Monte Carlo Sensor Networks

Thomas C. Henderson, Brandt Erickson,
Travis Longoria, Eddie Grant*, Kyle Luthy*,
Leonardo Mattos*, and Matt Craver*

UUCS-05-001

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

*Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695 USA

20 January 2005

Abstract

Biswas et al. [1] introduced a probabilistic approach to inference with limited information in sensor networks. They represented the sensor network as a Bayesian network and performed approximate inference using Markov Chain Monte Carlo (MCMC). The goal is to robustly answer queries even under noisy or partial information scenarios. We propose an alternative method based on simple Monte Carlo estimation; our method allows a distributed algorithm, pre-computation of probabilities, a more refined spatial analysis, as well as desiderata for sensor placement in the friendly agent surrounded by enemies problem. In addition, we performed experiments with real microphones and robots to determine the sensor correct response probability.

1 Introduction

Many advances have been made in sensor network technology and algorithms in the last few years. See [14] for an overview of the state of the art. Work has been done on: architecture [9], systems and security [4, 12, 13], and applications [10]. Our own work has focused on the creation of an information field useful to mobile agents, human or machine, that accomplish tasks based on the information provided by the sensor network [6, 2, 3, 5, 8].

Some drawbacks of sensor networks include the need to conserve power and not run all the nodes all the time (partial data), and sensors are noisy (sometimes return the wrong value). This motivates a statistical approach to decision making. As described in the abstract, Biswas et al. [1] introduced an interesting problem in the context of sensor networks, as well as an MCMC solution. We present an alternative approach here.

1.1 The Problem

Suppose there is a 2D area of interest (we'll consider the unit square), and a friendly agent located at a fixed position, LF, in the area. In addition, there are m sensor nodes located throughout the area (these sensor nodes report the presence or absence of enemy agents within some fixed range). Finally, n enemy agents are placed in the area; each enemy agent's coordinates are chosen by independently sampling a uniform distribution for x and y.

Query: Given a set of sensor responses, R_i (each declares there is or is not an enemy within its range), and a set of probabilities of the correctness of the responses, what is the probability, $P(S \mid R_i)$, that the friendly agent is surrounded by the enemy agents (surrounded means that it is within the convex hull of the enemy locations).

1.2 Previous Work

The authors cited above set up a Bayesian network with a given a priori probability of correctness of the sensor responses, and conditional probabilities on the locations for the enemies given the responses, and also a conditional probability of being surrounded given the enemy locations. They present results of answering the query for two examples, and

investigate the affect of various parameters on the posterior probability; these parameters include: number of sensors, number of enemies, sensing range and MCMC convergence.

2 Monte Carlo Sensor Networks

2.1 Sensor Model

We take a slightly different sensor model than Biswas et al. Let the ground truth be defined as:

$$\hat{d_i} = \begin{cases} 0 & \text{if there's no enemy in range of sensor i} \\ 1 & \text{otherwise.} \end{cases}$$

Then, let:

$$d_i = \begin{cases} \hat{d_i} & \text{with probability } \rho \\ \neg\hat{d_i} & \text{with probability } 1-\rho \end{cases}$$

2.2 No Sensor Data Available

In the absence of any sensor data, there is an a priori probability that the friendly agent is surrounded. We use Monte Carlo simulation to estimate this probability (see Figure 1 for $n = 3 \ldots 6$). Figure 2 shows the a priori probability that the center point (0.5,0.5) is surrounded as the number of enemies goes from 3 to 10.

2.3 One Sensor Case

To determine the probability when sensor data is available, we use Monte Carlo as follows. Consider the case when 1 sensor responds; i.e., $R := d_1$ is the sensor response and is correct with probability ρ. Then:

$$\begin{aligned} Prob(S \mid R) = \rho MC(\text{assume } d_1) \\ +(1-\rho)MC(\text{assume } \neg d_1) \end{aligned}$$

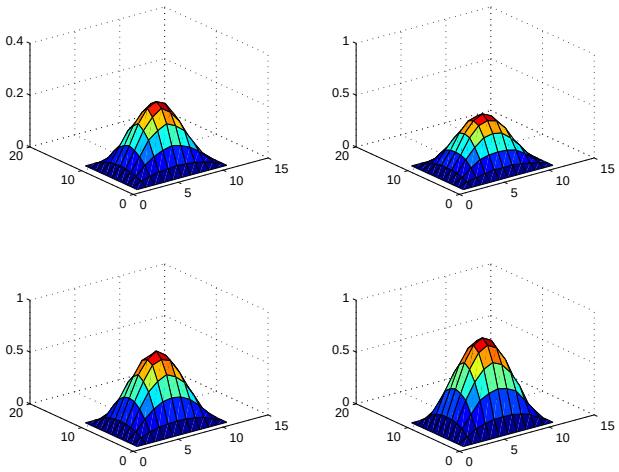

Figure 1: A Priori Probability that Point in Unit Square is Surrounded (for 3,4,5 and 6 enemies).

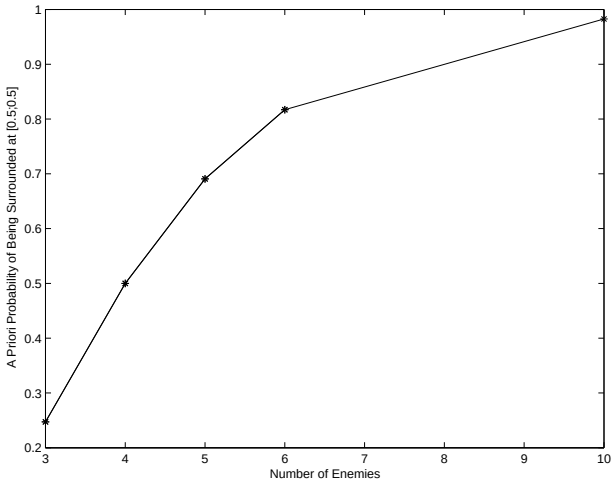

Figure 2: A Priori Probability that Point in Center of Unit Square is Surrounded (for 3,4,5,6 and 10 enemies).

where $MC(\cdot)$ means to run the Monte Carlo simulation with condition $\cdot$ set. That is, if d_1 is 1, then fix one enemy at LS_1 (sensor 1's location); if d_1 is 0, then don't use any sample with an enemy in range of LS_1.

2.4 m-Sensor Case

The same approach is used, but now there are 2^m terms in the probability calculation. Let $\mathbf{d} = [d_1, d_1, \ldots, d_m]$ be the set of sensor node assertions about the presence of enemies. We must consider all combinations of these being true or false. This is characterized by a vector $\mathbf{b} = [b_1, b_1, \ldots, b_m]$ and the contribution of each to the total probability of being surrounded:

$$Prob(S \mid R) = \sum_{\mathbf{b} \in \mathcal{P}(\{0,1\}^m)} Prob(\mathbf{b}) Prob(S \mid \mathbf{b})$$

For example, if we assume that all the sensor nodes report that there is an enemy present, and the first $m-1$ sensor responses are correct but that sensor node m is wrong, then we compute that term in the above summation as:

$$\rho^{n-1}(1-\rho) Prob(S|Positions)$$

where $Positions$ indicates that Monte Carlo is run assuming enemies in positions indicated by nodes 1 to $m-1$ and no enemy in range of node m.

3 Examples

We give here our results for the experiments of Biswas et al. (We estimate the locations from their figures!) The layout of Experiment 1 is shown in Figure 3. The corresponding table is given in Table 1; we give both our results (MC) and theirs (MCMC) for comparison.

R	MC	MCMC
56, 68	37	44
none	49	51
2, 41	87	80
41	82	81
41, 56, 68	99	86
36, 37, 41, 56, 68	99	93

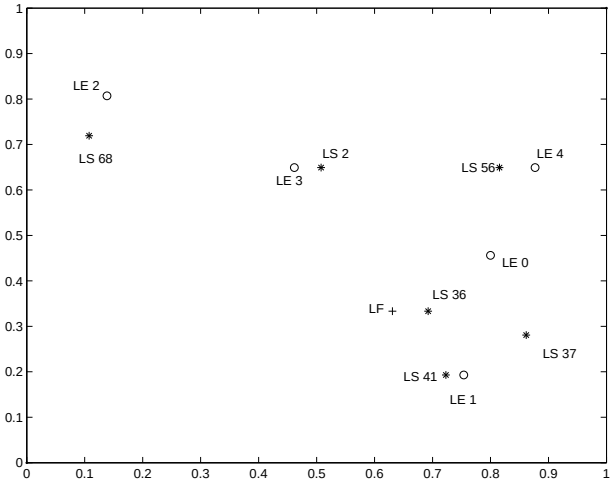

Figure 3: Biswas et al. Experiment 1 Layout of friendly agent ($+$), enemy agents, LE ($\circ$) and sensors, LS ($*$).

Table 1. Results from Experiment 1.

Figure 4 shows the layout of Experiment 2, and Table 2 shows the results obtained from both the MC and MCMC methods.

R	MC	MCMC
none	68	67
35, 50, 58	66	70
8, 36, 58	66	74
8, 36, 46	74	75
8, 35, 50, 58	87	78
8, 35, 36, 46, 50, 58, 75, 86	96	91

Table 2. Results from Experiment 2.

Figure 5 shows the convergence of the Monte Carlo process as the number of trials is increased. These results are for row 2 of Table 2, and for the number of trials ranging from 10 to 10^4.

We also ran a simulation experiment in which the sensor responses' correctness was sampled from the appropriate distribution (uniform, 70% correct; see physical experiments section). We ran one case with the friendly agent surrounded and one case with it not surrounded. (We used Experiment 2 layout for the former, and switched LF and LE 3 for the latter.) Table 3 gives the mean posterior found and the variance (the no sensor case is

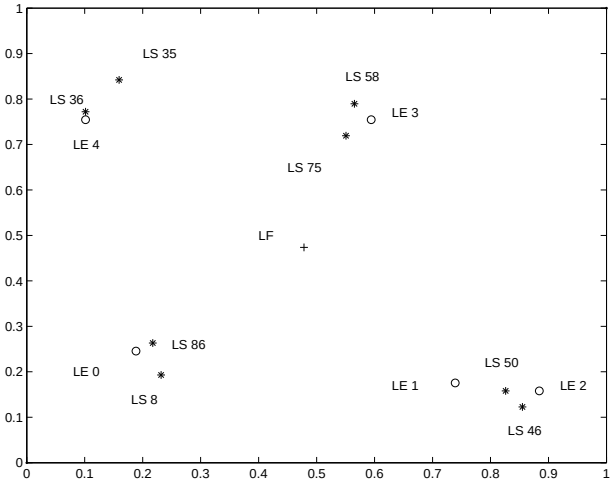

Figure 4: Biswas et al. Experiment 2 Layout of friendly agent (+), enemy agents, LE (○) and sensors, LS (∗).

eliminated).

Surrounded Case (Mean; Variance)	Not Surrounded Case (Mean; Variance)
(0.6776;4.7008e-005)	(0.4216;2.0093e-004)
(0.6904;6.2806e-005)	(0.3985;3.3633e-004)
(0.7298;1.0097e-00)	(0.3019;1.2547e-004)
(0.7629;2.3356e-004)	(0.3571;2.1346e-004)
(0.8418;5.5877e-004)	(0.1093;2.7538e-004)

Table 3. Re-run Experiment 2 with sensor correctness sampling and both with LF surrounded and not surrounded.

4 Physical Experiments

A set of experiments was undertaken to better understand the physical nature of acoustic detection systems, as well as their probability to produce a correct report of enemy presence. We determined the correctness probability to be around 70%.

For physical testing it was decided to use the EvBot II autonomous mobile robot testbed

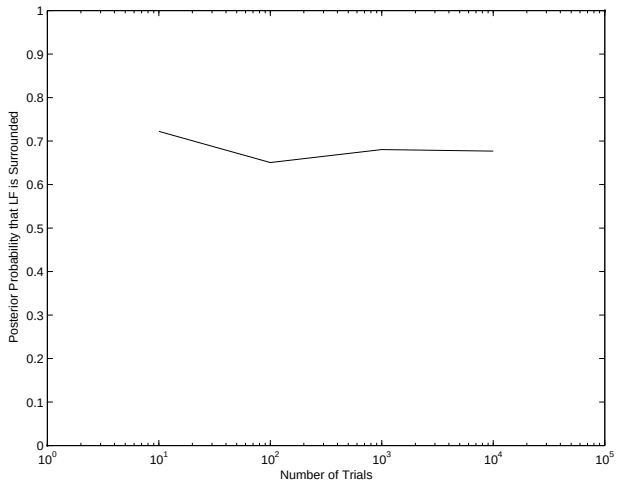

Figure 5: Convergence of MC for row 2, Table 2.

in the Center for Robotics and Intelligent Machines (CRIM) at North Carolina State University [11]. The autonomous mobile robot platforms in the testbed, the EvBots, are used widely for evolutionary algorithm and distributed sensor experimentation. To test the Mote Carlo sensor network at the heart of this paper a simple microphone circuit was constructed to provide the data for hit or miss decision making. For these experiments the EvBot II robots served as stationary sensor nodes. The enemy in this case is a speaker providing sound within a set range. The microphone circuit consists of a two stage amplifier, the output of which is rectified to yield a DC approximation of the amplitude. A DC approximation removes the need for rapid sampling and frequency determination and gives an integer value after analog to digital conversion. An example of the output of this circuit is shown in Figure 6. There is a slight voltage drop (approximately 0.7V) between the sound source and the rectified signal due to the rectifying diode. This serves to reduce the effective range of the ADC measurement (8 bits) from 1024 to around 930.

The output of the microphone circuit is sampled through an ADC channel that is part of the circuitry of a BasicX 24 microcomputer; which in turn is part of the EvBot II platform. Multiple robots can easily be used to represent a distributed network by utilizing their built in wireless networking. While not necessarily a valid representation of a true distributed sensor network (due to size, processing power, battery life, and cost), they serve as a good test-bed to provide proof of concept. An EvBot fitted with a sensor circuit is shown in Figure 7.

To provide a listening range for the robot, a 100Hz sound source was placed 22.25 inches from the microphone. The sound level was set at an amplitude value that equates to a sensor reading of 512, i.e., half the range of the ADC. Any fluctuations observed in the recorded

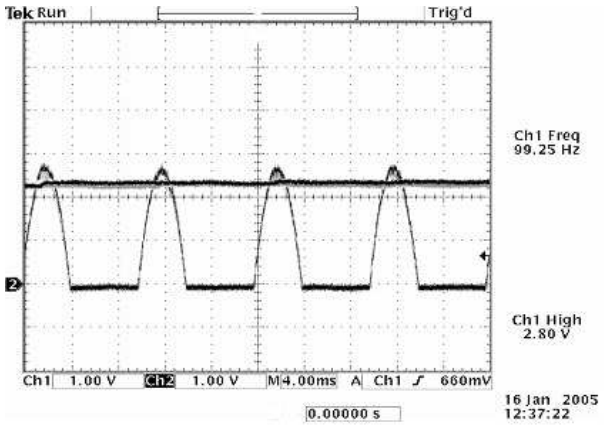

Figure 6: The Sensed Sound and the Rectified DC Approximation.

data were considered as being a function of a combination of sensor circuit design and variations in the sound source. The sound was calibrated using a BK Precision model 732 digital sound level meter. For this setup, a level of 107.6dBC equates to approximately half ADC range at 22.25 in. This calibration was performed in the center of the environment to reduce error caused by noise reflection from the walls in the room.

A series of hit or miss experiments was performed in order to characterize the sensor. The EvBot was placed at three positions within the environment, the center, the center to one side, and in a corner, see Figure 8. At each of these positions the sound source was placed at 11.125 inches, 22.25 inches, and 33.375 inches, well within range, putting it just in range (defined previously as 22.25 inches), and well out of range. Likewise, at each range, data from 3 threshold values were recorded; one at 512 (used to define the original range), one at 256, and one at 768. Each measurement location and threshold is recorded 100 times. Furthermore, this experiment is performed on two different sensor/EvBot systems.

The sound level was measured at the sensor while at standard range in each of the three positions in the environment. While in the center, the amplitude was measured as 93.3dBC. At the side, this number is higher at 94.2dBC, and similarly, in the corner, 94.6dBC is measured. All three systems are 22.25 inches from the same source. This is an indication of sound level variation due to location, most likely surface reflections.

Examination of the data in Tables 4 and 5 shows the variation in hit and miss results due to the microphone sensor changing between the positions, ranges, and detection thresholds (T is the threshold for enemy detection).

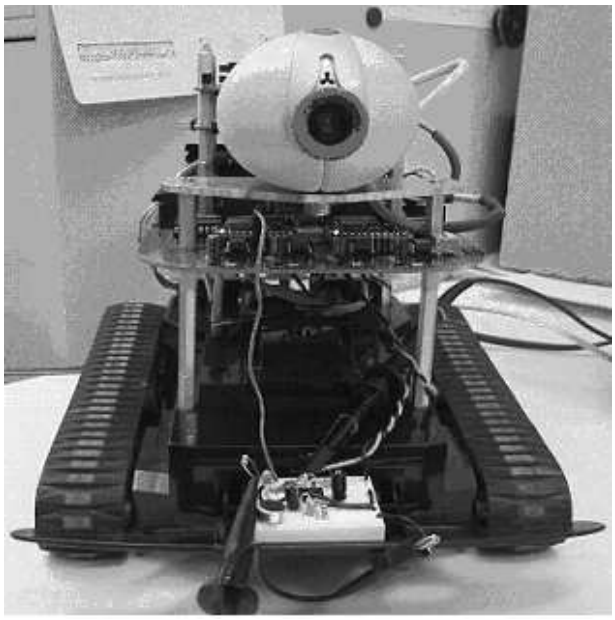

Figure 7: EvBot II with Affixed Sound Level Detection Circuit.

		Center		*Side*		*Corner*	
		Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
Half Range	T = 256	100	0	100	0	100	0
(11.125 inches)	T = 512	100	0	100	0	100	0
	T = 768	100	0	100	0	100	0
Full Range	T = 256	100	0	100	0	100	0
(22.25 inches)	T = 512	13	87	100	0	100	0
	T = 768	0	100	0	100	0	100
Far Range	T = 256	90	10	0	100	0	100
(33.375 inches)	T = 512	100	0	100	0	100	0
	T = 768	100	0	100	0	100	0

Table 4. Characterization of Correct Decisions
from Recorded Sensor Data (EvBot 7).

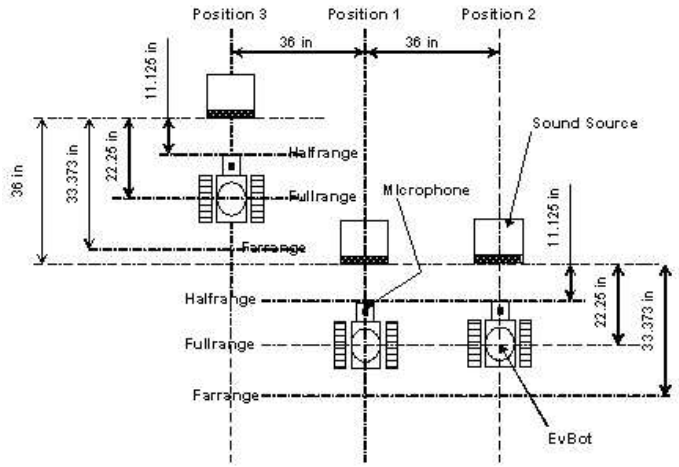

Figure 8: Experimental Setup.

		Center		*Side*		*Corner*	
		Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
Half Range	T = 256	100	0	100	0	100	0
(11.125 inches)	T = 512	100	0	100	0	100	0
	T = 768	100	0	100	0	100	0
Full Range	T = 256	100	0	100	0	100	0
(22.25 inches)	T = 512	0	100	100	0	100	0
	T = 768	0	100	0	100	0	100
Far Range	T = 256	100	0	0	100	0	100
(33.375 inches)	T = 512	100	0	100	0	100	0
	T = 768	100	0	100	0	100	0

Table 5. Characterization of Correct Decisions
from Recorded Sensor Data (EvBot 4).

To ensure that any variation in the sensor system is accounted for experimentally, two sets of experiments were carried out using two separate robots, i.e., EvBot 4 and EvBot 7. Whether or not there is a hit or a miss is determined by:

1. The position of the sensor within the environment (are there reflections off of walls?)
2. The threshold level used to determine whether or not an enemy is in the predefined range (a higher threshold will miss enemies in range, a lower will identify enemies out of range)

3. Variations between sensor systems.

The experiment proved that the percentage of correctness of hit or miss depends upon a variety of factors. When the system is set up such that the threshold is mid-range then the decision of hit or miss is predictable in cases where the sensor system is half-range and far-range, these give 100% hit and 100% miss, respectively. At the distance associated with the threshold, i.e., full-range, the percentage correctness obtained varies because of the variations in sound source, sensor, or ambient noise within the testbed. The experiments also proved that the position in the environment also plays an important role in the correctness of the results obtained, e.g., sound levels increase near a wall, due to sound reflection. This was verified using the BK sound meter.

5 Discussion and Conclusions

We have shown that a simple Monte Carlo approach gives good results in the presence of noise and partial information, and is comparable to the MCMC approach. Moreover, the Monte Carlo calculations can be performed off-line (once the positions of the sensors are known) and downloaded to them and a table lookup used to answer queries once the sensor responses are known. Alternatively, the computation is distributed among the sensor nodes if performed in real-time. Higher accuracy can be achieved by increasing the number of trials (note that this can be done incrementally – e.g., run ten 1000 trial experiments, then later run another ten, and keep averaging $P(S \mid R)$.

One assumption that we make in running the Monte Carlo experiments is that if multiple sensor nodes report the presence of an enemy agent, and if these sensor nodes have a non-empty intersection, then we assume that there is just one enemy agent and it is placed in the intersection area. We intend to explore the significance of better area estimates for enemy location in overlapping sensor areas. This and other slight differences between the MC and MCMC methods accounts for the differences in specific values in the common experiments.

Our results indicate that it is also possible to get a good level of confidence by taking separate singleton or pairs of sensor responses, and use the max probability of those to answer the query. Moreover, it is possible, given the friendly agent's location, to determine sensor placements that decrease the number of responses needed to produce a high-quality answer.

In future work, we intend to explore these issues in a 100-node sensor network testbed now under construction; the EvBots will provide the mobile agent platforms for use as friendly and enemy agents. Moreover, we are exploring the application of sensor networks to snow and avalanche monitoring[7], and in this case the goal might be to surround (e.g., the signal of a buried person) with mobile agents or people – thus, sometimes it may be of interest to ensure that an object of interest is surrounded.

References

[1] R. Biswas, S. Thrun, and L. Guibas. A probabilistic approach to inference with limited information in sensor networks. In *LCSN*, Berkeley, CA, April 2004.

[2] Y. Chen. Snets: Smart sensor networks. Master's thesis, University of Utah, Salt Lake City, Utah, December 2000.

[3] Y. Chen and T. C. Henderson. S-nets: Smart sensor networks. In *Proc International Symp on Experimental Robotics*, pages 85–94, Hawaii, Dec 2000.

[4] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly resilient, energy effi cient multipath routing in wireless sensor networks. *Mobile Computing and Communications Review*, 1(2), 2002.

[5] T. C. Henderson. Leadership protocol for s-nets. In *Proc Multisensor Fusion and Integration*, pages 289–292, Baden-Baden, Germany, August 2001.

[6] T. C. Henderson, M. Dekhil, S. Morris, Y. Chen, and W. B. Thompson. Smart sensor snow. *IEEE Conference on Intelligent Robots and Intelligent Systems*, October 1998.

[7] T. C. Henderson, E. Grant, K. Luthy, and J. Cintron. Snow monitoring with sensor networks. In *Proceedings IEEE Workshop on Embedded Networked Sensors*, pages 558–559, Tampa, FL, November 2004.

[8] T. C. Henderson, J.-C. Park, N. Smith, and R. Wright. From motes to java stamps: Smart sensor network testbeds. In *Proc of IROS 2003*, Las Vegas, NV, October 2003. IEEE.

[9] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization. Ece, UC Berkeley, October 2002.

[10] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor netwroks for habitat monitoring. In *WSNA 2002*, Atlanta, GA, September 2002.

[11] L. Mattos. The EvBot-II: An enhanced evolutionary robotics platform equipped with integrated sensing for control. Master's thesis, North Carolina State University, Raleigh, NC, May 2003.

[12] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept 2002.

[13] L. Zhang. Simple protocols, complex behavior. In *Proc. IPAM Large-Scale Communication Networks Workshop*, March 2002.

[14] F. Zhao and L. Guibas. Preface. In *Proc of IPSN 2003*, pages v–vi, Palo Alto, CA, April 2003. LNCS.

Appendix A: Matlab Code

```
-------------------------------------------------------
function prob = MC_apriori_prob(delx,dely,num_trials, num_enemies)
%
% MC_apriori_prob - A priori probability a point (in unit square) is
%                   surrounded
% On input:
%     delx (float): step in x direction
%     dely (float): step in y direction
%     num_trials (int): number of Monte Carlo trials
%     num_enemies (int): number of enemy agents to place
%                        (random,uniform)
% On output:
%     prob (mxn matrix): probability (at each location) that point is
%                  surrounded by enemies (i.e., in their convex hull)
% Call:
%      prob = MC_apriori_prob(0.1,0.1,1000,3);
% Author:
%     T. Henderson
%     U of U
%     Jan 2005
%

num_rows = length(0:dely:1.0);
num_cols = length(0:delx:1.0);
prob = zeros(num_rows,num_cols);
first = 1;
col_index = 0;

% Compute on 1/8 of square
for x = 0:delx:0.5             % try randomly chosen
    [x,0.5]
    col_index = col_index+1;
    row_index = num_rows+1;
    for y = 0:dely:x
        row_index = row_index-1;
        count = 0;
        if first==1
```

```
            tic;
        end
        for t = 1:num_trials
            pts = rand(num_enemies,2);
            [Ke,Ae] = convhull(pts(:,1),pts(:,2));
            [Kf,Af] = convhull([pts(:,1);x],[pts(:,2);y]);
            if Af<=Ae   % abs(Af-Ae)<eps
                count = count+1;
            end
        end
        if first==1
            time1 = toc;
            first = 0;
            time_est = time1*(num_rows*num_cols/8)/60
        end
        prob(row_index,col_index) = count/num_trials;
    end
end

% Use symmetries to get other 7/8ths of square
subp = prob((num_rows+1)/2:end,1:(num_cols+1)/2);
[num_rows_subp,num_cols_subp] = size(subp);

for ii = 1:num_rows_subp-1
    for jj = 1:num_cols_subp-ii
        subp(ii,jj) = subp(num_rows_subp-jj+1, num_cols_subp-ii+1);
    end
end
subpul = imrotate(subp,-90);
subpur = imrotate(subp,180);
subplr = imrotate(subp,90);
proba = [subpul(1:num_rows_subp-1,1:num_cols_subp) ...
         subpur(1:num_rows_subp-1,2:num_cols_subp);...
          subp, subplr(:,2:num_cols_subp)];
prob = proba;


-------------------------------------------------------
function v = MC_bad_pt_set(pts,fixed_not_locs,radius)
%
% MC_bad_pt_set - determine if sample contains excluded point
```

```
%

[num_pts,dummy] = size(pts);
[num_locs,dummy] = size(fixed_not_locs);
v = 0;

for p = 1:num_pts
    pt = [pts(p,1);pts(p,2)];
    for loc = 1:num_locs
        ctr = [fixed_not_locs(loc,1);fixed_not_locs(loc,2)];
        if norm(ctr-pt)<=radius
            v = 1;
            return
        end
    end
end


------------------------------------------------------
function prob = MC_complete_surround(LS,d,p,friendly_loc,...
                           num_enemies,radius,num_trials)
%
% MC_complete_surround - compute prob friendly agent surrounded
% On input:
%     LS (nx2 array): sensor locations
%     d (nx1 vector): Boolean sensor response (0: no enemy; 1: enemy)
%     p (nx1 vector): probability sensor is correct
%     friendly_loc (1x2 vector): location of friendly agent
%     num_enemies (int): number of enemies
%     radius (float): range of sensors
%     num_trials (int): number of Monte Carlo trials
% On output:
%     prob (float): probability that friendly agent is surrounded
% Call:
%     p = MC_complete_surround([0.1,0.1;0.8,0.8],[1,1],[0.8,0.8],...
                                [0.5;0.5],5,0.12,10000);
% Author:
%     T. Henderson
%     U of U
%     Spring 2005
%
```

```
[num_sensors,dummy] = size(LS);

Pa = zeros(2^num_sensors,1);
MC_prob = zeros(2^num_sensors,1);
num_cases = 2^num_sensors;
for c = 0:num_cases-1
    fixed_locs = [];
    fixed_not_locs = [];
    Pc = 1;
    for bs = 1:num_sensors
        b = num_sensors-bs+1;
        if (bitget(c,b) == 1)
            Pc = Pc*p(b);
            if (d(b)==1)
                fixed_locs = [fixed_locs; LS(b,1), LS(b,2)];
            else
                fixed_not_locs = [fixed_not_locs; LS(b,1), LS(b,2)];
            end
        else
            Pc = Pc*(1-p(b));
            if (d(b)==1)
                fixed_not_locs = [fixed_not_locs; LS(b,1), LS(b,2)];
            else
                fixed_locs = [fixed_locs; LS(b,1), LS(b,2)];
            end
        end
    end
    MC_prob(c+1) = MC_fixed_surround(num_trials,friendly_loc,...
        fixed_locs,fixed_not_locs,num_enemies,radius);
    Pa(c+1) = Pc*MC_prob(c+1);
end
prob = sum(Pa);


-------------------------------------------------------
function assert = MC_find_assert(assert_raw,range)
%
% MC_find_assert - finds assert points for within range cliques
% On input:
%     assert (nx2 array): set of points
```

```
%     range (float): sensor range
% On output:
%     assert (kx2 array): locations of reasonable locations for cliques
% Call:
%     assert = MC_find_assert([0.1,0.1;0.2,0.2;0.2,0.1;0.8,0.8;...
                                0.8,0.9],0.2);
% Author:
%     T. Henderson
%     U of U
%     Spring 2005
%

assert = [];
if isempty(assert_raw)
    return
end
q = MC_find_cliques(assert_raw,range);
[num_cliques,num_pts] = size(q);
assert = zeros(num_cliques,2);

for c = 1:num_cliques
    indexes = find(q(c,:));
    x_val = mean([assert_raw(indexes,1)]);
    y_val = mean([assert_raw(indexes,2)]);
    assert(c,1) = x_val;
    assert(c,2) = y_val;
end


------------------------------------------------------
function cliques = MC_find_cliques(pts,range)
%
% MC_find_cliques - finds sets of points mutually within range
% On input:
%     pts (nx2 array): set of points
%     range (float): sensor range
% On output:
%     cliques (kxn array): (by row) indexes of mutually within
%                                     range points
% Call:
%     q = MC_find_cliques([0.1,0.1;0.2,0.2;0.2,0.1;0.8,0.8;...
```

```
%                           0.8,0.9],0.2);
% Author:
%     T. Henderson
%     U of U
%     Spring 2005
%

[num_pts,dummy] = size(pts);
x_min = min([pts(:,1)]);
x_max = max([pts(:,1)]);
y_min = min([pts(:,2)]);
y_max = max([pts(:,2)]);

num_x = length(x_min:range/2:x_max);
num_y = length(y_min:range/2:y_max);
clique_array = zeros(num_y,num_x,num_pts);
cliques = [];

for p = 1:num_pts
    x = pts(p,1);
    y = pts(p,2);

    row = (num_y-1)-(floor((num_y-1)*(1-(y-y_min)/(y_max-y_min))))+1;
    col = floor((num_x-1)*(x-x_min)/(x_max-x_min))+1;
    for rr = -2:2
        r_nei = row+rr;
        for cc = -2:2
            c_nei = col+cc;
            if (r_nei>0)&(r_nei<=num_y)&(c_nei>0)&(c_nei<=num_x)
                clique_array(r_nei,c_nei,p) = 1;
            end
        end
    end
end
for r = 1:num_y
    for c = 1:num_x
        v = reshape(clique_array(r,c,1:p),1,num_pts);
        [num_cliques,dummy] = size(cliques);
        if num_cliques==0
            cliques = [v];
        else
```

```
            new_clique = 1;
            for cl = 1:num_cliques
                cur_clique = cliques(cl,:);
                if v<=cur_clique
                    new_clique = 0;
                end
            end
            if new_clique
                cliques_tmp = cliques;
                cliques = [v];
                for cl = 1:num_cliques
                    cl_tmp = cliques_tmp(cl,:);
                    if sum(cl_tmp<=v)<num_pts
                        cliques = [cliques;cl_tmp];
                    end
                end
            end
        end
    end
end

-------------------------------------------------------
function MC_gen_fig2
%
% MC_gen_fig2 - generate the Biswas Figure 2 Layout
%

LF = [(2*16+9);19]/16;
LE = [(3+(4/16)) (1+(10/16));
      (3+(1/16)) (11/16)    ;
      (9/16)     (2+(14/16));
      (1+(14/16)) (2+(5/16));
      (3+(9/16)) (2+(5/16))];
x_base = (4*16+1)/16;
y_base = (3*16+9)/16;
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
LS2 = [2+1/16; 2+5/16]./base;
LS36 = [2+13/16; 1+3/16]./base;
```

```
LS37 = [3+8/16; 1]./base;
LS41 = [2+15/16; 11/16]./base;
LS56 = [3+5/16; 2+5/16]./base;
LS68 = [7/16; 2+9/16]./base;

plot(0,0,'b.',LS2(1),LS2(2),'r*',LS36(1),LS36(2),'r*',...
     LS37(1),LS37(2),'r*',LS41(1),LS41(2),'r*',...
    LS56(1),LS56(2),'r*',LS68(1),LS68(2),'r*',...
    LEs(:,1),LEs(:,2),'co',LFs(1),LFs(2),'g+',1,1,'b.');
text(0.82,0.44,'LE 0');
text(0.73,0.14,'LE 1');
text(0.65,0.19,'LS 41');
text(0.7,0.37,'LS 36');
text(0.89,0.24,'LS 37');
text(0.59,0.34,'LF');
text(0.08,0.83,'LE 2');
text(0.09,0.67,'LS 68');
text(0.43,0.62,'LE 3');
text(0.51,0.68,'LS 2');
text(0.75,0.65,'LS 56');
text(0.88,0.68,'LE 4');


-------------------------------------------------------
function MC_gen_fig3
%
% MC_gen_fig3 - generate the Biswas Figure 3 Layout
%

LF = [(2*16+1);27]/16;
LE = [13/16 (14/16);
      (3+(3/16)) (10/16)    ;
      (3+(13/16)) 9/16;
      (2+(9/16)) (2+(11/16));
      7/16 (2+(11/16))];
x_base = (4*16+5)/16;
y_base = (3*16+9)/16;
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
```

```
LS8 = [1; 11/16]./base;
LS35 = [11/16; 3]./base;
LS36 = [7/16; 2+12/16]./base;
LS46 = [3+11/16; 7/16]./base;
LS50 = [3+9/16; 9/16]./base;
LS58 = [2+7/16; 2+13/16]./base;
LS75 = [2+6/16; 2+9/16]./base;
LS86 = [15/16;  15/16]./base;

plot(0,0,'b.',LS8(1),LS8(2),'r*',LS35(1),LS35(2),'r*',...
     LS36(1),LS36(2),'r*',LS46(1),LS46(2),'r*',...
    LS50(1),LS50(2),'r*',LS58(1),LS58(2),'r*',LS75(1),...
    LS75(2),'r*',LS86(1),LS86(2),'r*',...
    LEs(:,1),LEs(:,2),'co',LFs(1),LFs(2),'g+',1,1,'b.');
text(0.08,0.2,'LE 0');
text(0.18,0.14,'LS 8');
text(0.25,0.26,'LS 86');
text(0.08,0.7,'LE 4');
text(0.04,0.8,'LS 36');
text(0.2,0.9,'LS 35');
text(0.4,0.5,'LF');
text(0.45,0.65,'LS 75');
text(0.55,0.85,'LS 58');
text(0.63,0.77,'LE 3');
text(0.63,0.17,'LE 1');
text(0.92,0.17,'LE 2');
text(0.79,0.22,'LS 50');
text(0.83,0.07,'LS 46');


-------------------------------------------------------
function MC_gen_fig4(num_trials)
%

LF = [(2*16+1);27]/16;
LE = [13/16 (14/16);
      (3+(3/16)) (10/16)    ;
      (3+(13/16)) 9/16;
      (2+(9/16)) (2+(11/16));
      7/16 (2+(11/16))];
x_base = (4*16+5)/16;
```

```
y_base = (3*16+9)/16;
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
LS8 = [1; 11/16]./base;
LS35 = [11/16; 3]./base;
LS36 = [7/16; 2+12/16]./base;
LS46 = [3+11/16; 7/16]./base;
LS50 = [3+9/16; 9/16]./base;
LS58 = [2+7/16; 2+13/16]./base;
LS75 = [2+6/16; 2+9/16]./base;
LS86 = [15/16;  15/16]./base;
radius = (6/16)/x_base;

val(1) = MC_complete_surround([LS8(1),LS8(2)],[1],[0.8],...
                                LFs,4,0.12,num_trials);
val(2) = MC_complete_surround([LS8(1),LS8(2);LS35(1),...
           LS35(2)],[1,1],[0.8,0.8],LFs,4,0.12,num_trials);
val(3) = MC_complete_surround([LS8(1),LS8(2);LS35(1),...
         LS35(2);LS36(1),LS36(2)],[1,1,1],[0.8,0.8,0.8],...
          LFs,4,0.12,num_trials);

plot([0,1,2,3],[0.5,val(1),val(2),val(3)]);


-----------------------------------------------------
function table1 = MC_gen_table1(num_trials)
%
% MC_gen_table1 - generate Biswas Table 1 from Figure 2 Layout
%

table1 = zeros(6,1);
LF = [(2*16+9);19]/16;
LE = [(3+(4/16)) (1+(10/16));
      (3+(1/16)) (11/16)    ;
      (9/16)     (2+(14/16));
      (1+(14/16)) (2+(5/16));
      (3+(9/16)) (2+(5/16))];
x_base = (4*16+1)/16;
y_base = (3*16+9)/16;
```

```
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
LS2 = [2+1/16; 2+5/16]./base;
LS36 = [2+13/16; 1+3/16]./base;
LS37 = [3+8/16; 1]./base;
LS41 = [2+15/16; 11/16]./base;
LS56 = [3+5/16; 2+5/16]./base;
LS68 = [7/16; 2+9/16]./base;
radius = (6/16)/x_base;

%table(1) = MC_complete_surround(num_trials,LFs,...
%       [LS56(1),LS56(2);LS68(1),LS68(2)],[],5,radius);
table1(1) = MC_complete_surround([LS56(1),LS56(2);...
            LS68(1),LS68(2)],[1,1],[0.99,0.99],...
            LFs,5,0.12,num_trials);
table1(2) = MC_complete_surround([],[],[],LFs,5,...
             0.12,num_trials);
table1(3) = MC_complete_surround([LS2(1),LS2(2);...
            LS41(1),LS41(2)],[1,1],[0.99,0.99],...
            LFs,5,0.12,num_trials);
%table(3) = MC_complete_surround(num_trials,LFs,...
         [LS2(1),LS2(2);LS41(1),LS41(2)],[],5,radius);
table1(4) = MC_complete_surround([LS41(1),LS41(2)],...
         [1],[0.99],LFs,5,0.12,num_trials);
%table(4) = MC_complete_surround(num_trials,LFs,...
%         [LS41(1),LS41(2)],[],5,radius);
table1(5) = MC_complete_surround([LS41(1),LS41(2);...
         LS56(1),LS56(2);LS68(1),LS68(2)],[1,1,1],...
         [0.99,0.99,0.99],LFs,5,0.12,num_trials);
%table(5) = MC_complete_surround(num_trials,LFs,...
%         [LS41(1), LS41(2); LS56(1),LS56(2);LS68(1),...
%         LS68(2)],[],5,radius);
table1(6) = MC_complete_surround([LS36(1),LS36(2);...
         LS37(1),LS37(2);LS41(1),LS41(2);...
         LS56(1),LS56(2);LS68(1),LS68(2)],[0,0,1,1,1],...
         [0.99,0.99,0.99,0.99,0.99],...
         LFs,5,0.12,num_trials);
%table(6) = MC_complete_surround(num_trials,LFs,...
   [LS41(1), LS41(2);LS56(1),LS56(2);LS68(1),LS68(2)],...
```

```
%    [LS36(1), LS36(2); LS37(1), LS37(2)],5,radius);


-------------------------------------------------------
function table2 = MC_gen_table2(num_trials)
%
% MC_gen_table2 - generate Biswas Table 2 from Figure 2 Layout
%

table2 = zeros(6,1);
LF = [(2*16+1);27]/16;
LE = [13/16 (14/16);
      (3+(3/16)) (10/16)    ;
      (3+(13/16)) 9/16;
      (2+(9/16)) (2+(11/16));
      7/16 (2+(11/16))];
x_base = (4*16+5)/16;
y_base = (3*16+9)/16;
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
LS8 = [1; 11/16]./base;
LS35 = [11/16; 3]./base;
LS36 = [7/16; 2+12/16]./base;
LS46 = [3+11/16; 7/16]./base;
LS50 = [3+9/16; 9/16]./base;
LS58 = [2+7/16; 2+13/16]./base;
LS75 = [2+6/16; 2+9/16]./base;
LS86 = [15/16;  15/16]./base;
radius = (6/16)/x_base;

table2(1) = MC_complete_surround([],[],[],LFs,...
            5,0.12,num_trials);
table2(2) = MC_complete_surround([LS35(1),LS35(2);...
                 LS50(1),LS50(2);LS58(1),LS58(2)],...
        [1,1,1],[0.8,0.8,0.8],LFs,5,0.12,num_trials);
table2(3) = MC_complete_surround([LS8(1),LS8(2);...
               LS36(1),LS36(2);LS58(1),LS58(2)],...
       [1,1,1],[0.8,0.8,0.8],LFs,5,0.12,num_trials);
table2(4) = MC_complete_surround([LS8(1),LS8(2);...
```

```
            LS36(1),LS36(2);LS46(1),LS46(2)],...
       [1,1,1],[0.8,0.8,0.8],LFs,5,0.12,num_trials);
table2(5) = MC_complete_surround([LS8(1),LS8(2);...
    LS35(1),LS35(2);LS50(1),LS50(2);LS58(1),LS58(2)],...
    [1,1,1,1],[0.8,0.8,0.8,0.8],LFs,5,0.12,num_trials);
table2(6) = MC_complete_surround([LS8(1),LS8(2);...
      LS35(1),LS35(2);LS36(1),LS36(2);LS46(1),LS46(2);...
      LS50(1),LS50(2);LS58(1),LS58(2);LS75(1),LS75(2);...
      LS86(1),LS86(2)],[1,1,1,1,1,1,1,1],...
      [0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8],LFs,5,0.12,...
      num_trials);


-------------------------------------------------------
function MC_test_converge
%
% Get probabilities for sequence to determine convergence
%

LF = [(2*16+1);27]/16;
LE = [13/16 (14/16);
      (3+(3/16)) (10/16)    ;
      (3+(13/16)) 9/16;
      (2+(9/16)) (2+(11/16));
      7/16 (2+(11/16))];
x_base = (4*16+5)/16;
y_base = (3*16+9)/16;
base = [x_base;y_base];
LFs = LF./base;
basea = [base';base';base';base';base'];
LEs = LE./basea;
LS8 = [1; 11/16]./base;
LS35 = [11/16; 3]./base;
LS36 = [7/16; 2+12/16]./base;
LS46 = [3+11/16; 7/16]./base;
LS50 = [3+9/16; 9/16]./base;
LS58 = [2+7/16; 2+13/16]./base;
LS75 = [2+6/16; 2+9/16]./base;
LS86 = [15/16;  15/16]./base;
radius = (6/16)/x_base;
```

```
%vals(1) = MC_complete_surround([LS35(1),LS35(2);...
%        LS50(1),LS50(2);LS58(1),LS58(2)],[1,1,1],...
%        [0.8,0.8,0.8],LFs,5,0.12,10);
%vals(2) = MC_complete_surround([LS35(1),LS35(2);...
%         LS50(1),LS50(2);LS58(1),LS58(2)],[1,1,1],...
%        [0.8,0.8,0.8],LFs,5,0.12,100);
%vals(3) = MC_complete_surround([LS35(1),LS35(2);...
%        LS50(1),LS50(2);LS58(1),LS58(2)],[1,1,1],...
%         [0.8,0.8,0.8],LFs,5,0.12,1000);
%vals(4) = MC_complete_surround([LS35(1),LS35(2);...
%       LS50(1),LS50(2);LS58(1),LS58(2)],[1,1,1],...
%             [0.8,0.8,0.8],LFs,5,0.12,10000);
vals(1) = 0.7224;
vals(2) = 0.6507
vals(3) = 0.6804;
vals(4) = 0.6769;
%vals(5) = MC_complete_surround([LS35(1),LS35(2);...
%       LS50(1),LS50(2);LS58(1),LS58(2)],[1,1,1],...
%       [0.8,0.8,0.8],LFs,5,0.12,100000);

semilogx(1,0,'b.',[10,100,1000,10000],vals,10^5,1,'b.');
```