Easily Scalable Algorithms for Dispersing Autonomous Robots

Michael Siebold
Murray State University,
Department of Engineering and Physics
Murray State University, Murray, KY 42071
Michael.A.Siebold@gmail.com

James Hereford
Murray State University,
Department of Engineering and Physics
Murray State University, Murray, KY 42071
HerefordJ@gmail.com

Abstract

This paper describes three new algorithms for dispersing a swarm of bots throughout a search space. We assume that the bots do not have a central coordinating agent and we want to have no (or few) inter-bot communications so that the algorithms can scale to large swarm sizes. We simulated the three new dispersion algorithms plus two other random-walk based dispersion algorithms on five different search spaces. Each of the five algorithms was tested with swarm sizes from three to fifty bots. For swarm sizes larger than ten, we found that the minimize-intensity algorithm, which is based on decaying signal strengths, worked best. For small swarm sizes, the dispersion algorithm based on the dispersion of gas particles performed best.

1. Introduction

We address the problem of how to disperse a group (swarm) of bots throughout an unknown environment. We assume that the bots originate from a common starting point (such as a doorway), do not have a central coordinating agent, and do not know where the other bots are located. Our goal is to develop an algorithm that quickly spreads the bots throughout the space.

A good dispersion algorithm is a necessary step in several possible swarm applications. For example, a typical search application is as follows: the bots are released from an entry point and then disperse throughout the search area. After dispersal, they commence their search. Another possible swarm application is area mapping. The swarm disperses and a central processor can deduce the location of passageways or obstacles based on the areas the bots can reach. Another possible application is surveillance. In this case, the bots monitor the environment and report abnormal activity as they disperse.

We developed our dispersion algorithms with three self-imposed constraints. First, we specified that there would be little or no communications among the bots. This allows our dispersion algorithm to scale easily from small numbers of bots to large numbers. Second, we wanted the dispersion algorithm to be computationally simple. Eventually we want to shrink the bots to a small size, so we assume that the processors on the bots are small with limited memory. Third, we needed the algorithm to work with small numbers of bots in the group (even as few as three), as well as with large numbers of bots.

This paper describes three new dispersion algorithms. In section 2 we give background information and then in section 3 we describe the new algorithms that we have developed. Section 4 gives the simulation results for the new algorithms and compares them against some traditional algorithms. Section 5 gives the conclusions of this paper and suggests possible future work.

2. Background information

Several researchers have investigated the problem of dispersing swarms of autonomous bots to fill a search space. In Hsiang, et al. [1], the goal was to disperse autonomous particles throughout a search space comprised of pixels. Only one bot could occupy a pixel at a time and the ultimate goal was to fill all of the pixels in a search space quickly. The actual dispersion took place primarily based on boundary- following techniques. Morlok and Gini also contributed several good dispersion techniques in their work, "Dispersing Robots in an Unknown Environment" [2]. In this paper, they presented four promising algorithms that operated in a continuous simulated environment. One of these algorithms allowed a bot in a swarm to know the location of all the bots within a certain range of the bot. This allowed the bots to quickly disperse from each other. Another of their simulated algorithms allowed each bot no knowledge of its environment or the locations of other bots. This method worked by randomly varying the direction of each bot within $\pm 5^{o}$ at every iteration of the algorithm. These algorithms are still applicable to our purpose of dispersing a swarm, though they were analyzed purely on an area-covered basis.

Another piece of research in the field of dispersing autonomous bots is the kinetic theory work of Spears, et al. [3]. Their work focused on physics-based fluid flow algorithms. Every bot is modeled as a particle which can collide with other particles and obstacles in the search space. Since actual collisions would not be practical in a bot swarm, the collisions are detected with sonar range-finders placed on each bot and the "collision" is then simulated and prevented. Momentum is conserved in these collisions, which allows for accurate prediction of how quickly the swarm will fill a given search space. These algorithms are effective and predictable since they are based on physics principles. Unfortunately, this approach requires more complex communications and calculations for each "collision" of particles. These extra calculations are not practical for implementation on small, low-power bots with limited communications capabilities.

3. Algorithm Overviews

We developed three new algorithms and tested these dispersion algorithms along with two baseline random walk algorithms. The tested algorithms include a Random Gaseous Dispersion algorithm, a Conservation of Momentum Gas Dispersion algorithm, a Minimize-Intensity Dispersion algorithm, a Collision Random-Walk algorithm, and an Iteration Random-Walk algorithm. These algorithms function as follows.

Random Gas Dispersion- As the name suggests, this algorithm derives its motion based on the motion of diffusing gas molecules. Each bot starts with an initial velocity vector. The bots proceed until they "collide" with another bot or other obstacle. Upon collision, the bot randomly chooses a new velocity vector and repeats the process indefinitely. Since the goal of these algorithms is to be implemented on an actual swarm of bots (e-pucks), the "collisions" are actually simulated when any bot gets within a predetermined distance of another bot or obstacle. We determined this collision distance by running simulations while varying this parameter. In actual hardware implementation, the collisions will be detected based on a set of eight infra-red sensors positioned around the bot. Since communications traffic is the major limiting factor in any swarm size, we decided to randomize the collision velocity vector. A flowchart of the Random Gas Dispersion algorithm is shown in Figure 1.

```
while (FOM < .9 && time < 500)

    avoid_obstacle(); % If in a collision state choose
                      % a new random velocity vector
    move();
    set_obstacle();     % Check to see if the bot is
                        % in a collision state
    calculate_metric();
    time = time + 1;
end
```

Figure 1: A flowchart of the Gas Dispersion algorithm.

The Gas Dispersion flowchart functions as follows. The while loop is executed as long as the Figure of Merit is less than .9 and less than 500 seconds have passed. Inside the loop, the bots choose a new, random velocity vector if they are in "collision" state. Next, the bots calculate their new location. They then check to see if they are in a "collision" state. Following that, they increment their timer variable and repeat the loop.

Conservation of Momentum Gas Dispersion- This algorithm is very similar to the Random Gas Dispersion method. Each bot is initialized with a randomly chosen velocity vector and proceeds until it collides with another bot or stationary obstacle. The only difference between this and the Random Gas Model is that, in this algorithm, when a collision occurs the direction of the resulting velocity vectors is determined by the conservation of momentum. The magnitudes of the velocity vectors after a collision are still chosen randomly to reduce communication. This randomization reduces the bot-to-bot communications because they will not have to exchange velocity information.

Minimize Intensity Dispersion (MID)- Our algorithm is a modification of an algorithm presented by Morlok and Gini [2]. In our algorithm, each bot emits a decaying signal that can be detected by the other bots in the swarm. An example of this emitted signal is shown in Figure 2.

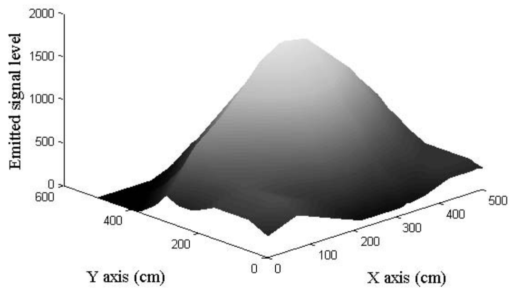

Figure 2: A snapshot of the MID algorithm showing the shape of a typical emitted signal intensity for a ten bot swarm.

The bots then use a single bot modified Particle Swarm Optimization (PSO), which was shown to be effective in [4], to seek the minimum values of the emitted signal. This requires no radio communication between bots; it also requires very few calculations since it is based on the PSO. The governing equation for this variation of the PSO is shown in equation 1.

$$\vec{V}_{i+1} = \vec{V}_i + 2 * rand * c * \left(\vec{L}_m - \vec{L}_c\right) \qquad (1)$$

In equation 1, V_i is the velocity vector at iteration i; $rand$ is a random number; c is a PSO constant equal to 2.05; L_m is the location of the maximum value of the emitted signal (also called the pbest location); and L_c is the current location of the bot. The factor of two is present, since in a standard PSO there is both an individual learning term and a population learning term. In this single bot modification, the individual term must carry the weight of the population term as well. Careful examination of this equation reveals that it only requires three multiplies and two additions per iteration. This use of the PSO brings with it a complication, however. Since the bots are always moving, the search space is continually changing. This change means that a location that was a pbest value for a bot will not stay at that value. This will cause error in the algorithm. There have been several different approaches to solve this problem of a changing search space in more traditional search techniques. [5]. The method that we decided to use was to reset the value of the pbest reading after a set number of iterations had passed. We determined this number of iterations through simulation. A flowchart of the MID algorithm is shown in Figure 3.

```
while (FOM < 0.9 && time <= 500)
  if modulus(time, 2) == 0
    reset_pbest();  % Resets the bots pbest every 2 iterations
  end
  measure_signal();
  PSO_disp_calc();     % Calculates the new velocity vector
  avoid_obstacle();    % Chooses a random velocity vector
                       % to avoid stationary obstacles
  move();
  calculate_metric(position_data)
  time = time+1;
end
```

Figure 3: A flowchart of the MID dispersion algorithm.

The flowchart for the MID functions as follows. The while loop executes if the figure of merit calculation is less than 0.9 and the amount of time elapsed is less than 500 seconds. The bot then checks to see if it has been two iterations since the personal best value has been reset. Next, the bot samples the signal emitted by the swarm. Then the bot generates a new velocity vector based on the modified single bot PSO. After this, the bot checks to see if it is in a collision state; if so, it generates a new random velocity vector which supersedes the PSO velocity vector. Finally, the bot moves to its new location and then calculates the figure of merit for the swarm. The time counter is incremented and then the loop is repeated.

Baseline Collision Random-Walk- This algorithm takes a very straightforward approach. Each bot moves in a direction until it encounters a wall or other stationary obstacle, and then randomly changes direction after that encounter. A similar version of the algorithm is found in Ryan Morlok and Maria Gini's work [2].

Baseline Iteration Random-Walk– This algorithm is similar to the Collision Random-Walk method, in that it requires the bots to move until they encounter a stationary obstacle before they change directions by more than a few degrees. The difference is that at each iteration the bot randomly modifies its velocity vector by at most ± 5°. This algorithm more closely resembles the Random-Walk algorithm outlined in Morlok and Gini's work [2].

4. Simulations

4.1. Parameter Selection

We created five different search spaces to evaluate the effectiveness of these dispersion algorithms in the presence of various environmental challenges. The search spaces are shown in Figure 4. We also used an open search space. All search spaces are 500 cm x 500 cm.

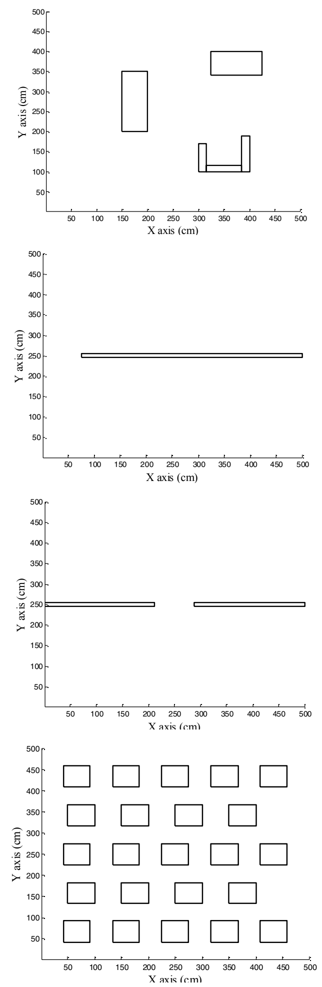

Figure 4: The layout of the simulated search spaces from top to bottom: Room, Door (edge), Door (center), Boxes.

We then developed a dispersal and coverage figure of merit, based on the standard deviations of the swarm's position relative to an ideally dispersed swarm of like size in the same search space. This figure of merit is shown in equation 2.

$$FOM = \frac{S_x+S_y}{U_x+U_y} - \left|\frac{S_x}{U_x} - \frac{S_y}{U_y}\right| \quad (2)$$

In equation 2, S_x and S_y are the standard deviation of the bot positions in the x and y directions respectively, U_x and U_y are the standard deviations of an evenly dispersed swarm in the x and y directions respectively. A simulation run will be considered a success when this figure-of- merit is greater than 0.9. The swarm always started in the bottom center of each search space (at the point 250, 0).

Before we ran the simulations, there were several parameters that needed to be determined for both the MID and the Gas Model algorithms.

In the gaseous dispersion algorithm, both the radius of the emitted signal and the reset time of the pbest reading needed to be determined. Rather than arbitrarily assigning these values, we decided to vary the radius of the decaying signal and the reset time in the clear search space to find appropriate values for these parameters. We expected the radius would be roughly half the width of the search space. On the surface, this implies that this algorithm requires prior knowledge of the search space, but in fact this radius can be used to limit the dispersion of the bots by having them stop or change their behavior if they reach some threshold value in the received signal. (For example, this would allow the bots not to spread too far in a search space without clear boundaries). Figures 5 and 6 show the results of those simulations.

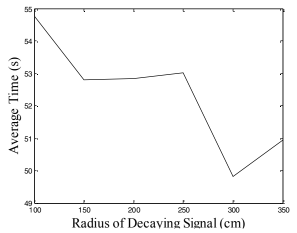

Figure 5: The minimum of average time per run (s) vs. the radius of the decaying signal was used to determine the radius for the decaying signal in the MID algorithm.

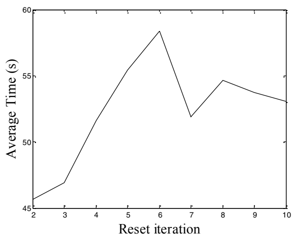

Figure 6: The minimum of the average time per run vs. the number of iterations before the bot resets its pbest was used to determine an appropriate reset value in the MID algorithm.

These simulations showed that a reasonable value for the number of iterations before the pbest is reset is 2 and the best radius for the decaying signal is 300 cm.

For the Gaseous Diffusion model, we needed to determine what the initial energy (velocity) should be in the swarm and what the simulated collision radius should be set to. To accomplish this, we ran a set of simulations varying the initial velocity and a set of simulations varying the collision range. These simulations led us to choose a collision radius of seven cm and an initial velocity of 12.8 cm/s (the maximum velocity of the e-puck bots).

As a further test of the reduction in communications achieved by randomizing the collision velocity vector, we ran an entire set of simulations using the conservation of momentum method for generating the new velocity vectors. We placed a counter variable that kept track of the number of communications packets sent by all the bots. We assumed that each communication would require two communications packets since it would be transmitting both an x and y component of velocity. The results of this test are shown in Table 1.

Table 1: The number of communications packets required for a conservation of momentum version of the Gaseous Dispersion algorithm.

Num Bots	Comm Packets
3	94.80
5	275.07
10	896.77
15	1829.65
20	2809.68
25	4045.09
30	5422.61
35	6643.41
40	8314.11
45	10295.81
50	11498.72

These results show that the number of communications packets increases as the size of the swarm increases roughly in a linear fashion. We linearly curve fit the data in Table 1 and the result is equation 3. The R^2 value of this linear fit is 0.979.

$$Comm\_Packets = 246.9 * Num\_Bots - 1501 \quad (3)$$

4.2. Simulation Results

The simulations and environments were written in Matlab. Each algorithm was simulated thirty times in each of the search spaces. Each search space was simulated with three bots, five bots, and up to fifty bots in increments of five. The results of these simulations are shown in Table 2.

Table 2: The results of the timed simulations for each algorithm in each search space. These times reported (in seconds) are an average of 30 runs at a 90% confidence interval.

MID with Signal Radius = 300					
Bots	**Clear**	**Room**	**Door (edge)**	**Door (center)**	**Boxes**
3	155.67 ± 51.49	129.70 ± 48.46	304.83 ± 63.85	157.70 ± 50.08	338.73 ± 53.57
5	60.30 ± 30.37	65.27 ± 23.01	88.30 ± 21.06	57.93 ± 17.77	226.33 ± 36.25
10	32.43 ± 2.52	43.53 ± 11.92	265.13 ± 31.07	100.10 ± 41.64	273.10 ± 31.24
15	28.17 ± 1.01	40.40 ± 12.01	183.77 ± 24.69	70.63 ± 36.52	232.33 ± 29.90
20	30.80 ± 1.08	33.13 ± 4.22	225.70 ± 21.21	49.57 ± 11.12	274.50 ± 15.82
25	31.07 ± 0.74	34.03 ± 5.03	253.67 ± 24.90	77.80 ± 22.77	252.70 ± 14.01
30	30.77 ± 0.65	35.80 ± 5.82	220.37 ± 20.28	47.27 ± 11.51	236.60 ± 15.41
35	31.20 ± 0.83	30.53 ± 1.88	228.40 ± 11.97	41.77 ± 8.17	245.80 ± 14.50
40	31.20 ± 0.84	34.57 ± 4.22	256.50 ± 16.25	52.23 ± 11.34	232.67 ± 13.34
45	33.13 ± 0.72	32.13 ± 1.17	293.60 ± 20.23	53.20 ± 10.19	249.50 ± 14.42
50	29.13 ± 0.60	29.13 ± 0.62	210.60 ± 13.69	38.33 ± 4.11	189.97 ± 13.23
Random Gaseous Dispersion Model					
Bots	**Clear**	**Room**	**Door (edge)**	**Door (center)**	**Boxes**
3	63.70 ± 13.36	62.52 ± 13.76	84.47 ± 15.38	89.24 ± 15.94	87.19 ± 16.42
5	33.30 ± 3.56	50.95 ± 9.18	59.78 ± 10.41	60.45 ± 13.44	86.22 ± 15.17
10	46.52 ± 5.04	48.63 ± 4.12	109.93 ± 16.81	145.77 ± 42.01	100.00 ± 11.42
15	40.67 ± 3.29	47.53 ± 4.11	98.51 ± 11.05	92.74 ± 21.88	87.39 ± 7.46
20	45.11 ± 2.00	55.38 ± 4.46	131.74 ± 24.56	91.53 ± 14.53	107.70 ± 8.66
25	52.96 ± 3.61	53.89 ± 3.68	134.04 ± 22.44	118.85 ± 18.38	117.76 ± 13.74
30	52.93 ± 5.40	58.57 ± 4.94	124.43 ± 13.02	111.60 ± 14.07	109.97 ± 9.86
35	54.82 ± 2.86	59.12 ± 3.15	145.77 ± 16.75	120.82 ± 24.07	115.24 ± 8.71
40	52.19 ± 3.05	65.90 ± 3.15	140.15 ± 13.23	128.38 ± 17.29	125.56 ± 10.63
45	56.80 ± 4.02	66.84 ± 3.61	177.62 ± 17.29	161.97 ± 22.38	135.57 ± 9.50
50	48.99 ± 2.94	59.65 ± 1.76	138.38 ± 11.87	98.67 ± 9.45	107.67 ± 6.75
Conservation of Momentum Gaseous Dispersion Model					
Bots	**Clear**	**Room**	**Door (edge)**	**Door (center)**	**Boxes**
3	37.43 ± 7.84	49.18 ± 9.58	84.38 ± 14.47	78.57 ± 20.54	94.92 ± 20.68
5	30.28 ± 4.42	38.84 ± 10.01	50.89 ± 9.25	44.10 ± 8.31	63.05 ± 8.95
10	32.63 ± 1.64	39.04 ± 3.77	82.20 ± 10.46	102.12 ± 22.72	84.42 ± 10.71
15	29.68 ± 1.29	38.36 ± 5.03	93.33 ± 12.80	66.80 ± 12.74	86.50 ± 10.11
20	33.45 ± 1.26	41.27 ± 3.30	106.45 ± 15.24	97.41 ± 21.54	91.22 ± 7.88
25	36.00 ± 1.29	42.45 ± 3.69	118.48 ± 17.81	100.18 ± 21.42	112.22 ± 11.33
30	35.50 ± 1.54	48.36 ± 4.86	112.07 ± 11.76	91.78 ± 17.73	108.88 ± 10.11
35	36.37 ± 1.68	48.83 ± 4.05	128.52 ± 19.79	108.29 ± 19.80	112.74 ± 8.94
40	36.77 ± 1.82	44.68 ± 2.94	137.45 ± 15.83	98.69 ± 18.14	117.72 ± 9.53
45	37.32 ± 1.76	49.62 ± 3.58	181.38 ± 18.93	145.72 ± 18.87	129.64 ± 12.07
50	33.88 ± 0.79	43.90 ± 2.84	121.89 ± 13.07	74.67 ± 14.30	107.35 ± 9.22
Collision Based Random Walk					
Bots	**Clear**	**Room**	**Door (edge)**	**Door (center)**	**Boxes**
3	58.63 ± 8.75	60.75 ± 12.29	90.10 ± 17.82	95.70 ± 25.95	104.26 ± 18.15
5	31.52 ± 3.03	41.70 ± 8.00	62.81 ± 10.09	57.82 ± 11.67	85.09 ± 16.11
10	40.29 ± 2.21	49.33 ± 6.75	92.97 ± 13.11	97.17 ± 17.19	81.58 ± 8.47
15	36.85 ± 2.12	38.92 ± 3.75	75.13 ± 10.89	67.30 ± 10.19	81.98 ± 9.48
20	36.87 ± 1.33	40.07 ± 2.58	76.67 ± 8.99	93.60 ± 15.19	82.96 ± 8.43
25	41.49 ± 1.62	45.11 ± 3.47	85.75 ± 11.76	112.11 ± 17.63	96.16 ± 8.93
30	38.17 ± 1.04	41.02 ± 2.06	99.91 ± 14.19	116.73 ± 19.14	93.97 ± 7.41
35	39.31 ± 1.07	45.48 ± 2.85	97.24 ± 10.16	121.72 ± 21.35	95.50 ± 8.05
40	38.45 ± 0.97	45.74 ± 2.33	108.85 ± 10.08	117.24 ± 15.80	97.14 ± 6.82
45	43.17 ± 2.15	48.12 ± 2.97	117.79 ± 14.69	150.92 ± 23.59	105.94 ± 7.18
50	36.84 ± 1.33	40.18 ± 1.63	104.25 ± 12.63	97.32 ± 17.11	85.17 ± 4.84

	Iteration Based Random Walk				
Bots	**Clear**	**Room**	**Door (edge)**	**Door (center)**	**Boxes**
3	53.60 ± 10.99	47.56 ± 8.65	110.66 ± 32.52	83.89 ± 18.38	106.17 ± 22.73
5	45.56 ± 9.60	40.93 ± 6.62	54.55 ± 10.84	56.68 ± 11.91	66.85 ± 10.87
10	38.51 ± 1.80	43.79 ± 3.94	109.81 ± 22.31	87.51 ± 18.13	85.18 ± 9.41
15	34.77 ± 1.58	37.05 ± 2.68	70.64 ± 7.84	83.69 ± 19.43	70.67 ± 6.32
20	36.80 ± 1.44	39.34 ± 2.28	88.69 ± 13.15	100.69 ± 17.24	91.90 ± 6.80
25	40.29 ± 2.72	43.81 ± 3.16	96.76 ± 13.54	111.41 ± 25.50	100.74 ± 8.90
30	38.64 ± 1.18	43.08 ± 2.84	99.75 ± 13.92	98.12 ± 13.48	93.90 ± 7.87
35	40.05 ± 1.65	43.35 ± 2.79	99.50 ± 15.22	110.53 ± 21.38	89.80 ± 6.10
40	38.87 ± 1.24	44.28 ± 2.60	89.83 ± 8.72	114.11 ± 15.90	100.23 ± 7.46
45	41.94 ± 1.75	47.47 ± 2.13	135.37 ± 17.85	124.73 ± 17.56	106.72 ± 5.96
50	36.39 ± 0.86	41.03 ± 1.30	86.72 ± 8.72	100.75 ± 17.61	88.07 ± 5.68

The reported numbers are average time values (and the 90% confidence interval). A run consisted of the time it took a swarm to achieve a value of 0.9 in our FOM, or a maximum of 500 seconds.

These simulations clearly show that the MID algorithm is superior to the other tested algorithms when the number of bots exceeds ten. The exceptions to this are the search spaces where there is a wall across the search space and a door near the edge of the wall and the space filled with boxes. Our understanding of these aberrations is as follows: in the MID algorithm, each bot is essentially repelled from all of the other bots by the emitted signal as a bot enters the door. At the edge of the search space the emitted signal then repels the other bots that would pass through the door until that bot has moved on. In the search space filled with boxes the bots are in collision mode so much that the emitted signal does not effectively spread the swarm. This would explain the longer search times for these spaces. For small swarm sizes and for more complex search spaces, the Conservation of Momentum Gas Dispersion algorithm is superior. It also performs quite well in less complicated search spaces with larger swarm sizes. The surprise in these results is the performance of the two random-walk based algorithms. These very low calculation algorithms performed well. They performed consistently better than the Random Gas Dispersion algorithm. We believe that the performance of these two algorithms may be exaggerated since these algorithms were performed without consideration of bot-to-bot collisions. The Conservation of Momentum Gas Dispersion method out-performed all algorithms for smaller swarm sizes of three to five bots.

5. Conclusions

We have developed three new low communication dispersion algorithms for use on small low powered autonomous bots. The Minimize Intensity Dispersion Algorithm performed best for large swarm sizes in less complex search spaces, while the Conservation of Momentum Gas Model Dispersion worked better for smaller swam sizes and in more complex search spaces.

Looking forward to hardware implementation, we have obtained several e-puck bots to conduct a set of proof of principle experiments. We are in the process of implementing both the Random Gas and the Minimize Intensity Dispersion algorithms in these e-pucks.

References

[1] T.-R. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell. Algorithms for Rapidly Dispersing Robot Swarms in Unknown Environments. *Proc. of the 5th Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2002, pp. 77-94.

[2] R. Morlok and M. Gini, “Dispersing robots in an unknown environment”, *Proc. Int'l Symp. on Distributed Autonomous Robotic Systems*, 2004, pp. 241–249.

[3] D. Spears, W. Kerr, and W. Spears “Physics-Based Robot Swarms For Coverage Problems”, *The International Journal of Intelligent Control and Systems*, September 2006, pp. 124-140

[4] James M. Hereford, Michael Siebold, Shannon Nichols, “Using the Particle Swarm Optimization algorithm for robotic search applications”, 2007 Swarm Intelligence Symposium, Honolulu, HI, pp. 53 – 59, April 2007.

[5] T. Blackwell and B. Jürgen, “Mutiswarms, Exclusion, and Anti-Convergence in Dynamic Environments”, *IEEE Transactions on Evolutionary Computation,* IEEE Computational Intelligence Society, August 2006, pp. 459-472.