Swarm Robot Foraging with Wireless Sensor Motes

Katherine Russell
Dept. Computer Science
George Mason University
Fairfax, Virginia USA
krussellc@gmu.edu

Michael Schader
Dept. Computer Science
George Mason University
Fairfax, Virginia USA
mschader@gmu.edu

Kevin Andrea
Dept. Computer Science
George Mason University
Fairfax, Virginia USA
kandrea@gmu.edu

Sean Luke
Dept. Computer Science
George Mason University
Fairfax, Virginia, USA
sean@cs.gmu.edu

ABSTRACT

We investigate the use of wireless sensor motes as mobile deployable waypoints for swarm robot navigation in a foraging scenario. Each robot can deploy, retrieve, and optimize the location of the sensor motes. After deployment, the robots treat the sensor motes as nodes in a sparse graph, and store and retrieve multiple pheromones and flags locally in each of them. Pheromone information stored in the sensor motes allows the robots to build up gradients to different targets of interest, and to determine which sensor motes are good candidates to optimize location, or to harvest for reuse elsewhere. Unlike many earlier pheromone-based foraging techniques, our method must deal with the physical reality of deploying and manipulating sensor motes, including a limited mote supply both onboard and in total, robustly dealing with occlusion and interference from other robots, and handling noise and robot or mote failure. We demonstrate the effectiveness of the technique both on differential drive robots of our own design, and in simulation, to examine its ability to robustly deal with various failure modes and changes in environment.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Experimentation, Performance

Keywords

Beacons; Pheromone Robotics; Sensor Motes

1. INTRODUCTION

A major hurdle to realistic swarm robotics is performing scalable and reliable communication in difficult environments. A common solution has been to apply *indirect communication*, where the local environment itself is modified by the robots so as to leave information for future robots visiting the area. Examples include leaving trails of breadcrumbs, depositing pheromones, and erecting signs or directional markers. Indirect communication is attractive for several reasons. First, it scales to any arbitrary size area and number of agents. Second, it is potentially long-lasting. Third, information received through indirect communication is often highly relevant to a robot's current situation, since it was embedded locally in that environment. Fourth, it is robust in scenarios where global communication modes (such as a global wireless network) are infeasible.

Inspired by ants and termites, and following the lead of the artificial life literature, some swarm robotics research has examined simulating the embedding of *pheromones* directly in the environment, via drawing on surfaces with inks, scents, and phosphorescent dyes. We think these approaches have limited practical utility, partly because pheromones cannot store much information, and partly because they may mar the environments in which the robots operate.

A common alternative is to scatter some form of *breadcrumbs* or deployable *waypoints* in the environment. Robotic breadcrumbs may be small barcodes, QR codes, or RFID tags, for example. Because such objects often are read-only, much of the existing literature has augmented this kind of indirect communication with some other line-of-sight or global communication mode, or has treated such breadcrumbs as trivial environmental markers.

In this paper we study the use of wireless *sensor motes* as sophisticated read/write breadcrumbs in a food foraging scenario for a swarm of physical robots. A sensor mote is a tiny, low powered, wireless computer outfitted with an array of sensors and designed to run for months or years on batteries. Sensor motes are useful as breadcrumbs particularly because they are small, long-lived, easy to carry and deploy, and can store and retransmit significant amounts of data. Though many sensor motes can form ad-hoc networks among themselves, in this study they will only communicate with nearby robots as they pass by. Robots will not communicate with each other except indirectly through information stored in the motes.

Our experiments use small differential drive robots each outfitted with a sensor mote (for communications), and capable of picking up, moving, and releasing a single sensor mote in the environment. Robots may acquire sensor motes at a *nest* (a home base) or serendipitously in the environment, then deploy these motes as bread-crumbs. They can later optimize the location of these motes by physically adjusting their location, and can reacquire them if they are no longer needed. Robots may store and read various information in the sensor motes, including the values of different kinds of pheromones, flags, and atomic locks. The objective is for the robots to use the breadcrumbs to discover a *food source*, then optimize the breadcrumbs to shorten the trail between the nest and the food so as to maximize the amount of food being brought back to the nest.

The algorithm we will develop here is inspired by the approach we took in [9] and, as in that paper, we will likewise refer to our sensor motes as *beacons*. However the algorithm described in [9], while intended as a step in the direction of physical robots, cannot actually be deployed to real robots for a large number of reasons which we discuss later. Thus we have devised a major revision which is capable of deployment to real robots, and which deals

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

robustly with sensor failure, occlusion, race conditions, noise, and other issues encountered in dealing with physical swarm robots in imprecise environments.

We will describe the algorithm in replicable detail, then discuss our robot platform. After this we will show an experiment with the physical robots, and validate a robot simulation with an identical experiment. The simulator affords us both faster runtime and environments with many more (virtual) robots. We will then perform experiments in simulation involving varying levels of occlusion, beacon reliability, and catastrophic beacon failure.

2. PREVIOUS WORK

Both pheromones and breadcrumbs are examples of *local indirect communication* among swarm agents, where such agents leave information in the environment for later agents to exploit. Early work in this area largely came from the artificial life community [2, 4, 12, 15, 19]. In this literature, simulated agents performed foraging on a virtual grid, using a single pheromone, and agents typically deposited and read pheromones on grid squares. Some work has extended this to multiple pheromone models [3, 25], and pheromones with rich semantics (such as different evaporation rates [16]).

Later work has come from the swarm robotics community. Some of this work directly draws pheromones on the ground using odiferous pens [20], ink [21], or phosphorescent paint [11]; or generally emits odors [10] or other detectable chemicals [18]. Other work has approximated indirect communication using a breadcrumbs or beacons of some sort, but usually this is augmented with some other communication mode. For example, some work has used a global wireless network to simulate local indirect communication [22, 23, 24, 26]. Other work has used line-of-sight or short-range communication [13, 14, 17], or presumed fixed global communications devices embedded in the environment [1]. Robots have also communicated "pheromone"-like data by forming chains among themselves rather than fixing information in the environment directly (such as in [7, 8]). Finally, robots have themselves acted as mobile beacons for other classes of robots [5].

We are interested in a system which performs indirect communication by deploying and otherwise manipulating beacons which are capable of storing and transmitting rich information to local robots. We first proposed this approach in [9], where simulated swarm agents deployed, removed, and updated mobile beacons in the environment. These beacons would store local pheromone information deposited in them by the agents, and which acted as nodes in a sparse graph embedded in the environment. The agents used these beacons, and their pheromone information, to build up different gradients for other agents to follow. This model was meant as a first-step towards deployment to real physical robots, but the algorithm made a large number of simplifying assumptions which would prevent this, including:

- The agents shared a global, immediately available pool of beacons. An agent could immediately teleport a beacon from this pool to itself, or transport a beacon from the environment to the pool.
- The agents were point objects and could not collide with nor obstruct one another. Any number of agents could arrive at a given beacon at the same time.
- An agent could always see all beacons in its local environment: beacons were never occluded, nor difficult to identify.
- The beacons were reliably read from, written to, deployed, retrieved, and moved.
- The agents were not in parallel, and so could not encounter race conditions when working with the same beacon.
- The algorithm was generally not robust to noise.
- The algorithm was not sufficiently conservative when moving or retrieving beacons, and would often strand agents alone or on "islands" of beacons, with no way for them to find their way back home.

3. ALGORITHM DESCRIPTION

In this paper we describe a major revision of the technique described in [9] and demonstrate its deployment for the first time to actual physical robots. The "beacons" in our revised algorithm are mobile wireless sensor motes which communicate with nearby robots. One of the challenges in implementing this algorithm and deploying it to real physical robots is that there are a great number of corner cases and other critical implementation details necessary to describe for purposes of replicability. We beg the reader's indulgence as we discuss the algorithm in some detail here.

In the foraging scenario, robots emerge from a *nest* and search for a *food* source, then iteratively bring pellets of food back from the source to the nest. The sensor mote beacons deployed by the robots form a sparse embedded graph in the environment. For simplicity's sake, we also associate the nest and the food with unique, immobile beacons, though this is not a required feature of the environment. Robots have the ability to both add and remove beacons from the environment, as well as optimize beacon positions.

Each beacon stores several pieces of information. First, a beacon stores three real-valued *pheromones* which establish gradients in the environment to be followed by the robot: a pheromone associated with a gradient to the *nest*, one to a *food* source, and an additional *wandering* pheromone value meant to encourage exploration. As robots visit beacons, a negative wander pheromone builds up and encourages them to look elsewhere. A robot's behaviors are essentially finite-state automata, where depending on the mode of the robot, it will follow a certain pheromone gradient. Specifically, a robot may be either FORAGING for food (following the *food* gradient) or FERRYING food back to the nest (following the *nest* gradient). At all times, the robot updates the local gradients of all three pheromones based on new information.

Second, a beacon stores certain integer *indicators*: the first is the *id* of the robot, if any, that is presently "in control of" the beacon (this functions as an atomic lock when a robot wishes to move or control the beacon). The second is a *status* which can be set to one of two values: *deployed*, indicating that the beacon is in use in the environment, and *inactive*, meaning that it is not in use. New beacons, offered to robots at the nest or perhaps placed in the environment by the experimenter, are initially set to *inactive*. The status of a beacon may also be read as *faulty* if it is not operating properly. The third is a *last used* timestamp which allows the beacon to return a status of *disused* when it hasn't been updated in some period of time. This period is configurable: we set it to $3\times$ the rough time it takes for a robot to travel from one beacon to another.

We note that that neither unique robot ids nor a global clock is required for the method proposed. Robot ids are merely a simple method of creating unique locks for small swarms, but other low conflict lock ids could be generated for larger swarms. Additionally, the *last used* timestamp mentioned above is based on the beacon's internal clock and could be (and likely often is) completely out of sync with the clocks of the robots and other beacons.

Beacon pheromones gradually decay by being iteratively multiplied by some value γ. The exact value is not important, but in our simulation experiments, we cut down by $\gamma = 0.99$ every timestep.

Robots iterate the following loop. First, they identify the beacons they can find in their immediate neighborhood (by looking about). If they are not presently associated with a beacon, they attempt to acquire one if it is available and appropriate. Then, if they are associated with a current beacon c they will read and write to it, and possibly move it. They also keep track of p, the beacon they were previously associated with, and with the next beacon n which with they intend to associate.

Below is that top-level loop. Note that initially $R_{ferrying}$ is set to some positive constant REWARD: a robot starts at the nest, believing it has just successfully ferried food back to the nest, has received a reward accordingly, and is now FORAGING. Probabilities, such as $P_{Acquire}$, will be discussed in more detail later.

TopLevel( )

```
 1: global variables
 2:     $mode \leftarrow$ FORAGING                ▷ The robot's current state
 3:     $R_{ferrying} \leftarrow$ REWARD          ▷ Reward just received while ferrying
 4:     $R_{foraging} \leftarrow 0$               ▷ Reward just received while foraging
 5:     $c \leftarrow \square$                    ▷ Current beacon, where □ means "null"
 6:     $p \leftarrow \square$                    ▷ Previous beacon
 7:     $n \leftarrow \square$                    ▷ Next beacon
 8:     $countdown \leftarrow 0$                  ▷ Timer used to explore for a while
 9:     $deployFailed \leftarrow$ false           ▷ Did beacon deployment fail last time?

10: loop
11:     $\{neighbors, inactiveBeacons\} \leftarrow$ SurveySurroundings( )
                                                  ▷ Try to acquire a beacon if we don't have one
12:     if CanAcquireBeacon(inactiveBeacons) and Rand($P_{Acquire}$)
13:         TryToAcquireBeacon(inactiveBeacons)
14:         $\{neighbors, inactiveBeacons\} \leftarrow$SurveySurroundings( )
                                                  ▷ Update the current beacon as needed
15:     if $c \neq \square$
16:         $deployFailed \leftarrow$ false       ▷ Allow us to start deploying again
17:         UpdateCurrentBeacon(neighbors)        ▷ See Section 3.1
18:     DecideMode( )
19:     DecideAndGo(neighbors)                    ▷ See Section 3.2
```

The UpdateCurrentBeacon(...) and DecideAndGo(...) sub-algorithms will be described in later sections. For now, we describe the other functions here.

Rand(prob) Return true with probability *prob*, else false.

SurveySurroundings( ) Returns two (possibly empty) sets: one of nearby *deployed* beacons which are part of the network, and one of nearby *inactive* beacons. The robot's current beacon c is never included in either set. In our physical robots, these sets are built by rotating to visually observe nearby beacons, then querying the observed beacons for their status. Other methods of observing the local environment could easily replace this for robots with different capabilities.

CanAcquireBeacon(inactiveBeacons) Returns true if the robot could hold an additional beacon (in our scenario, this means that the robot is holding no beacons at present, as our robots can only carry one beacon at a time) and either *inactiveBeacons* is nonempty, or the current beacon c is the nest.

TryToAcquireBeacon(inactiveBeacons) If *inactiveBeacons* is nonempty, the robot chooses one at random and attempts to get a lock on it. If this is successful, it goes to the beacon, acquires it, and sets $c \leftarrow \square$, as the robot could now be at an unknown location. If *inactiveBeacons* is empty, the robot assumes it is at the nest, and so attempts to get a beacon from the nest's dispenser, acquire a lock on it, and set $c \leftarrow nest$.

DecideMode( ) Sets the current mode to FERRYING if the following scenario is true: (1) the current mode is FORAGING, (2) the robot is not carrying any beacons, and (3) there is no trail to the food from the current beacon (the FORAGING pheromone at c is 0). This mode change is done when the robot has reached an edge in the beacon graph and decides it must expand the graph, but has no beacon to contribute to this goal. Since beacons are generally available at the nest, the robot changes its mode to head to the nest.

3.1 Updating the Current Beacon

The *UpdateCurrentBeacon(...)* sub-algorithm takes information from the survey step and uses it to decide on what to change about the current beacon c. The algorithm is as follows:

UpdateCurrentBeacon(*neighbors*)

```
1: UpdatePheromones(neighbors)
2: $n \leftarrow$ NextBeacon(neighbors, mode)
3: if CanMoveBeacon(neighbors) and Rand($P_{Optimize}$)
4:     MoveBeacon( )
5: if CanRemoveBeacon(neighbors) and Rand($P_{remove}$)
6:     RemoveBeacon( )
7: UpdateLastUsed( )
```

The process of updating the current beacon c involves updating its pheromone values, then optionally optimizing or deleting c. Updating the pheromone values follows essentially the same procedure as in [9]. We then determine $n \in neighbors$, the next beacon to follow for the current *mode*. n is also used later in the calculations to determine whether c can be optimized or deleted.

UpdatePheromones(neighbors) As in [9], each pheromone value $U_p(c)(\forall p)$ at the current beacon c is updated using Equation 1, where c is the current beacon, $R_p(\forall p)$ is the reward for the given pheromone, and γ is a value between 0 and 1.

$$U_{foraging}(c) \leftarrow \max(U_{foraging}(c),\ R_{foraging} + \gamma \max_{i \in neighbors} U_{foraging}(i))$$
$$U_{ferrying}(c) \leftarrow \max(U_{ferrying}(c),\ R_{ferrying} + \gamma \max_{i \in neighbors} U_{ferrying}(i))$$
$$U_{wandering}(c) \leftarrow U_{wandering}(c) - 1 \quad (1)$$

This is a form of value iteration which builds up the foraging and ferrying pheromone gradients in the environment. We want beacons with very small pheromone values to be considered disused. Thus if either $U_{foraging}(c)$ or $U_{ferrying}(c) < \varepsilon$, then it is simply set to 0 to indicate that the beacon is disused. In our experiments, we set $\varepsilon = 0.000000001$.

NextBeacon(neighbors, mode) Finds the beacon $n \in neighbors$, if any, with the highest pheromone value for the given *mode*. If there is no such beacon, or if the associated pheromone for n is less than or equal to the same pheromone in c, returns $\square$, else returns n.

CanMoveBeacon(neighbors) Returns true if (1) the robot can carry more beacons, (2) there exists a path to optimize, that is: $c, p, n \neq \square$, the pheromones for the current mode in c, p, and n are in strictly increasing or decreasing order, and $n, p \in neighbors$, (3) the robot is allowed to move the beacon (no other robot has locked *c, p,* or *n*) (4) the swarm is not presently expanding the network (*c* has both zero or both non-zero pheromone values for FORAGING and FERRYING) (5) the robot would not be breaking any other path between the *neighbors*. This last step can be defined as follows: there does not exist a strictly increasing or decreasing ordering $y \rightarrow c \rightarrow z$, where $y, z \in$ *neighbors*, for the FERRYING or FORAGING pheromones, other than $p \rightarrow c \rightarrow n$, which would be disrupted by the movement of *c* (so either there is no path $y \rightarrow c \rightarrow z$, or robots can travel directly between beacons *y* and *z*).

MoveBeacon( ) The robot attempts to acquire a lock on *c*, *p*, and *n*. If it cannot get a lock it releases all locks it has acquired and gives up moving the beacon. Otherwise it picks up *c* and attempts to move it directly between *p* and *n*. This fails if an obstacle is encountered in placing the beacon. There are two options for failure recovery: (1) attempt to put the beacon back, or (2) deposit the beacon at the robot's current location. We used option 2.

CanRemoveBeacon(neighbors) Performs a series of checks to determine if the current beacon *c* can be safely removed from the network. These are heuristics to determine if the beacon is either at an edge of the graph which not presently being expanded by the swarm, is redundant, or is on an island formed by erroneous past removals or optimizations. An *island* is a disconnected subgraph of beacons for which there is no path to the nest.

This function returns true if all of the following are true: (1) No one is using another beacon on this path (there is no lock on *c*, and *p* and *n* have no locks or are □) (2) Neither the FERRYING nor FORAGING pheromone values at *c* are zero, or they both are (3) A heavily used but occluded path is not likely to be broken (any of the following are true: c is *disused*, there is more than one beacon in *neighbors*, or both pheromones FERRYING and FORAGING are zero) (4) *p* and *n* are within range of each other, if they exist: that is, removing *c* would not break the path from *p* to *n* (5) Any of the following is true: (a) Both FERRYING and FORAGING pheromones are zero. In other words, the robot believes it is on an island with no gradient to either food or the nest. Or: (b) *neighbors* is empty and the status of *c* is *disused*. This indicates that the beacon is likely an unimportant part of the network. Or: (c) there is a redundant beacon which can take the place of *c*. That is, there exists a beacon $b \in$ *neighbors* with higher pheromone values than *c* in both FERRYING and FORAGING, which is in range of all other beacons in *neighbors*.

RemoveBeacon( ) The robot sets the *deployed* status of *c* to false, resets all pheromones on *c*, and attempts to acquire *c* if the robot can carry more beacons. If it cannot acquire *c*, it will leave it on the field to be (potentially) acquired by robots at a later time.

UpdateLastUsed( ) Asks *c* to update its *last used* timestamp (so it will not report that it is *disused* for a while).

3.2 Deciding What To Do Next

Finally, the DecideAndGo(*neighbors*) sub-algorithm decides where to go next, attempts to travel there, and potentially deploys a beacon if appropriate. To begin, the robot tries to go to the food or nest if it exists, otherwise it considers *exploration*. Exploration is designed to add some amount of randomness to the algorithm, by making the robot do a random walk through the graph for some EXPLORECOUNTDOWN steps. This is done with probability $P_{Explore}$. Next, the robot will try *following* the current pheromone trail — the primary robot behavior — with probability P_{Follow}. If the robot can not follow the trail, it will try to *deploy* a beacon, if it can, with P_{Deploy} probability (or with 100% probability if it has no neighbors). If deployment is not an option, it may *wander* about, following the WANDERING pheromone, which is higher for beacons not frequently visited. As a last resort, the robot will *panic*.

DecideAndGo(*neighbors*)
```
 1: if mode=FORAGING and food is within range
 2:     GoTo(food)                        ▷ This will also update $R_{ferrying}$
 3: else if mode=FERRYING and nest is within range
 4:     GoTo(nest)                        ▷ This will also update $R_{foraging}$
 5: else
 6:     if countdown $= 0$ and Rand($p_{Explore}$)
 7:         countdown $\leftarrow$ EXPLORECOUNTDOWN
 8:     if countdown $> 0$ and neighbors is nonempty
 9:         GoTo(random neighbor $\in$ neighbors)
10:         countdown $\leftarrow$ countdown$-1$
11:     else if $n \neq \square$ and Rand($p_{Follow}$)
12:         GoTo($n$)
13:     else if CanDeploy(neighbors) and either Rand($p_{Deploy}$)
                                                or neighbors is empty
14:         Deploy(neighbors)
15:     else if neighbors is nonempty
16:         $n \leftarrow$ NextWanderBeacon(neighbors)
17:         GoTo($n$)
18:     else
19:         Panic( )
```

GoTo(beacon) Tries to move to the given *beacon*. If successful, *c* is set to *beacon* and *n* is set to □. If *beacon* is the food source, the robot picks up a food pellet, sets $R_{foraging}$ to REWARD, and sets *mode* to FERRYING. If the beacon is the nest, the robot deposits any food it is carrying, sets $R_{ferrying}$ to REWARD, sets *mode* to FORAGING, and, if the robot arrived with food and with a beacon, returns the beacon to the nest if Rand(P_{return}) is true. Beacons are returned so that there are occasionally robots traversing the field who can pick up *inactive* beacons abandoned by other robots.

GoTo(*beacon*) can fail in some cases. It is possible to temporarily lose sight of the target beacon as the robot approaches it. To correct for this, if a robot loses sight of a beacon without sensing an obstacle (such as a robot or another beacon), it will wait for a few seconds to see if the beacon becomes visible again before declaring failure. However, if an obstacle is detected when sight is lost, the robot will attempt to go around the object and continue towards the target.

If GoTo(*beacon*) fails, *c* and *n* are both set to □.

CanDeploy(neighbors) Returns true if all of the following are true: (1) *deployFailed* is false or the list of *neighbors* is empty, (2) the robot is carrying a beacon, (3) there is a reasonable place to deploy. On our robots, a reasonable place to deploy was determined by the fact that either the list of *neighbors* contains fewer than four neighbors or a pair of consecutive *neighbors* can be found in a rotational sweep of whose angle is greater than 90 degrees. Other methods would work as well and might be preferable with more capable robots or when a different density of beacons was desirable.

Deploy(neighbors) Attempts to place a beacon in the optimal direction (our robots chose the largest angle among those found as described in CanDeploy( )) at a distance slightly smaller than the range at which a robot can see a beacon from another beacon.

Deployment can fail if there is an obstacle (such as a robot) in the way of deploying at a given location. Deployment will also fail if doing so would deploy the beacon very close to previously-undetected beacons (because they were out of range at the time that

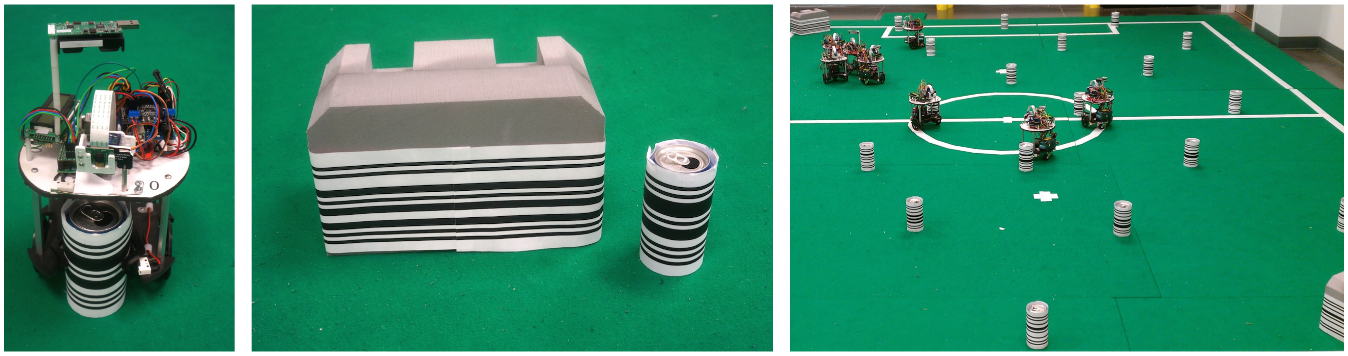

Figure 1: Physical Robot Setup. (Left) Differential-drive robot with sensor mote (at top), holding a beacon (a soda can wrapped in a unique bar code). The soda can is associated with a unique sensor mote. (Center) Larger food/nest beacon (to allow more robots to congregate) at left, also associated with a unique sensor mote, with regular beacon shown at right. (Right) Robots at work ferrying food to the nest in an environment of deployed beacons.

CanDeploy( ) was called). To detect this second situation, our robots walked some distance in the deploy direction and verified that there are no additional beacons near that location now visible which had not been previously.

If deployment fails, c is set to □ as it is now at an unknown location, and *deployFailed* is set to true. *deployFailed* prevents a second deployment without first returning to a beacon in the network. Without this, it is possible that a robot could make repeated attempts to deploy in bad locations (such as along a wall), and eventually form a single-beacon island. *deployFailed* will be reset next time that c is set in the main loop.

If deployment is successful, c is set to the deployed beacon, the lock is released for that beacon, and the *inactive* status on the beacon is set to false.

NextWanderBeacon(neighbors) Finds the beacon $n \in$ *neighbors*, if any, with the highest pheromone value WANDERING. If there is no such beacon, returns □.

Panic( ) p is set to □ and a spiral (or some other) search pattern is initiated. While the search pattern is running, the robot constantly searches for deployed beacons which are not c. The search pattern stops when such a beacon is found. c is then set to □.

4. EXPERIMENTS

Physical Robot Platform. Our test platform for the algorithm is shown in Figure 1, and consisted of multiple small robots, mobile beacons, and larger food and nest beacons.

The robots were small differential drive robots of our own design, capable of grasping a single beacon (in the form of a soda can) and moving it from place to place. These robots had an Arduino Uno microcontroller coupled with a Raspberry Pi Linux computer, and were outfitted with a camera, 802.11 wireless communication (not needed here), USB interfaces, five Sharp IR infrared distance sensors, two simple bump sensors, two encoded wheels, an embedded gripper capable of collecting small cans, a flat push surface, and an I2C-driven display. The robots acted entirely autonomously.

The robots were also outfitted with a Tmote Sky wireless sensor mote attached to the Raspberry Pi. The sensor motes used a version of Contiki OS, an operating system designed for wireless low-power and lossy networks (LLNs) [6]. The sensor motes communicated over 802.15.4, channel 26, via UDP multicast. Flooding was not used, but a double send was used to decrease the chance of missed messages.

A beacon was a soda can wrapped in a unique barcode which could be read by the robot's camera. Each such beacon was associated with a unique sensor mote. The food and nest were larger "beacons", also wrapped in a unique barcode each, and associated with unique sensor motes. These objects were larger to make it easier for more robots to congregate near them.

When a robot approached a beacon, the robot's onboard sensor mote could exchange messages with the beacon's sensor mote. Though our environment was small enough that most sensor motes could see one another, we required that robots would not communicate with one another, nor with distant beacons, nor would beacons communicate with one another.

Simulator. There were several physical limitations on both our robots and our environment. First, the battery life prevented us from running large numbers of robots for a prolonged period of time. Second, we were physically limited in the size of the field. Third, the speed of the physical experiments was too slow to gather more than a few runs per treatment. Fourth, it was not possible to smoothly "ramp down" the sensor motes to introduce noise and reliability issues. For these reasons we also developed a simulator testbed which tried to replicate the physical environment, robots, beacons, and many of the issues involved in their operation (such as occlusion, sensor mote failure, etc.).

We will begin with a physical robot experiment and a simulator validation, then discuss simulated experimental results under various conditions. We will discuss details special to the simulator later with its experiments.

Experimental Details. The robot system had the following user-defined parameters, expressed as probabilities. These will be specified in the experimental sections:

P_{Follow}	How often the robot tried to follow the gradient.
$P_{Explore}$	How often the robot would begin exploring.
$P_{Acquire}$	How often the robot tried to acquire a beacon.
P_{Return}	How often a robot would return a beacon upon arrival to the nest with food.
P_{Deploy}	How often the robot tried to deploy a beacon.
$P_{Optimize}$	How often the robot tried to optimize a beacon.
P_{Remove}	How often the robot tried to remove a beacon.

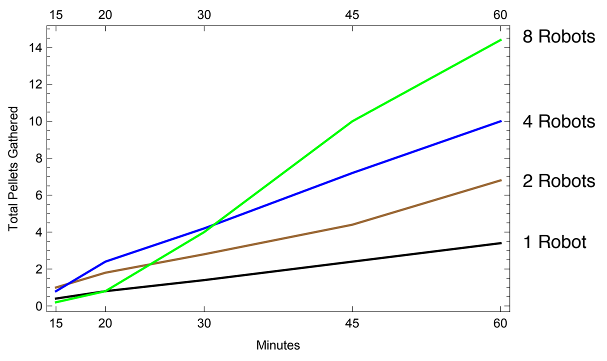

Figure 2: Performance of varying numbers of physical robots in environment with pre-deployed beacons. Measurement is sum total food pellets gathered so far.

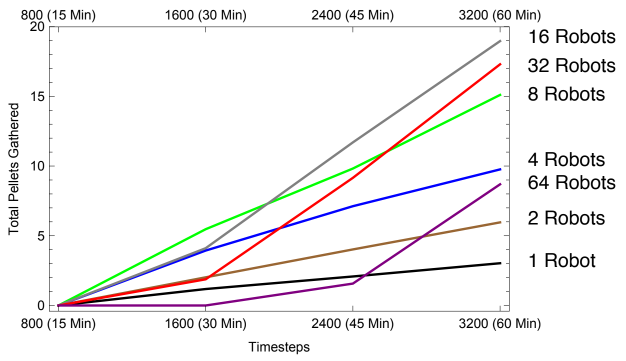

Figure 3: Validation of simulated model, with field size identical to physical robots in Figure 2. Note that 800 simulation timesteps roughly equals 15 minutes of physical robot time. Measurement is sum total food pellets gathered so far.

Additionally, the robots had the constant ExploreCountdown, which defined how many beacon-hops a robot would travel before it stopped exploring.

All experimental runs were done 100 times each, except for the physical robot tests. Experiments were verified for statistical significance using an ANOVA at $p = 0.05$ with a Tukey post-hoc test.

5. PHYSICAL ROBOT EXPERIMENT AND VALIDATION

Our first experiment exhibited the basic features of the algorithm and tested performance. We pre-deployed a grid of 28 beacons in the environment, including the nest and food. The grid of beacons included a reasonable trail from food to nest, but did not establish a pheromone gradient for the rest of the field. Since a path was already established, the robots were given parameters conducive to foraging: $P_{Follow} \leftarrow 0.95$, $P_{Explore} \leftarrow 0.05$. All other probabilities were set to 0. ExploreCountdown was set to 1, so that random walks were performed for only one step.

We varied the number of robots to observe performance. 5 runs each were performed for teams of 1, 2, 4, and 8 robots. Each run lasted one hour, and the amount of food gathered so far was recorded at 15, 20, 30, 45, and 60 minutes. Robots were released one at a time (half-way through the survey step of the previous robot) to avoid traffic jams near the nest and to prevent robots from panicking due to complete occlusion at the start.

Results. Figure 2 shows the performance results for various numbers of physical robots in our testbed. These results were verified with an ANOVA at the 60 minute mark: all differences are statistically significant.

While it might be expected that increasing the number of robots would have a beneficial effect on the amount of food gathered, this was in not certain at the outset. Occlusion was a serious issue in finding and using the optimal path, and traffic jams were common when multiple robots were on the field. However, robots would take advantage of their explore and wander behaviors to visit nearby beacons, establishing new paths or merely get out of the way of other robots. As a result the speedup was sublinear but substantial.

Of additional interest was the visibly slower start to the 8 robot experiments. This was due to early traffic jams among them. Once the robots had spread out, the collection rate sped up.

5.1 Simulation Validation

We next sought to validate the simulator against the physical robot results. Our simulator had several additional parameters meant to help approximate certain physical realities of interest:

$P_{Occlude}$ When two robots are near one another, one robot may occlude certain beacons normally visible to the other robot. Many variables determine the occlusion. To keep things simple, we defined an occlusion parameter ($P_{Occlude}$) which described the probability that a robot next to a beacon would occlude that beacon from another robot. Occluded beacons were removed from the *neighbors* list.

$P_{Unreliable}$ This described the (usually low) probability that a robot could not properly communicate with a sensor mote. A malfunctioning beacon was removed from the *neighbors* list. Additionally, if Rand($P_{Unreliable}$) was true, then UpdatePheromones(...) would immediately fail.

P_{GiveUp} When a robot bumps into another robot, it tries to maneuver around it and reestablish visual contact with its target beacon. This can be difficult if the area is crowded. In simulation this was handled with a probability P_{GiveUp} of giving up going to a target beacon after bumping into and going around a robot.

The time to do twelve iterations of the loop (roughly equivalent to crossing the field and returning) was measured in the simulator to be approximately 800 "steps" and on the physical robots to be about 15 minutes. Other configurable parameters include: the field size (65×100 units in the validation experiment), the nest location $(50, 95)$, the food location $(7, 7)$, the robot radius (2), the beacon range (15), the minimum distance from a beacon needed to interact with it (2), the robot forward speed (0.2 units per timestep), the robot rotation speed (1 radian per timestep), and the maximum number of beacons a robot could hold (1). The settings for these correspond with a translation of the real field into simulated space.

Using this simulator, we established the same setup as the physical robot experiment, with 28 predeployed beacons (including the food and nest) and a predefined path. Robots were introduced to the field one at a time. and parameters were set to either the same as the real robot experiments or, for the simulator specific parameters, they were set to mimic the physical attributes of the robots. Specifically, we set $P_{Occlude} \leftarrow 0.25$ based on the number of robots that could physically surround a beacon, and $P_{GiveUp} \leftarrow 0.5$ based

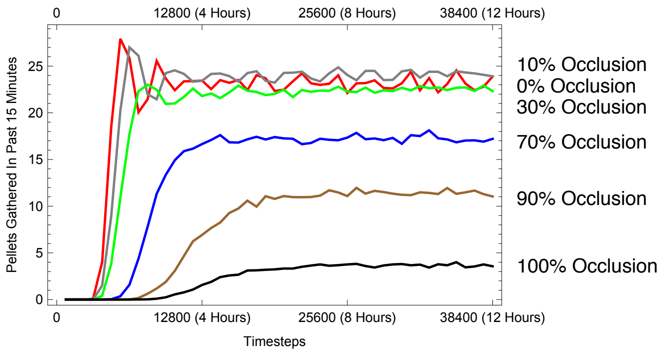

Figure 4: Occlusion results, with no beacon removal or optimization. Measurement is food pellets gathered in the past 15 minutes.

% Occlusion	% Removal and Optimization		
0%	0%	25%	50%
10%	50%	25%	0%
30%	50%	25%	0%
70%	50%	0%	25%
90%	0%	25%	50%
100%	0%	25%	50%

Table 1: Occlusion Results by Removal and Optimization Probability. The various removal and optimization behavior probability settings are sorted by performance, with higher performance further to the right. Overbars group together settings with no statistically significant differences among them. Compare to Figure 5.

on observation of the robots during the physical robot experiments. By default we also left $P_{Unreliable} \leftarrow 0$.

The number of robots in the field was varied in the same way as the physical robot experiments, except that 100 runs were performed for 1, 2, 4, 8, 16, 32, and 64 robots. Each run was a simulation of 1 hour (3200 steps) with the amount of food recorded at 15, 30, 45, and 60 minutes.

Results. Figure 3 shows the performance results for 1, 2, 4, 8, 16, 32, and 64 robots in the simulation environment. These results were verified with an ANOVA at the 3200 timestep (60 minute) mark. All differences are statistically significant.

The speedup was nearly the same as for the (1–8) physical robots, and continued until 16 robots. At this point crowding began to take its toll, causing performance to drop with 32 robots, then significantly drop with 64 (these also had increasingly slow start-up times). With 128 robots (not shown), the swarm was paralyzed. This was expected due to the relative size of the robots to the field.

6. SIMULATION EXPERIMENTS

We then used the simulation to examine the robustness of the method under different physical challenges. To test with more robots, we increased the size of the field (to 100×100, with the nest at $10, 10$ and the food at $90, 90$), reduced the size of the robots (to 0.5), and slightly increased the beacon range (to 20). We also extended the experiment length to 12 hours (38400 timesteps). We no longer used pre-deployed beacons: in all experiments, robots were responsible for deploying beacons and were permitted only a fixed number of beacons to use in the field (160 total beacons). These changes allowed us to use 100 robots in our simulation experiments.

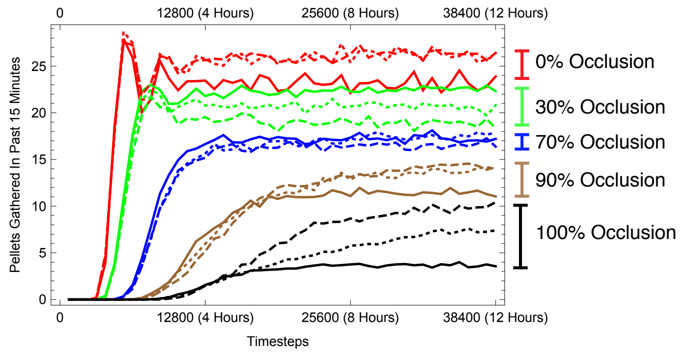

Figure 5: Full occlusion results. 10% Occlusion removed for clarity (they are similar to 30% Occlusion). Solid lines —— indicate no beacon removal or optimization (same as in Figure 4). Dotted lines - - - - indicate 25% probability of beacon removal or optimization. Dashed lines – – – indicate 50% probability of beacon removal or optimization. Measurement is food pellets gathered in the past 15 minutes. Compare to Table 1.

For the parameters, we started with defaults similar to the physical robot experiments, and adjusted them to more efficiently explore the environment, forage, and ferry. They were: $P_{Follow} \leftarrow 0.9$, $P_{Explore} \leftarrow 0.05$, $P_{Acquire} \leftarrow 0.75$, $P_{Return} \leftarrow 0.5$, $P_{Deploy} \leftarrow 0.25$, $P_{Optimize} \leftarrow 0$, $P_{Remove} \leftarrow 0$, $P_{GiveUp} \leftarrow 0.05$, $P_{Occlude} \leftarrow 0.01$, $P_{Unreliable} \leftarrow 0$.

6.1 Beacon Occlusion

Resource blocking and occlusion is a common problem for swarm robots, especially when robots must share resources (such as the beacons). We expected our algorithm to behave robustly with regard to this issue. The default simulation parameters were used, except $P_{Occlude}$, which was given the values 0.0, 0.1, 0.3, 0.7, 0.9, and 1.0.

Results. Figure 4 shows the performance of the system over the simulated twelve hour period, measured by number of food pellets returned to the nest in the past fifteen minutes. An ANOVA of the sample at hour 12 indicates that the 0%, 10%, and 30% occlusion amounts are not statistically significantly different: but all other amounts are. It should be noted that 100% occlusion means that a robot using a beacon will always occlude it.

As can be seen, small amounts of occlusion are not particularly detrimental: but the effect quickly grows with more occlusion.

Effect of Beacon Removal and Optimization. As noted in [9], beacon removal and optimization would be expected to significantly improve performance as robots straighten out the path to the food. However occlusion can have a serious detrimental impact on both removal and optimization, as these behaviors are heavily reliant on being able to accurately see p and n as well as make changes based on the *neighbors* list. We wondered what impact varying levels of occlusion would have on them.

To this end, we re-ran the same occlusion experiment two more times, once with $P_{Optimize}$ and P_{Remove} both set to 0.25, then again with both of them set to 0.5.

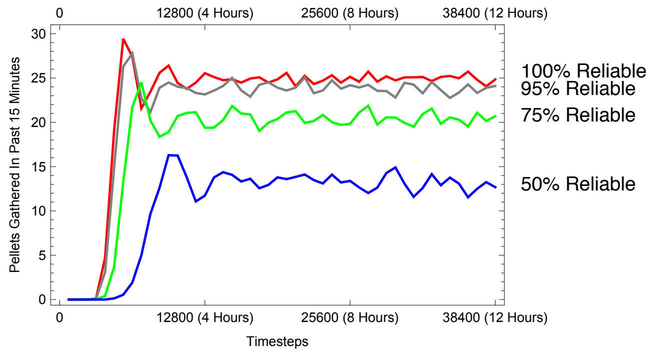

Figure 6: Beacon reliability results. Measurement is food pellets gathered in the past 15 minutes.

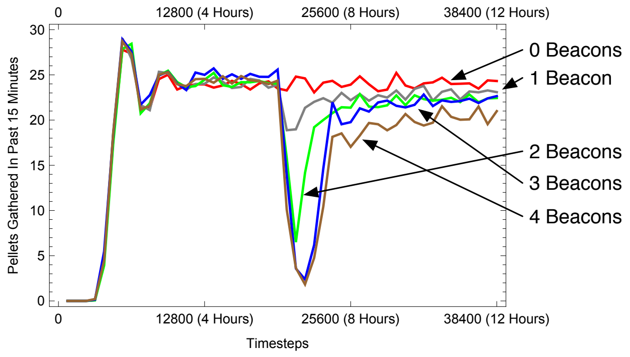

Figure 7: Beacon removal recovery results, for varying width strips removed (measured in number of beacon ranges across). Measurement is food pellets gathered in the past 15 minutes.

Figure 5 expands on Figure 4, including results when the robots use beacon removal or optimization with a 25% or 50% probability. For clarity the 10% occlusion results are not shown, as they are the same pattern as the 30% results. Table 1 summarizes the statistical significance results, verified with an ANOVA at hour 12.

With 0% occlusion, turning on removal and optimization at either level significantly increased its performance over the basic experiment. This result was expected.

As occlusion increased (10% and 30%), increasing removal and optimization *worsened* performance. This seemed natural as higher levels of occlusion would quickly hamper the ability to do removal or optimization: the robots were essentially wasting their time.

What wasn't expected was that at high levels of occlusion (90% and 100%), increasing removal and optimization once again significantly *improved* performance. A closer examination of the results revealed that this was an effect of the maximum number of beacons (160) allotted to the robots. With high levels of occlusion, the robots would deploy large numbers of beacons very densely near the nest, and only sparsely near the food source. When the robots ran out of beacons, optimization and removal enabled them to redistribute the beacons to provide new food routes; without optimization or removal, the robots were often stuck on a difficult-to-see single path.

6.2 Beacon Reliability

A large deployment of beacons might involve hundreds or thousands of sensor motes. This would require small, inexpensive disposable units likely to have reliability issues, especially if deployed over a period of years. We wanted to test the robustness of our approach over different amounts of beacon reliability. To this end the same default parameters were used, but we varied $P_{Unreliable}$. Optimize and deletion were not permitted for this experiment.

Figure 6 shows the results when the robots are operating under varying levels of beacon reliability. The differences between 100% and 95% reliability are not statistically significant, but all other differences are. The percentage of reliability almost perfectly linearly correlates with performance.

Since unreliability acts very much like a occlusion, but with additional consequences that can harm pheromone updates (keeping a beacon out-of-date compared to its neighbors) we expected that reliability would be an issue. However, small amounts of unreliability (for example, 5%) were handled robustly by the system.

6.3 Recovery from Beacon Removal

Finally, we wished to consider what would happen if beacons were removed wholesale from the environment, breaking existing paths to the food. Quite unlike the algorithm in [9], our method was meant to robustly handle this situation.

Our test scenario was as follows: The default parameters were used, such that removal and optimization were disabled. We then allowed the robots to deploy beacons and establish a path from the nest to the food. We then eliminated a roughly vertical strip of beacons centered in the field and pushed the robots in this area to the border of the strip (depending on which side they were closest to). This strip was some n number of beacon ranges wide: we chose values from 1 to 4 (20, 40, 60, and 80 units wide). This final value removed nearly all the beacons on the field between the nest and the food (the field being only 100 units wide). As a control, we included the scenario where no beacons were removed ($n = 0$).

Results. As shown in Figure 7, the robots were clearly capable of recovering from catastrophic beacon failure. Recovery was not total, but it was close. As expected, the rate and degree of recovery was largely related to the number of beacons removed.

7. CONCLUSION AND FUTURE WORK

We have demonstrated a multi-pheromone foraging algorithm deployed to real robots, and which uses stores pheromone information in mobile wireless sensor motes. We showed the technique running on real swarm robots and in simulation, and demonstrated how the algorithm deals robustly with noise, occlusion, and catastrophic beacon failure.

We have barely tapped into the potential of wireless sensor motes for with swarm robotics. Wireless sensor motes can hold much richer information than a few pheromones and flags and they have *sensors* on them which can assist robots. We intend to explore how this might benefit collaboration among the robots to do more complex behaviors other than collective foraging. Sensor motes can also easily *communicate with one another*, something ignored in our paper. This is no longer an indirect communication mode, but sensor motes are designed to do this communication efficiently and at extremely low power. We intend to further explore how this combination: robot-deployable and re-deployable beacons, plus both local and ad-hoc wireless communication, might enable much less trivial robot behaviors.

8. ACKNOWLEDGMENTS

The work presented in this paper is supported by NSF NRI grant 1317813.

REFERENCES

[1] E.J. Barth. A dynamic programming approach to robotic swarm navigation using relay markers. In *Proceedings of the 2003 American Control Conference*, volume 6, pages 5264–5269, June 2003.

[2] E. Bonabeau. Marginally stable swarms are flexible and efficient. *Phys. I France*, pages 309–320, 1996.

[3] Colin Chibaya and Shaun Bangay. A probabilistic movement model for shortest path formation in virtual ant-like agents. In *Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, pages 9–18. ACM, 2007.

[4] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Insect Behavior*, 3:159–168, 1990.

[5] Frederick Ducatelle, Gianni A. Di Caro, Alexander Förster, and Luca Gambardella. Mobile stigmergic markers for navigation in a heterogeneous robotic swarm. In *Swarm Intelligence (ANTS)*, pages 456–463. Springer, 2010.

[6] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki: A lightweight and flexible operating system for tiny networked sensors. In *IEEE International Conference on Local Computer Networks*, pages 455–462, 2004.

[7] S. Goss and J. L. Deneubourg. Harvesting by a group of robots. In *First European Conference on Artificial Life*, pages 195–204. MIT Press, 1992.

[8] Nicholas Hoff, Robert Wood, and Radhika Nagpal. Effect of sensor and actuator quality on robot swarm algorithm performance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.

[9] Brian Hrolenok, Sean Luke, Keith Sullivan, and Christopher Vo. Collaborative foraging using beacons. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1197–1204, 2010.

[10] Gideon Kowadlo and R. Andrew Russell. Robot odor localization: A taxonomy and survey. *Int. J. Rob. Res.*, 27(8):869–894, August 2008.

[11] Ralf Mayet, Jonathan Rober, Thomas Schmickl, and Karl Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. In *International Conference on Swarm Intelligence*, 2010.

[12] M. Nakamura and K. Kurumatani. Formation mechanism of pheromone pattern and control of foraging behavior in an ant colony model. In C. G. Langton and K. Shimohara, editors, *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 67–76. MIT Press, 1997.

[13] K.J. O'Hara and T.R. Balch. Distributed path planning for robots in dynamic environments using a pervasive embedded network. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1538–1539, 2004.

[14] K.J. O'Hara, V. Bigio, S. Whitt, D. Walker, and T. Balch. Evaluation of a large scale pervasive embedded network for robot path planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2072–2077, May 2006.

[15] Liviu Panait and Sean Luke. A pheromone-based utility model for collaborative foraging. In *Proceedings of the Third International Joint Conference on Autonomous Angents and Multi Agent Systems (AAMAS)*, pages 36–43, 2004.

[16] H. Van Dyke Parunak, Sven A. Brueckner, and John Sauter. *Digital Pheromones for Coordination of Unmanned Vehicles*, pages 246–263. Springer, 2005.

[17] D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone Robotics. *Autonomous Robots*, 11(3):319–324, 2001.

[18] Anies Hannawati Purnamadjaja and R. Andrew Russell. Bi-directional pheromone communication between robots. *Robotica*, 28(1):69–79, March 2010.

[19] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[20] Andrew Russell. Ant trails: an example for robots to follow? In *IEEE International Conference on Robotics and Automation*, 1999.

[21] J. Svennebring and S. Koenig. Building terrain-covering ant robots: A feasibility study. *Autonomous Robots*, 16(3):313–332, 2004.

[22] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Blazing a trail: insect-inspired resource transportation by a robot team. In *Proceedings of the International Symposium on Distributed Autonomous Robot Systems*, 2000.

[23] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Whistling in the dark: Cooperative trail following in uncertain localization space. In C. Sierra, M. Gini, and J. S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 187–194. ACM, 2000.

[24] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. LOST: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18:796–812, 2002.

[25] M. Wodrich and G. Bilchev. Cooperative distributed search: The ants' way. *Control and Cybernetics*, 26, 1997.

[26] Vittorio Ziparo, A. Kleiner, B. Nebel, and Daniele Nardi. RFID-based exploration for large robot teams. In *IEEE International Conference on Robotics and Automation*, 2007.