entity that always tries to maximize its sensing coverage.

Coverage Degree: The most important metric for this problem domain is the coverage degree, which measures the ratio of the entire sensing field that is covered. PDND intends to maximize the coverage degree of a given network by coordinating the movement of the sensors, and as a final result, sensors should be sufficiently separated from each other to maximize coverage, but at the same time, sufficiently close to each other to stay connected. Before presenting how to measure the coverage degree, we first introduce the two terms that are associated with each sensor: coverage range and Voronoi cell. In this study, we assume a simple disc coverage model, in which a sensor can cover a circular area (referred to as coverage range) with radius r_s centered at the sensor itself. After all sensors are deployed on the field, each sensor has a Voronoi cell, a generalized polygon whose interior consists of all points in the plane which are closer to that sensor than to any other. In order to calculate the overall coverage degree, we adopt a divide-and-conquer strategy: we first divide the whole sensing field into Voronoi cells based on the positions of the sensors; then, we let each sensor calculate what fraction of its own Voronoi cell is covered (by comparing the area of the Voronoi cell and the coverage range), which is a simple geometric problem and the details can be found in [18].

Force Model: Although any force model that satisfies Assumption 2 can be used, we choose the following one because of its simplicity:

$$\mathbf{f_{ij}} = \left[(r^* - d_{ij})I_t(d_{ij}) + \frac{(r^*-r_t)(r_f-d_{ij})}{r_f-r_t}\Big(I_f(d_{ij}) - I_t(d_{ij})\Big)\right]\mathbf{u_{ij}}. \tag{29}$$



(a) Case 1 (b) Case 2

Fig. 3. Initial topologies.

The notations in the foregoing equation are explained as follows:

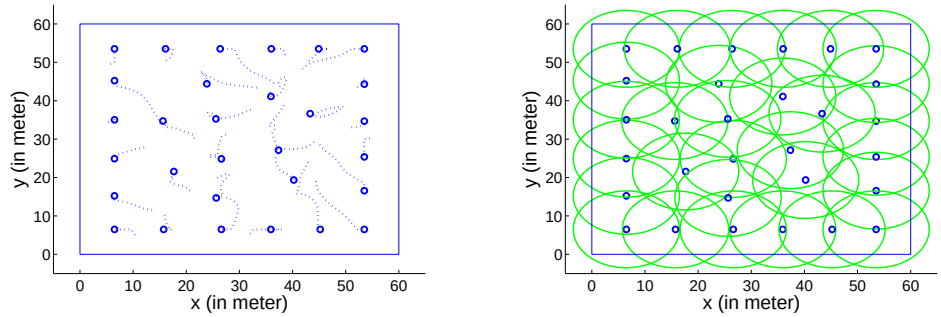
- r^* is the distance between two sensors when the force between them is zero. As the distance becomes shorter (or longer), they start to repel (or attract) each other.
- r_f $(> r^*)$ is the distance beyond which the attractive force between two sensors vanishes.
- r_t $(r^* < r_t < r_f)$ is the distance at which two sensors attract each other most. The attractive force drops off as the distance decreases or increases.
- $d_{ij} = \|\mathbf{p_i} - \mathbf{p_j}\|_2$.
- $\mathbf{u_{ij}} = \Big(\mathbf{p_i} - \mathbf{p_j}\Big)/d_{ij}$.
- $I_t(d) = \begin{cases} 1 & \text{if } d \le r_t \\ 0 & \text{otherwise} \end{cases}$, $I_f(d) = \begin{cases} 1 & \text{if } d \le r_f \\ 0 & \text{otherwise.} \end{cases}$.

The constant C in Assumption 2 of this force model can be easily verified to be $\sqrt{2}$.

B. Simulation Results

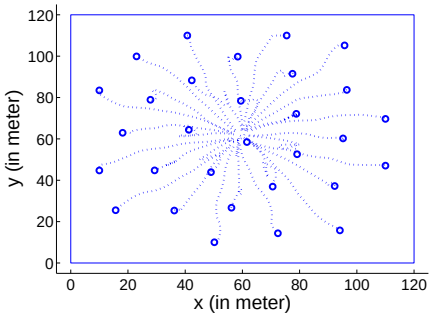
In this exercise, we conduct detailed simulation studies to examine the effectiveness and efficiency of PDND in improving sensor network's spatial coverage. In addition, we develop a variation of PDND that utilizes a more relaxed convergence criterion. Finally, we also compare the performance of the two PDND algorithms with an Voronoi diagram based movement algorithm.

In our simulation studies, we consider two random initial deployments involving 30 sensors, one over a $60 \times 60m^2$ area and the other over a $120 \times 120m^2$ area, which are illustrated in Figures 3(a) and (b). We use these two cases to represent two typical random deployment strategies: case 1 represents the deployments where the sensors are randomly thrown over the whole area, and case 2 represents the deployments where the sensors are randomly dumped within a very small area (in this case, the sensors are placed within a $10 \times 10m^2$ area while the intended deployment area is $120 \times 120m^2$). Additionally, case 1 represents a dense network while case 2 a sparse one. The initial topologies shown in Figure 3 have coverage degrees of 0.8709 and 0.0306 respectively.

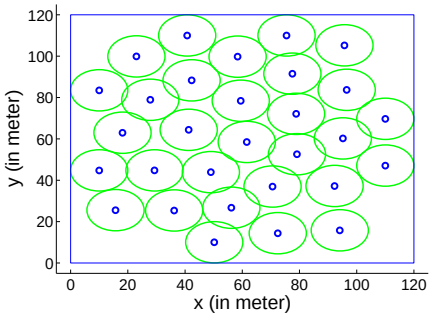


(a) Sensor trajectories in Case 1

(b) Final topology in Case 1



(c) Sensor trajectories in Case 2



(d) Final topology in Case 2

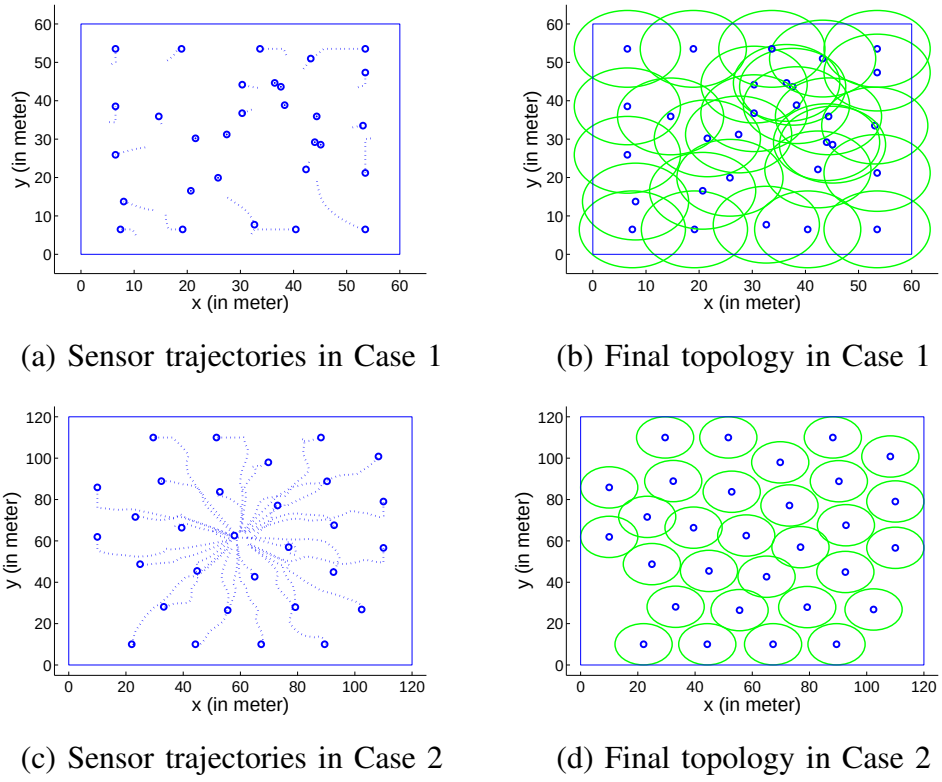
Fig. 4. Sensor trajectories and final topologies under PDND algorithm. On (a) and (c), the final position of each sensor node is shown in addition to the trajectory.

PDND has several important parameters, and their default values are summarized in Table I. Most of the parameters are self-explainable, except for the stopping criterion. The stopping criterion works in the following way. At the beginning of each time interval, every sensor calculates the position at which its force will be zero based on the local information. However, it only starts moving if the distance between the target position and its current position is larger than the stopping criterion. We note that setting a threshold on the force magnitude as discussed in Section III would be a more natural choice for PDND, but we chose to use a distance-based threshold because the algorithm using Veronoi cells which we are going to compare against uses a distance threshold. In some experiments, we also adopt different values for these parameters, and we shall explicitly specify them when presenting those results.

parameters	**default values**	**parameters**	**default values**
sensing radius r_s	$10m$	communication radius r_c	$30m$
sensor speed	$1m/s$	stopping criterion	$0.1m$
force model parameter r^*	$22m$	force model parameter r_t	$24m$
force model parameter r_f	$26m$	time interval	$1s$

TABLE I

DEFAULT VALUES FOR IMPORTANT PARAMETERS IN OUR PDND ALGORITHM.



(a) Sensor trajectories in Case 1

(b) Final topology in Case 1

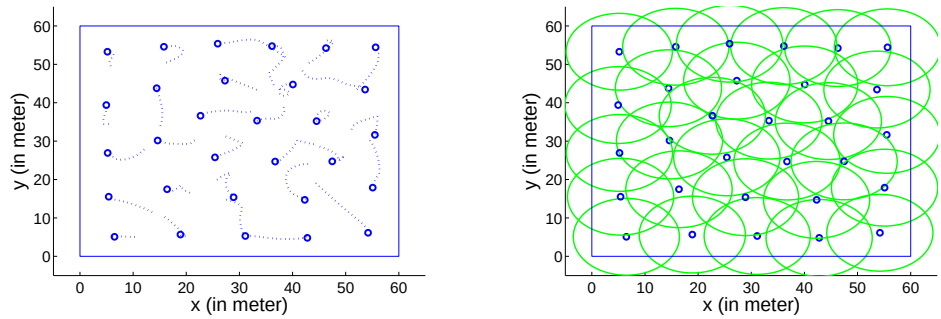
(c) Sensor trajectories in Case 2

(d) Final topology in Case 2

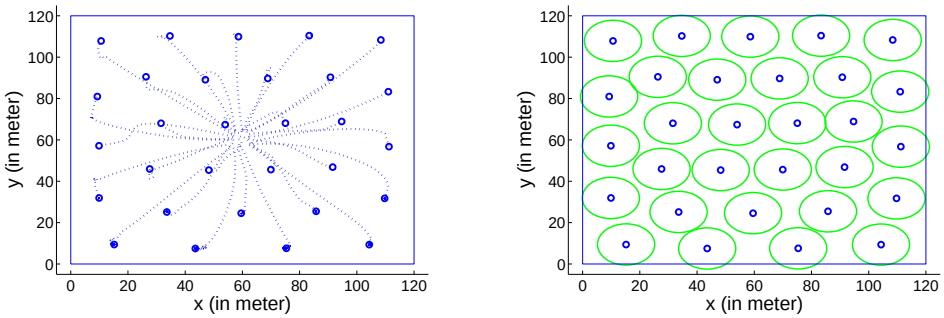
Fig. 5. Sensor trajectories and final topologies under the approximate PDND algorithm.

1) Effectiveness of PDND: After applying PDND to the two deployments, we present the resulting sensor trajectories and final topologies in Figures 4(a)-(d). For both deployments, PDND can significantly increase the coverage degree by guiding the movement of the sensors: the coverage degree in case 1 is boosted from 0.8709 to 1, and in case 2 from 0.0306 to 0.6449. We would like to point out that in case 2, the coverage degree can be at most 0.6545 because 30 nodes are not enough to fully cover the sensing field. The final topology for case 2 shown in Figure 4(d) still includes small overlap among sensing areas, and this is because the stopping criterion is not strict enough. The relatively straight trajectories in Figures 4(a) and (c) reveal that, PDND can not only achieve better coverage degree, but it can also lead to efficient sensor movements.

2) Approximate PDND: The rationale behind PDND is that sensors should keep moving until the potential energy of the network is minimized, and when the algorithm converges, the sensors are usually well separated from each other and the coverage degree is maximized. However, if the network has enough number of sensors, it is often inefficient, and unnecessary, to evenly distribute them to fully cover the sensing field. We would like to emphasize that in a dense network as in case 1, an approximately uniform distribution of the sensors can also achieve the maximum coverage degree, and more importantly, can significantly reduce the convergence time and the total travel distance. As a result, we have proposed the *approximate PDND* algorithm (sometimes referred to as *PDND2*) that employs a more relaxed convergence criterion. During a time interval, a sensor only moves if the two conditions are true: (1) its Voronoi cell



(a) Sensor trajectories in Case 1 (b) Final topology in Case 1



(c) Sensor trajectories in Case 2 (d) Final topology in Case 2

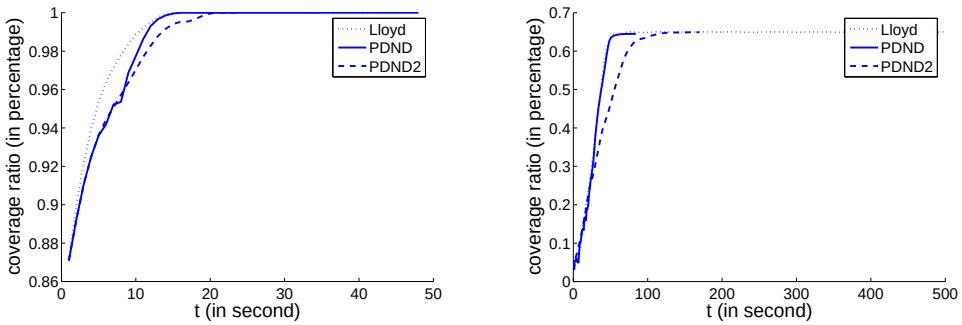
Fig. 6. Sensor trajectories and final topologies under Lloyd algorithm

is not fully covered, and (2) its intended movement distance is longer than the stopping criterion. In this way, sensor nodes will move much less compared with the original PDND algorithm.

We apply the approximate PDND algorithm to the two initial deployments shown in Figures 3(a) and (b), and present the resulting final topologies and movement trajectories in Figures 5(a)-(d). The results confirm that the approximate PDND algorithm can significantly reduce the overhead for dense deployments. For example, in case 1, the approximate PDND reduces the total travel distance by 59.65%, while still maintaining a coverage degree of 1. Furthermore, it reduces the convergence time by 50%. On the other hand, its performance is comparable to that of the original PDND algorithm in case 2 where the sensor density is low.

3) Voronoi Cell Based Mobility Management: While PDND uses the concept of potential energy and virtual forces to govern the movement of the sensors, another set of popular mobility control algorithms are built upon the concept of Voronoi tessellation, such as the technique discussed in [11], [19]. In order to study the difference between these two sets of algorithms, in this exercise, we also implement a Voronoi diagram based algorithm (referred to as Lloyd algorithm), and compare its performance with that of the PDND algorithms. Like PDND algorithms, Lloyd also partitions the time axis into discrete intervals, but during each interval, every sensor moves toward the centroid of its Voronoi cell. Details of Lloyd algorithm can be found in [19].

We apply the Lloyd algorithm to both initial deployments, and we notice that, when the network does



(a) Case 1 (b) Case 2

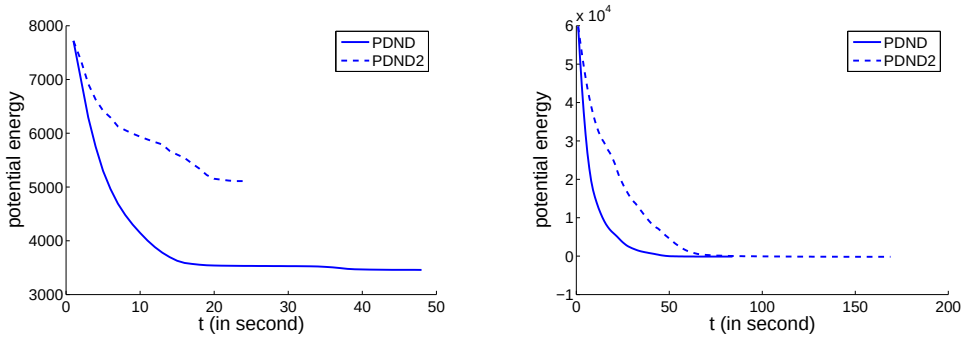
Fig. 7. Coverage degree time series

not have enough sensors as in case 2, sensors at the edge of the network may keep oscillating. In such a sparse network, a sensor at the edge may have a large Voronoi cell, and thus the centroid of the Voronoi cell is likely out of the communication range of any other sensor. This loss of connection with the rest of the network will in turn lead to errors in calculating the Voronoi cell. In order to cope with such oscillations, we manually stop Lloyd algorithm when it hits the pre-set upper bound of the convergence time. As an example, in case 2, such oscillations occur and we terminate the Lloyd algorithm after 500 seconds. We never observe this in dense networks as in case 1. Finally, the final topology and movement trajectory for both cases are shown in Figure 6 (a)-(d).

4) Performance Comparison: In order to take a closer look at the execution of these three algorithms, we present the time series for the coverage degree and potential energy of the entire network in Figure 7 and 8. After carefully examining the spatial coverage time series, we have the following observations. Firstly, for dense networks (refer to Figure 7(a)), the convergence time of Lloyd algorithm is comparable with PDND's. For sparse networks, on the other hand, Lloyd algorithm converges very slowly (or it has to be forced to stop) as discussed above. Secondly, both original PDND and Lloyd can maximize the coverage degree quickly, and then spend a long time fine-tuning the positions of each sensor. On the contrary, the approximate PDND algorithm can efficiently reduce the fine-tuning overhead without sacrificing the overall network coverage degree. Thirdly, the approximate PDND takes a longer time to reach the maximum coverage degree than the other two, as the sensors move at a slower pace due to the adopted movement criterion. Fourthly, moving toward centroids or moving along the directions of virtual forces does not necessarily lead to strict coverage degree increasing in the middle of the algorithms' running, and therefore, the time series may exhibit zigzag-like behaviors.

The system potential energy time series (refer to Figure 8) show similar trends. However, we would like to emphasize two issues. First, we observe that the potential energy of the network strictly decreases as we proved before. Second, In case 1, the approximate PDND algorithm stops when the potential energy is still high because each sensor has already fully covered its Voronoi cell.

In the next set of experiments, we study how these three algorithms perform when we vary some of the



(a) Case 1 (b) Case 2

Fig. 8. Potential energy time series

parameters, i.e. the time interval duration and the stop criterion value. The detailed results are presented in Tables II and III. In general, a smaller time interval may speed up the convergence as each node will have more up-to-date knowledge about other nodes, while a smaller stop criterion may lead to a slower convergence time and longer travel distance. In order to compensate for the randomness of the results, each entry in the table is the average over 100 different initial deployments. Further, all the 100 initial deployments that belong to case 1 are generated by uniformly randomly throwing sensors over the entire $60 \times 60m^2$ field, while all the 100 initial deployments that belong to case 2 are generated by uniformly randomly throwing sensors over the central $10 \times 10m^2$ area of the $120 \times 120m^2$ sensing field.

For network topologies that belong to case 1, with the time interval of 1 second, PDND and Lloyd deliver very similar performances regardless of the stop criteria. With the same configuration, the approximate PDND algorithm incurs much lower overheads while achieving the same coverage degree: compared to the other two algorithms, it can reduce the convergence time by 40% and the total travel distance by 60%. As the time interval becomes smaller, the performances of PDND and Lloyd start to differ: PDND results in a shorter travel distance while Lloyd can converge faster. More specifically, the performance gain of PDND in terms of total travel distance is more significant with slightly larger stop criteria, while the performance gain of Lloyd in terms of convergence time is more significant with slightly smaller stop criteria. On the other hand, under a smaller time interval, the approximate PDND can reduce the total travel distance even further, but its convergence time becomes longer relative to Lloyd.

We observe very different trends for sparse network topologies that belong to case 2. As discussed earlier, Lloyd may cause oscillations in such cases, and thus either converge very slowly or never converge. As a result, both of the PDND algorithms significantly outperform Lloyd. In the simulations, we stop Lloyd at 1000 seconds when the time interval is 1 second or 0.5 second, at 500 seconds when the time interval is 0.1 second. As a result, the true performance of the Lloyd algorithm can be much worse than what is shown in Tables III. Also, as we have discussed before, the advantage of the approximate PDND algorithm is less pronounced for sparse networks.

VI. Case Study II: Spatial Migration

In our second case study, we examine the possibility of using network dynamics to realize mobile sensor networks that can migrate themselves to track moving objects.

A. Problem Setup

Many sensor applications aim at monitoring mobile objects, such as the one presented in [20]. There are three general ways of implementing such applications. First, we can attach sensor nodes to the mobile targets, and then let the sensors ship the data back to the base station in an opportunistic manner. Second, we can place a sufficiently large number of sensors to cover all the possible (or, important) areas that the target may visit frequently. Third, we can deploy mobile sensor nodes that can autonomously follow the moving objects. There are pros and cons associated with each of these approaches, and no single approach can suit all different application scenarios. For instance, though the first approach is cost-effective, attaching sensor nodes to a mobile target might not always be possible. As another example, if the target's movement is predictable and is within a limited range, the second approach is viable. But it will be too costly an option if the target covers a large area. On the other hand, as more sensor nodes are equipped with mobility, and as the mobility-management techniques advance, the third approach will become more prevalent.

Migration Metric: In this case study, we adopt the third approach, and the most important metric is the sensor network's capability of chasing the target. If the network of sensor nodes can follow the target at its movement, the mobility management algorithms are considered successful.

Force Model and Application Model: Since the focus of this study is on governing sensor mobility to chase mobile targets, we do not intend to elaborate on how sensor nodes can detect the movement or location of the target. Instead, we assume that sensors whose sensing ranges cover the target can obtain the target's location information. Further, we assume that each sensor can operate in two modes: normal mode and chasing model. A sensor node switches to chasing mode when it detects that the target is moving. In some cases, it may be more desirable to prevent sensor nodes from chasing the target when

Time Interval (in s)		1						0.1					
Stop Criteria (in m)		0.2			0.05			0.02			0.005		
Algorithm		Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2
Final Coverage Degree	Mean	1	1	1	1	1	1	1	1	1	1	1	1
	Var (1e-9)	2.2	6.8	1097	0	0.2	822	0	42	1801	0	3.3	391
Convergence Time (in s)	Mean	33.04	37	23.42	67.57	67.11	34.18	20.52	27.02	21.05	27.12	50	30.24
	Var	50.79	83.27	51.16	365.34	402.1	255.83	14.59	41.69	47.69	43.12	207.65	310.99
Total Travel Distance (in m)	Mean	247.65	254.47	109.37	288.87	286.96	105.31	335.22	275.54	96.25	348.16	306.56	99.01
	Var	1233.8	937.7	551	1988.9	1618.4	672.81	2174.5	1404.6	548.46	2179.4	1643.6	593.41

TABLE II

Numerical results for case 1

the target moves about within the network. To address this concern, we can let boundary nodes switch to chasing mode first because they can detect whether the target is leaving the network.

Sensor nodes in the normal mode employ the same force model as discussed in the previous case study. In addition to the forces that exist in the normal mode, a sensor in the chasing mode is also influenced by an attractive force, $\mathbf{f}'$, pointing to the position of the target. In this study, we choose to set $\mathbf{f}'$ to a constant value for all sensors in the chasing mode, regardless of their distances to the target. The larger this constant value, the tighter the sensors follow the target.

Now, let us look at how the entire network migrates following the moving target. As soon as the target moves and the nodes that can detect its movement decide to chase, these sensor nodes switch to chasing mode. As for the rest of the sensor nodes, we can adopt two strategies:

- *RapidChase*. In this case, nodes in the chasing mode immediately flood the entire network with a *Switch2Chase* REQ message, and a sensor node in the normal mode reacts to the REQ message by switching to the chasing mode.
- *SlowChase*. In this case, only the nodes that can detect the movement of the target will stay in chasing mode. Other nodes, though in normal mode, will also move due to the movement of their neighbors.

As mentioned before, nodes in chasing mode are affected by an attractive force pointing to the target, whose magnitude is assigned a constant value. The calculation of the force requires each sensor node to have the knowledge of the target's location. For this purpose, we assume that the target's position can only be sensed within a certain range (referred to as *detection range*). Sensor nodes within the detection range of the target can obtain the target's position, and if under the rapid chase strategy, they will broadcast the information to the rest of the network periodically. Sensors may lose the target if all of them are outside of the detection range. This is the scenario that we try to avoid.

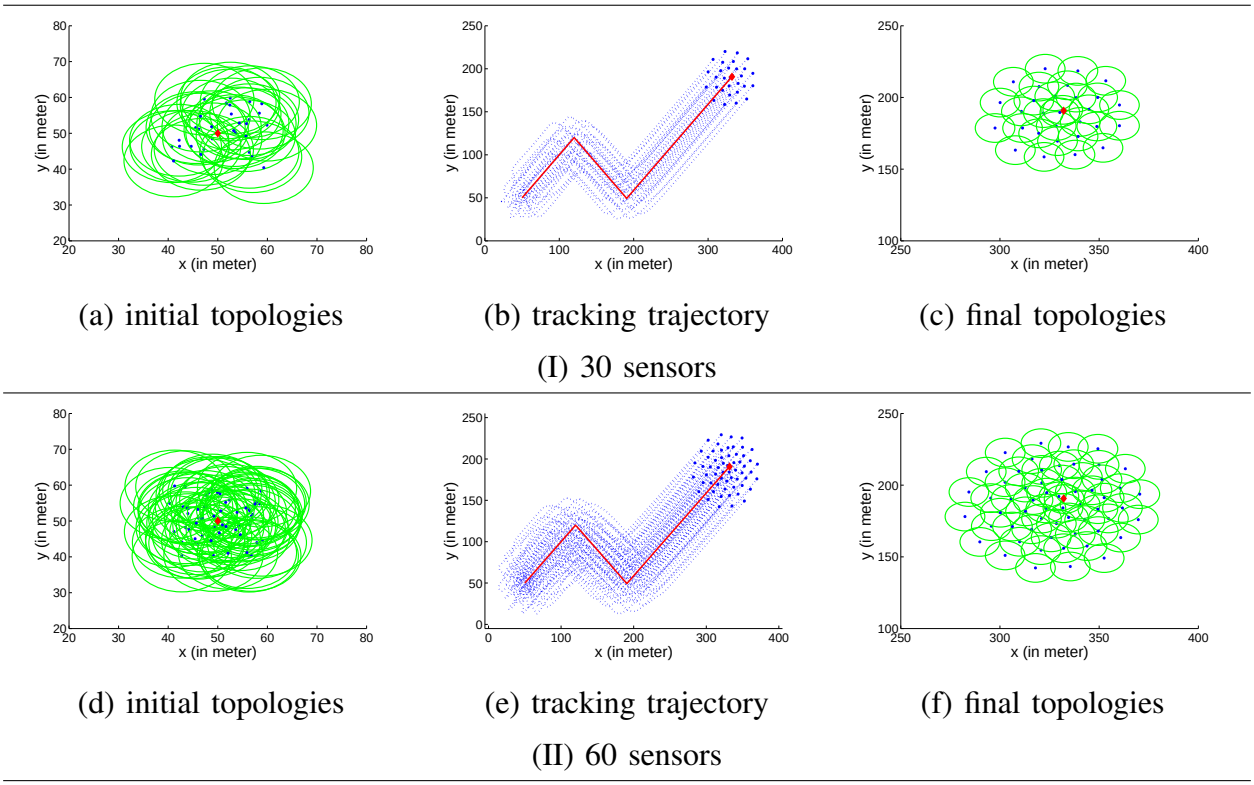
B. Simulation Results

1) Rapid Chase Strategy: We set up simulation experiments to demonstrate the effectiveness of PDND in enabling mobile sensor networks to track mobile targets. Meanwhile, to examine the scalability of the PDND algorithm, we test two networks, a smaller one with 30 sensors and a larger one with 60 sensors.

Time Interval (in s)		1						0.1					
Stop Criteria (in m)		0.2			0.05			0.02			0.005		
Algorithm		Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2
Final Coverage	Mean	0.6498	0.6433	0.6412	0.6516	0.6455	0.6452	0.652	0.6462	0.6452	0.652	0.6461	0.6451
Degree	Var (1e-6)	5.3	9.6	15	3.2	4	4.4	3.8	3.2	4.2	1.5	3.4	3.7
Convergence	Mean	475.87	82.66	119.4	519.86	141.45	186.08	367.88	73.51	97.43	431.78	159.41	156.17
Time (in s)	Var	193170	138.13	237.39	170120	951.08	1105.3	39256	102.87	276.04	24936	7734.5	5646.4
Total Travel	Mean	2805.4	1296.9	1192.6	3006.3	1397.8	1311.3	2201.8	1584.2	1418.7	2658.8	1713.6	1511.2
Distance (in m)	Var	2754000	454.39	878.32	3603800	1063.2	1529	391800	3927.5	2912.6	562770	16225	8464

TABLE III

NUMERICAL RESULTS FOR CASE 2



(a) initial topologies (b) tracking trajectory (c) final topologies

(I) 30 sensors

(d) initial topologies (e) tracking trajectory (f) final topologies

(II) 60 sensors

Fig. 9. RapidChase results.

At the beginning of the experiments, the target is located at $(50, 50)$ and sensors are randomly deployed around it, as shown in Figure 9 (a) for the 30-node network and (d) for the 60-node network.

Once the experiment starts, the target moves following the trajectory as shown in Figures 9 (b) and (e) respectively (the red lines) for 400 seconds. To make the chasing task challenging, we let the target move at a speed of $1m/s$, which is the highest speed a sensor can move at. The detection range of the target is $10m$, and the nodes in the detection range broadcast the target's location once every $1s$. The sensors have the same parameters as in the previous case study, i.e. communication radius $r_c = 30m$ and sensing radius $r_s = 10m$. The time interval is $1s$. The stop criteria is set to $0.1m$. To reduce the energy consumption while maintaining the chasing ability, a sensor stays still whenever it is in the normal mode. In addition to the target's trajectory, Figures 9(b) and (e) also present the trajectory for all the sensor nodes, which demonstrate that, though the target moves at a fast speed, sensors can closely chase the target, regardless of the network size. The final topologies after the target stops are presented in Figures 9(c) and (f). One interesting phenomenon needs to be mentioned is that in the final topology, all sensors tightly surround the target – such a topology is advantageous because it can tolerate sensor failures. This is the result of having an attractive force pointing to the moving target for each sensor.

2) Slow Chase Strategy: Unlike the rapid chase strategy in which the target applies an attractive force to every sensor node, the slow chase strategy only applies the attractive force to the sensor nodes that are within the target's detection range. Since nodes outside the target's detection range are only affected

by the forces from their neighbors, the network's capability of chasing the target is thus determined by how closely those nodes follow the target, which in turn is determined by the relationship between the speed of the sensor nodes and that of the target, and how fast sensors get information about the target's location. In order to study the impact of this relationship on the chasing performance, we have looked at three scenarios. In all the three scenarios, we have 30 sensor nodes, and the sensors can move at the speed of $1m/s$. In case 1, the time interval is $0.1s$ and the target moves at the speed of $0.5m/s$. In this case, the sensor network loses the target very soon, with the trajectory and the final topology shown in Figures 10(b) and (c). In case 2, the time interval is the same as that in case 1, but the target moves at a much slower speed: $0.2m/s$. The slower target in this case, enables the rest of the network to follow it to the final destination, as shown in Figures 10(d)-(f). In case 3, we keep the target speed the same as that in case 1, but we decrease the time interval to $0.02s$ so that the sensors can get their neighbors' location information more often, and hence, the sensor network again successfully chases the target, as shown in Figure 10.

Comparing rapid chase strategy and slow chase strategy, the former is always able to follow the target with additional broadcast overhead. In the slow chase strategy, in order to successfully chase the moving target, either the sensors should be able to move much faster than the target does, or the sensors should frequently exchange their position information.

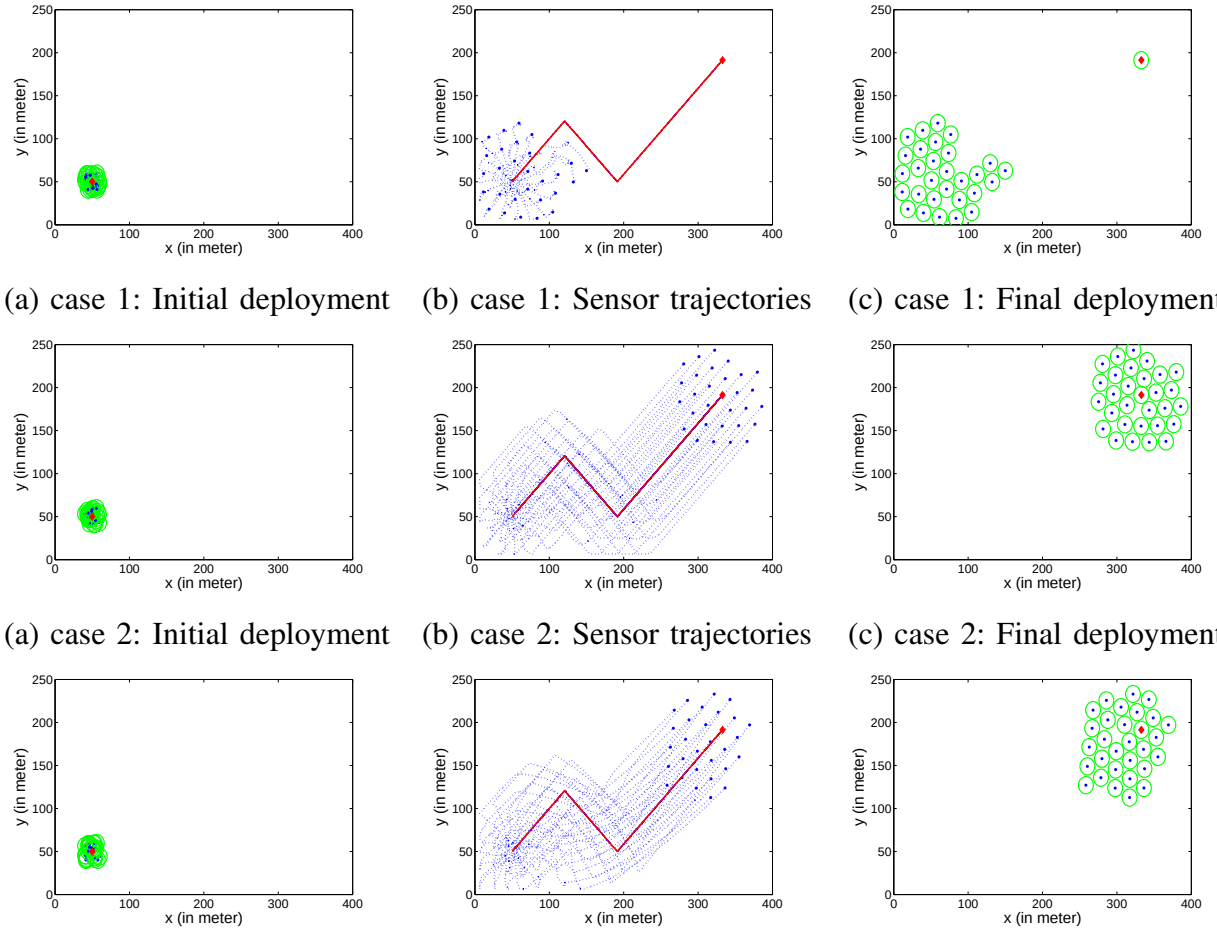
VII. Case Study III: Spatial Retreats

In this section, we discuss a second application of network dynamics that involves repairing ad hoc networks subjected to attacks of radio interference.

A. Jamming attacks and Spatial Retreats

The shared nature of the wireless medium makes wireless networks susceptible to a broad array of security threats. In particular, one class of powerful attacks that has received attention recently is jamming [21]–[25]. Adversaries may easily jam legitimate wireless communications by either continuously emitting RF signals to occupy the wireless channel, or by interrupting the transmission and reception of legitimate packets [25]. Either way, the net result is that legitimate traffic will be interfered with. Jamming attacks can have a particularly deleterious effect on MSNs as the presence of a jammer may block whole regions of the network, as depicted in Figure 11(a).

To defend against jamming attacks, two strategies were recently proposed in [24], namely channel surfing and spatial retreats. The underlying idea behind each strategy is to evade the interferer– either in the spectral sense, or in the physical sense. In this paper, we are interested in spatial retreats, in which jammed nodes try to evacuate from jammed regions. As such, spatial retreats are suitable for MSNs. In the remainder of this section, we discuss our proposed spatial retreat strategies for MSNs, and



(a) case 1: Initial deployment (b) case 1: Sensor trajectories (c) case 1: Final deployment

(a) case 2: Initial deployment (b) case 2: Sensor trajectories (c) case 2: Final deployment

(a) case 3: Initial deployment (b) case 3: Sensor trajectories (c) case 3: Final deployment

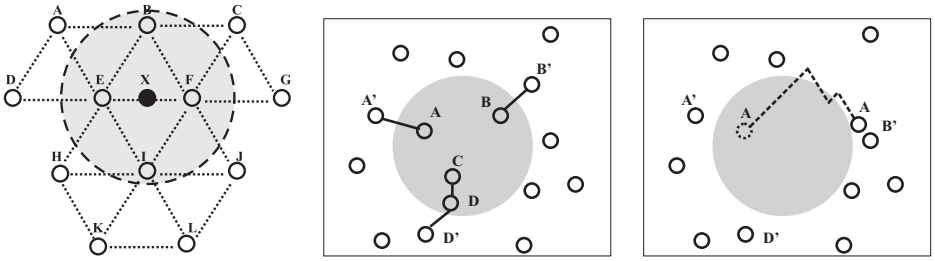
Fig. 10. Slow Chase: three cases

show how network dynamics may be incorporated into spatial retreats to achieve robustness in network communications.

The rationale behind spatial retreats is that when mobile nodes are interfered with, they should simply move to a safe location. Assuming each mobile node can detect jamming attacks in a timely fashion [25], the key to the success of this strategy is to decide where the mobile nodes should escape to and how these nodes should coordinate their escapes. Merely escaping from a jammed region is not a sufficient defense mechanism, however, as a mobile adversary can move through the coverage area and cause large swaths of the MSN to relocate. By doing so, an adversary can cause the network to become unevenly distributed, or even partitioned, thereby severing network communications.

Therefore, spatial retreat strategies should be robust to mobile jammers. In order to achieve this robustness, our spatial retreat strategy has two phases:

- *Escape phase*. In this phase, the nodes that are located within the jammed area move to "safe" regions, and stay connected with the rest of the network.
- *Reconstruction phase*. After the jammed nodes escape from the jammed area, a distributed network



(a) (b) (c)

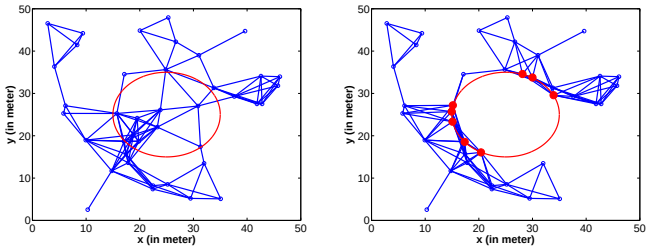
Fig. 11. (a) illustrates the jamming attacks in a network. (b) and (c) illustrate how a node escapes from a jammed area, where (b) illustrates the network topology when the jamming attack occurs with the jammed area highlighted by the gray area, and (c) uses the dashed line to mark the trace through which node A escapes from the jammed area and re-connects to the rest of the network.

dynamics algorithm is applied to re-adjust the network deployment to be more uniformly covered. In particular, after the jammer has moved on, the jammed area will leave a hole in the network coverage, and network dynamics will serve to quickly fill in the hole and restore network coverage.

We begin by discussing the escape strategy that we have developed. Suppose the network is connected before the jamming attack, i.e. every node within the jammed area is connected with nodes outside via one-hop or through multiple hops. In the example shown in Figure 11(b), before the jamming attack, node A was directly connected with A', node B was directly connected with B', node D was directly connected with D', and C was connected with D' via D. After the jamming attack is detected, the nodes within the jammed area choose a random direction to evacuate. While moving, each node continuously runs the jamming detection algorithm until it leaves the jammed area. As soon as it leaves the jammed area, it tests whether there are some nodes within its radio range. If not, it moves along the boundary of the jammed area until it re-connects to the rest of the network. In Figure 11(b), if node A moves along the boundary, it will eventually arrive at a location which is between the location of A' and the original location of A, where it can reconnect to A'.

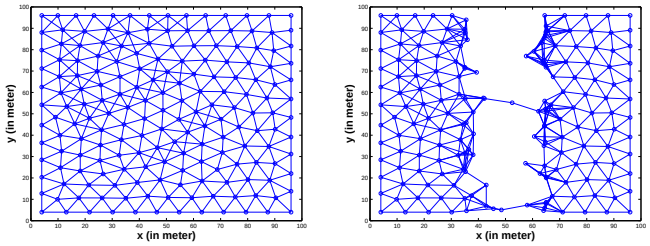
In order to make sure a node moves along the boundary of the jammed area, the node must continually run the jamming detection algorithm. Following the hull-tracing strategy in [24], it can use the simple strategy: whenever it feels the jammer's power is increasing, it makes a 90 degree left turn; whenever it feels the jammer's power is decreasing, it makes a 90 degree right turn. After it has moved a total r distance, where r is its radio range, the node will probe to see whether there is another node in its radio range (probing can also occur at a finer granularity). If not, it will continue moving along the boundary. Figure 11(c) illustrates how node A chooses a random direction to escape from the jammed area, and how it re-connects to the rest of the network using the simple policy.

Figures 12 (a) and (b) present a set of simulation results for the escape strategy. In this set of experiments, we generate a random network topology with 40 nodes deployed over a 50x50m^2 network field. Each node's radio range is 10m. The jammed area is a circle centered at location (25, 25), with a radius of 10m.



(a) The topology before escaping. (b) The topology after escaping.

Fig. 12. Simulation results illustrating the effectiveness of the escape strategy.



(a) (b)

Fig. 13. A mobile jammer may partition the network: (a) The topology before the jamming attack, and (b) The topology after the jammer moves from the bottom to the top.

The topology before the victims escape is shown in Figure 12(a). Figure 12(b) shows the topology after these nodes escape to the boundary of the jammed area and reconnect to the rest of the network. These results show that our simple escape strategy is effective at escaping the jammed area, while ensuring evacuated nodes reconnect with the network.

We next turn to the reconstruction phase. Although our simple escape strategy guarantees that every jammed node can escape from the jammed area and successfully connect to the rest of the network, a serious problem remains. As we noted earlier, if the jammer is mobile, its movement may cause the network to become severely unbalanced, or even partitioned. As an example, in Figure 13, we depict an initial network configuration (Figure 13(a)), and then introduce a jammer that moves in the y-direction through the middle of the network. The result is a network that is severely partitioned (Figure 13(b)).

In order to address this problem, we propose to apply the network dynamics to continuously repair the network topology, regardless of the jammer movement. In this scenario, there are three types of forces in the network field: the forces between the nodes, the force from the boundary of the region, and the force from the boundary of the jammed area. We used the same model for the internode forces and region boundary forces as we used in Section V. The force that we used for the boundary of the jammed area is similar to the force for the boundary of the network field. Nonetheless, we cannot pre-program the jammed area boundary as in the case of network field boundary. Instead, jammed area mapping techniques such as the one proposed in [21] should be incorporated here. In [21], upon detecting the presence of a jammer, nodes will report their situations to their neighbors, and based on these reports, nodes on the boundary of the jammed area can collectively map out the jammed area.

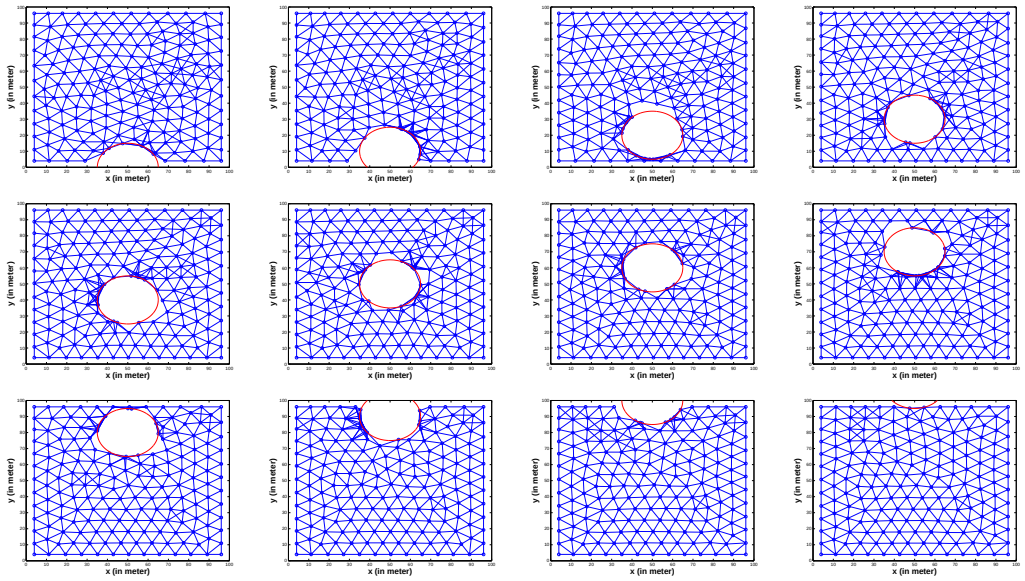


Fig. 14. The figures depict the ability of the robust spatial retreats algorithm to repair the effect of a jammer passing through the coverage region using the PDND algorithm.

We now examine the behavior of our robust spatial retreat strategy by looking at an experiment involving a mobile jammer cutting across the network coverage area. Figure 14 illustrates the evolution of the MSN's topology as the jammer moves through the network, and the robust spatial retreat algorithm not only evacuates the jammed area but also repairs regions left empty by the mobile jammer. In this experiment, we used PDND. Overall, the benefit of applying distributed network dynamics is two-fold: (1) as soon as the victim nodes escape from the jammed area, instead of gathering around the boundary, the nodes redistribute to cover the rest of the network field more evenly; and (2) as soon as the jammer leaves the current location, the resulting "hole" can be quickly repaired.

VIII. Related Literature

The mobility of the communication infrastructure is a distinguishing feature of mobile ad hoc networks. Random mobility in mobile networks has been extensively investigated, e.g. [26]–[28]. Traditionally, mobility has been a burden for wireless networks to cope with. However, it has become a recent trend in the area of mobile communications to look for scenarios where mobility can improve network performance. In [29], Grossglauser et al. show that random mobility in conjunction with multiuser diversity routing can enhance the capacity of a communication network. Goldenberg et al. [30] examined cases where mobility improves network performance. They presented one of the first algorithms for controlling node mobility to improve network performance. This distributed algorithm adjusts the locations of mobile devices according to local information to achieve global communication objectives.

It has been widely discussed in the literature that the mobility of a MSN can be used to improve the network's sensing coverage, following either a non-ideal initial deployment of sensors or the presence of sensor failures [3]–[11], [19], [31]–[33]. The majority of these papers adopt the potential field (or virtual

force) approach, which, as far as we know, was first proposed to navigate a robot in [2]. As its name suggests, the potential field approach builds a potential field based on the environment and the objective, and guides the movement accordingly. For example, in [34], the objective is to navigate a robot to a specified location through a field with obstacles. To complete the task, authors build a potential field by assuming that each obstacle applies a repulsive force to the robot, and the objective position applies an attractive force.

Regarding how to perform the deployment, aforementioned potential-field-based works can be classified into centralized ([3]) and distributed ([4]–[11]) approaches. In a centralized approach, the movement of all sensors is guided by a central entity that possesses the global information. In a distributed approach, each sensor directs its own movement based on the local information. The distributed approaches can be further divided into parallel ([4]–[7], [10], [11]) and sequential ([8], [9]) approaches. In a parallel approach, all sensors can move simultaneously; in a sequential approach, only one sensor is allowed to move at each moment.

While some distributed approaches ([3], [10], [11]) adopt heuristic methods with respect to how the deployment process stops, others ([4]–[9]) obey Newton's laws of motion. If Newton's laws of motion are adopted (implicitly or explicitly), sensors are assumed to possess both potential and kinetic energy, and a friction force (damping factor) is introduced for each moving sensor. Because the friction force drains energy out, the total energy possessed by a sensor is a Lyapunov function, which guarantees that the deployment process stops sooner or later.

As discussed in Section II, adopting Newton's laws of motion has an obvious disadvantage: a sensor oscillates before it stops. An example can be found in [34]. Before the robot in [34] settles down, it moves around the objective position for a while. Oscillations should be avoided because they waste energy and prolong the deployment process. However, if Newton's laws of motion are adopted, oscillation is inherent and hard to be eliminated.

The issue of jamming detection was briefly studied by Wood and Stankovic in [21] in the context of sensor networks. Several different models for jammers, along with multimodal detection mechanisms were presented in [25]. Countermeasures for coping with jammed regions in wireless networks has been studied in [23], [24]. In [24], two countermeasures are presented for coping with jamming. The first method, channel surfing, involves a form of on-demand link-layer frequency hopping, where valid participants change the channel they are communicating on when a denial of service attack occurs. The second method, spatial retreats, focused on the case of a stationary jammer, and did not deal with the pernicious effects of a mobile jammer.

IX. Concluding Remarks

This paper presented a framework for managing the mobility in a mobile sensor network by defining suitable potential energy functions. We proposed that the equations of motion should follow a steepest

descent formulation to minimize potential energy instead of following Newton's laws of motion adopted by other works, and devised a parallel distributed network dynamics algorithm, PDND, whereby devices make movement decisions based on local knowledge. We formally proved the convergence of PDND, and explored its effectiveness in three applications towards efficiently managing mobile sensor networks. First, we demonstrated that PDND can successfully maximize a spatial coverage of any given random deployment. In addition to the baseline algorithm, we also proposed an enhanced version of PDND which can shorten the convergence time by terminating the motions as soon as each node covers its Voronoi cell. We also showed our PDND algorithms can work in scenarios that were impossible for popular Voronoi-cell based mobility management algorithms. Second, we discussed how to apply PDND to those MSNs that track moving targets. We showed that, if each sensor node's speed is reasonable compared to the target's speed, the network can successfully chase the moving target as it moves. Finally, we examined the application of network dynamics to the problem of managing a MSN's topology in the presence of a jamming attack. In particular, simple spatial retreat strategies, whereby jammed nodes seek to escape the jammed area, can lead to fractured network topologies. By employing network dynamics in conjunction with a jamming escape algorithm, we developed a robust strategy capable of repairing network partitioning as a jamming device cuts through the MSN. By conducting these three case studies, we show that the model of network dynamics and the parallel distributed movement algorithm can efficiently utilize node mobility towards more robust and efficient sensor network operations.

REFERENCES

[1] "Adaptive sampling and prediction," http://engineering.princeton.edu/gallery/asap/index.htm.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, 1985, pp. 500–505.

[3] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *Proceedings of IEEE INFOCOM'03*, Mar. 2003, pp. 1293–1303.

[4] A. Howard, M. Mataric, and G. Sukhatme, "Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem," in *Proceedings of 6th International Symposium on Distributed Autonomous robotics Systems (DARS'02)*, June 2002.

[5] Sameera Poduri and Gaurav S. Sukhatme, "Constrained coverage for mobile sensor networks," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2004, pp. 165–171.

[6] Petter Ögren, Edward Fiorelli, and Naomi Ehrich Leonard, "Cooperative control of mobile sensor networks: adaptive gradient climbing in a distributed environment," *IEEE Transactions on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, 2004.

[7] Naomi Ehrich Leonard and Edward Fiorelli, "Virtual leaders, artificial potentials and coordinated control of groups," in *Proceedings of 40th IEEE Conference on Decision and Control*, 2001, pp. 2968–2973.

[8] Jindong Tan and Oscar Mateo Lozano, "A distributed graph model for the coordination and formation of multi-robot systems," International Journal of Robotics Research, submitted.

[9] Jindong Tan and Ning Xi, "Peer-to-peer model for the area coverage and cooperative control of mobile sensor networks," in *Proceedings of SPIE symposium on Defense and Security*, 2004, pp. 439–450.

[10] Nojeong Heo and Pramod K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 35, no. 1, pp. 78–92, 2005.

[11] G. Wang, G. Cao, and T. Porta, "Movement-assisted sensor deployment," in *Proceedings of IEEE INFOCOM'04*, Mar. 2004, pp. 2469–2479.

[12] P. G. Fernandes, P. Stevenson, A. S. Brierley, F. Armstrong, and E. J. Simmonds, "Autonomous underwater vehicles: future platforms for fisheries acoustics," *ICES Journal of Marine Science*, vol. 60, no. 3, pp. 684–691, 2003.

[13] R. P. Feynman, *Feynman Lectures On Physics*, Addison Wesley, 1970.

[14] C.G. Gray and K. E. Gubbins, *The Theory of Molecular Fluids 1: Fundamentals*, Clarence Press, Oxford, 1984.

[15] P. Enge and P. Misra, *Global Positioning System: Signals, Measurements and Performance*, Ganga-Jamuna Pr, 2001.

[16] K. Langendoen and N. Reijers, "Distributed localization in wireless sensor networks: a quantitative comparison," *Computer Networks*, vol. 43, no. 4, pp. 499–518, 2003.

[17] Dimitri P. Bertsekas and John N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific, 1997.

[18] A. Ghosh, "Estimating coverage holes and enhancing coverage in mixed sensor networks," in *Proceedings of 29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 68–76.

[19] J. Cortés, S. Martłnez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[20] Ting Liu, Christopher M. Sadler, Pei Zhang, and Margeret Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, June 2004, pp. 256–269.

[21] A. Wood, J. Stankovic, and S. Son, "JAM: A jammed-area mapping service for sensor networks," in *Proceedings of 24th IEEE Real-Time Systems Symposium*, 2003, pp. 286–297.

[22] A. Wood and J. Stankovic, "Denial of service in sensor networks," *IEEE Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.

[23] G. Noubir and G. Lin, "Low-power DoS attacks in data wireless lans and countermeasures," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 29–30, 2003.

[24] W. Xu, T. Wood, W. Trappe, and Y. Zhang, "Channel surfing and spatial retreats: defenses against wireless denial of service," in *Proceedings of the 2004 ACM workshop on Wireless security*, 2004, pp. 80–89.

[25] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proceedings of the 2005 ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2005, pp. 46–57.

[26] Q. Li and D. Rus, "Sending messages to mobile users in disconnected ad-hoc wireless networks," in *Proceedings of ACM MobiCom'00*, Aug. 2000, pp. 44–55.

[27] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri, "Towards realistic mobility models for mobile ad hoc networks," in *Proceedings of ACM MobiCom'03*, Sept. 2003, pp. 217–229.

[28] J. Yoon, M. Liu, and B. Noble, "Sound mobility models," in *Proceedings of ACM MobiCom'03*, Sept. 2003, pp. 205–216.

[29] M. Grossglauser and D. Tse, "Mobility increases the capacity of ad-hoc wireless networks," in *Proceedings of IEEE INFOCOM'01*, Mar. 2001, pp. 1360–1369.

[30] D. Goldenberg, J. Lin, and A. Morse, "Towards mobility as a network control primitive," in *Proceedings of ACM MobiHoc'04*, May 2004, pp. 163–174.

[31] G. Wang, G. Cao, T. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *Proceedings of IEEE INFOCOM'05*, Mar. 2005, pp. 2302–2312.

[32] S. Ganeriwal, A. Kansal, and M. B. Srivastava, "Self aware actuation for fault repair in sensor networks," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'04)*, Apr. 2004, pp. 5244–5249.

[33] A. Howard, M. Mataric, and G. Sukhatme, "An incremental deployment algorithm for mobile robot teams," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'02)*, Sept. 2002, pp. 2849–2854.

[34] Bao Q. Nguyen, Yao-Li Chuang, David Tung, Chung Hsieh, Zhipu Jin, Ling Shi, Daniel Marthaler, Andrea Bertozzi, and Richard M. Murray, "Virtual attractive-repulsive potentials for cooperative control of second order dynamic vehicles on the caltech mvwt," in *Proceedings of the 2005 American Control Conference*, 2005, pp. 1084–1089.