Robust and self-repairing formation control for swarms of mobile agents

Jimming Cheng
Harvard University
jvcheng@post.harvard.edu

Winston Cheng
Harvard University
cheng2@post.harvard.edu

Radhika Nagpal
Harvard University
rad@eecs.harvard.edu

Abstract

We describe a decentralized algorithm for coordinating a swarm of identically-programmed mobile agents to spatially self-aggregate into arbitrary shapes using only local interactions. Our approach, called SHAPEBUGS, generates a consensus coordinate system by agents continually performing local trilaterations, and achieves shape formation by simultaneously allowing agents to disperse within the defined 2D shape using a gas expansion model. This approach has several novel features (1) agents can easily aggregate into arbitrary user-specified shapes, using a formation process that is independent of the number of agents (2) the system automatically adapts to influx and death of agents, as well as accidental displacement. We show that the consensus coordinate system is robust and provides reasonable accuracy in the face of significant sensor and movement error.

Introduction

Biology has shown that complex global behaviors can arise from simple interactions between large numbers of relatively unintelligent agents [1]. Swarm approaches to robotics, involving large numbers of simple robots rather than a small number of sophisticated robots, has many advantages with respect to robustness and efficiency. Such systems can typically absorb many types of failures and unplanned behavior at the individual agent level, without sacrificing task completion. At the same time, one can exploit parallelism and spatially distributed sensing and action. Emerging technologies are making it possible to cheaply manufacture small robots with sensors, actuators and computation [2,3]. This makes swarm intelligence an attractive solution for many problem domains.

In this paper, we focus on the problem of organizing mobile agents into arbitrary self-sustaining 2D formations. Keeping strategic formations of mobile agents is important for many tasks, especially when individual agents have limited abilities or the task requires collective action. For example, agents may aggregate for coordinated search and rescue, collectively moving large objects, exploring and mapping unknown terrain, or maintaining formations for defense or herding. However, current work on multi-robot formations achieves only simple shapes or uses complex negotiations unsuitable for large groups of robots [4].

We propose a decentralized approach to multi-agent formation, that can not only achieve arbitrary shapes but is also robust to varying numbers of agents, agent influx and death, and practical hardware limitations like sensor and movement error. Our approach, SHAPEBUGS, achieves shape formation by generating a consensus coordinate system through agents performing local trilaterations, while simultaneously allowing agents to disperse themselves within the defined 2D shape. Briefly, the system works as follows: agents initially start in a "wandering" state, with no information about their environment (including their own world coordinates), but with an internal knowledge of the desired shape to be formed. A small number of agents are temporarily seeded with initial positions. Agents are equipped with imperfect proximity sensors and wireless communication with only nearby neighbors. As agents move, they continually perform local trilaterations to learn and maintain a common coordinate system. At the same time, agents influence each other's movements according to a gas expansion model inspired by pheromone robots and flocking rules [2,5]; this causes them to disperse within the shape and fill it efficiently.

This approach has several salient features. Not only can agents easily aggregate into arbitrary user-specified shapes, but also the shape formed *is independent of the number of agents*. The gas expansion model causes agents to disperse evenly within the shape, and varying the number of agents simply changes the equilibrium density. This model is also capable of automatic self-repair; the system can quickly recover from most patterns of agent death and can receive an influx of new agents at any location without blocking problems. We show, through simulation experiments, that the consensus coordinate system is robust and remains accurate in the face of practical limitations such as sensor and movement error.

The rest of the paper is as follows: we present related work, followed by a description of our agent model. We then describe the SHAPEBUGS algorithm in detail. Finally we present simulation experiments that investigate time efficiency, robustness to agent error, and self-repair.

Related Work

Algorithms for spatial organization of agents/robots via local interactions have steadily increased in sophistication. Several approaches have been proposed that only work for a small set of simple shapes. Unsal and Bay [4] developed a model where some agents become beacons, instructing others to remain at or within a certain distance, enabling the construction of rings and circles. Mamei et al. [6] have a similar approach, but use message hop count instead of a proximity sensor and can form crude polygons.

Multi-agent algorithms for forming arbitrary shapes have been designed for other agent models. For example,

Kondacs [7] presents an approach to shape formation on biologically-inspired agents that grow (self-replicate) and die. They use global-to-local compilation to automatically generate an agent program for a given shape; this system can be programmed for a large class of shapes and can self-repair. Stoy and Nagpal [8] present a related approach to 3D self-assembly on a simulated self-reconfigurable modular robot, where individual modules are mobile but must remain connected. While this system generates a wide variety of shapes, it cannot create solid shapes because agents may block each other and create internal holes that no wandering agents can reach. To avoid this, the system focuses on scaffolds and porous shapes only. Solid structures also make self-repair difficult because agent death deep within the shape is hard to get to.

Gordan et al. [9] address arbitrary shape formation with mobile agents by setting up a shared coordinate system and then distributing agents. However, their procedure moves in stages and involves significant centralization, so it is hard to adapt to failures during the formation process.

Our system also achieves arbitrary shapes by forming a decentralized coordinate system. However, instead of agents taking fixed positions in the shape, we use an adapted form of a dispersion algorithm proposed by Payton et al. [2]. We show that local dispersion rules can be effectively combined with shape formation, allowing the system to self-adjust to agent density, avoid blocking, and self-repair by automatically collapsing internal holes. Spears et al [10] use similar dispersion rules based on natural physics to achieve surveillance and perimeter defense, but they do not attempt organized shapes.

Mobile Agent Model

We assume a particular agent model that is motivated by capabilities of real autonomous robots (Fig. 1). We assume that agents move in 2D continuous space, all agents execute the same program, and agents interact only with other nearby agents by measuring distance and exchanging messages. We also assume that agents have a perfect compass, but that both distance measurements and movement have error. Currently, we simplify the handling of agent trajectories in simulations, by assuming that the world is finite and an agent that wanders off one side will reappear on the other side (wrap around space). For future work, agents may infer orientation using multiple trilaterations instead of depending on a compass.

SHAPEBUGS Algorithm

In the SHAPEBUGS algorithm, each agent has a map of the shape to be constructed that is later overlaid on the agent's learned coordinate system. Initially, agents are scattered randomly in the world in a lost state, oblivious of their own positions. When turned on, each agent begins to

Proximity Sensor	Gives estimated real distance of each neighbor within range with uniformly distributed measurement error.
Compass	Gives perfect directional orientation
Wireless Connection	Allows agent to query the perceived coordinates of neighbors within range.
Locomotion	Moves agent in discrete predetermined step sizes on the real world. Movement has error, so actual distance traveled may vary.
Shape Map	Map of the destination shape to overlay on perceived coordinate system.
Program	Trilateration and movement processes.

Figure 1: Agent Model

execute its program using only data from its proximity sensor and its wireless link with nearby neighbors.

The agent program can be broken down into two processes that run continuously and concurrently. In the first process, an agent adjusts its perceived coordinate system so that it coincides with other agents' perceived coordinate systems. This is achieved by trilateration using proximity sensor data. The second process controls agent movement. If an agent believes it is inside the shape, then it behaves like a gas particle with the shape as an closed container. Otherwise, the agent wanders randomly.

Process 1: Local Trilateration Process

The trilateration process allows an agent to find its perceived position (x_p, y_p) on the consensus coordinate system for the first time and subsequently adjust it. Trilateration can only occur if there are at least three neighbors that are not themselves lost. An agent uses its proximity sensor to estimate distance $d^{PS}{}_i$ to each neighbor NB_i, and also queries NB_i's own perceived coordinates (x_i, y_i) with its wireless connection. The best fit for (x_p, y_p) minimizes over all neighbors the difference between $d^{PS}{}_i$ and the calculated distance from (x_p, y_p) to neighbor's reported coordinates (x_i, y_i):

$$\underset{(x_p, y_p)}{\arg\min} \sum_{NB_i} \left| \sqrt{(x_i - x_p)^2 + (y_i - y_p)^2} - d^{PS}{}_i \right|$$

With this information, the agent can calculate its own position using a standard gradient descent algorithm:

1. Start with some initial position ($x^{(i)}$, $y^{(i)}$), i = 0.
2. Find gradient ∇ at current position ($x^{(i)}$, $y^{(i)}$).
3. Move away from the gradient with a step size β
   $(x^{(i+1)}, y^{(i+1)}) = (x^{(i)}, y^{(i)}) - \beta\nabla(x^{(i)}, y^{(i)})$
4. Repeat 2 and 3 until a local minimum is reached.

The success of the gradient descent algorithm depends largely on the initial starting position. If an agent has calculated its position recently, (x_p, y_p) is used as the starting point. If the agent is *lost*, then (x_p, y_p) is undefined and a good starting candidate can be found using the following procedure. First, pick three random neighbors NB_p, NB_q and NB_r, and draw circles P, Q and R of radius d_p, d_q and d_r around their centers. Each pair of circles will most likely intersect at two points—one close to the correct position and the other extraneous. The approximately

correct circle intersection points among all neighbors are clustered tightly while extraneous points are scattered. Any of the clustered points would be a good starting candidate. Let PQ_A and PQ_B be the two intersection points between circles P and Q. To decide which one is in the cluster (non-extraneous), we draw line L_1 through PQ_A and PQ_B and L_2 through PR_A and PR_B. Because one of the points on L_1 is clustered with one of the points on L_2, the intersection of L_1 and L_2 will be closer to either PQ_A or PQ_B, whichever is the better approximation. This serves as the starting point.

Sources of Error: Positioning is approximate and may have error for several reasons. First, even with perfect sensors, the gradient descent algorithm may settle at local minima. Second, distances measured by proximity sensors may have error. Third, coordinates reported by neighbors may be inaccurate. Thus, it is important to take the mean of several trilaterations instead of relying on a single trilateration for each coordinate adjustment decision. In addition, imperfect control can limit an agent's ability to track its actual movement; an agent that thinks it moved distance d in some direction may actually move $d \pm \Delta$. As a result, perceived coordinates accumulate error over time even if no further trilaterations are performed. Thus, it is important to readjust coordinates at regular intervals. Sensor and movement tracking error are common in real hardware, so it is important to account for their effects.

We address these issues by implementing averaged and repeated trilateration as follows. Each agent calculates new trilaterated coordinates at every time step. A window of the last w trilaterations is always kept in memory. Every r steps, the agent updates its perceived coordinates by averaging over the last w trilaterations. The effects of averaged and repeated trilaterations can be studied by varying w (number of previous trilaterations considered) and r (interval between coordinate adjustments).

Process 2: Movement Rules

Agents follow different movement patterns depending on whether they think they are inside or outside the shape. If an agent is lost, it assumes it is outside the shape. Agents that think they are outside the shape hope to find their way into the shape. Our agents simply walk continuously in a random semi-straight path. In our finite, wrapping world, this allows agents to find the shape with high probability[1].

Agents who think they are inside the shape have a more complex objective, since they are part of the intelligent swarm that comprises it. First, these agents should not take any steps that will put them outside of the shape. This helps keep the shape intact once formed. Second, the swarm should be capable of executing desired tasks such as self-repair and graceful absorption of new agents.

The SHAPEBUGS algorithm achieves these goals by modeling agents in the shape as gas particles in a closed container. Agents react to different densities of neighbors around them, moving away from areas of high density

[1] In the future, a new class of recruiting agents can scout for lost agents, and non-lost agents can be programmed to navigate towards the shape.

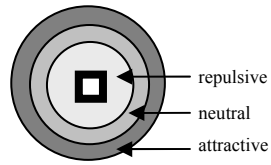


Figure 2: Pheromone robot's influence zones

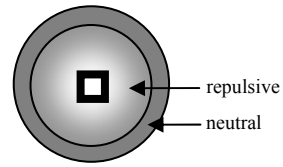


Figure 3: SHAPEBUG's influence zones

towards low density. Over time, they settle into an equilibrium of constant pressure throughout the shape.

When agents die, surrounding agents quickly flood the resulting area of low pressure until equilibrium is restored. Thus, the swarm can respond to any loss as long as there are enough agents left to generate a sensible equilibrium pressure. If new agents are injected into the swarm at any point, the resulting area of high pressure will quickly dissipate. Therefore, many agents entering the swarm at a single location will not be a barrier to subsequent agents.

This behavior is inspired by Payton et al.'s Gas Expansion Model [2], but is also similar in nature to the flocking rules proposed by Reynolds [5]. While the Payton model (Fig. 2) strives for maximal dispersal to an optimum agent density, our movement model has two goals: 1) Equalize pressure at any agent density. 2) Superimpose the notion of a container.

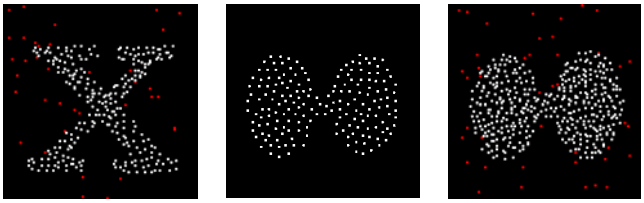
The first goal is achieved by giving each agent a varying repulsive force that has a maximum value adjacent to the agent and decays at a constant rate until it reaches a neutral zone (Fig. 3).. The agent's movement vector, calculated at every time step, is the sum of the vectors away from repulsing neighbors, weighted inversely by distance. This allows agents to disperse evenly at any density. Let R be the repulsive radius and d^{PS}_i be the distance measured by the proximity sensor. Given the agent's own perceived position $\vec{v} = (x_p, y_p)$, and the perceived position $\vec{v}_i$ of each neighbor i, the movement vector $\vec{m}$ is given by:

$$\vec{m} = \sum_i \frac{\vec{v}_i - \vec{v}}{d_i^{PS}} (R - d_i^{PS})$$

The second goal is achieved by superimposing a container on the agents' coordinate systems. Agent movement is restricted to only steps that keep them inside the container. If the calculated movement vector would take an agent outside the shape, it is discarded in exchange for a either staying still or making a random movement within the shape with some small probability. This keeps agents on borders from getting stuck.

Evaluation and Results

We show through simulation that SHAPEBUGS can form arbitrary shapes while automatically compensating for various sources of error and agent influx and death. We implemented SHAPEBUGS in Java, using the Swarm Development Group's multi-agent simulator to run tests [11]. Shape Maps are represented as 1-bit bitmap images. When a Shape Map is overlaid on the 2D continuous



a) "X", 300 agents b) Barbell, 180 agents c) Barbell, 450 agents

Figure 4: a) Formation of the letter X by 300 agents. b) and c) demonstrate shape formation under varying agent densities.

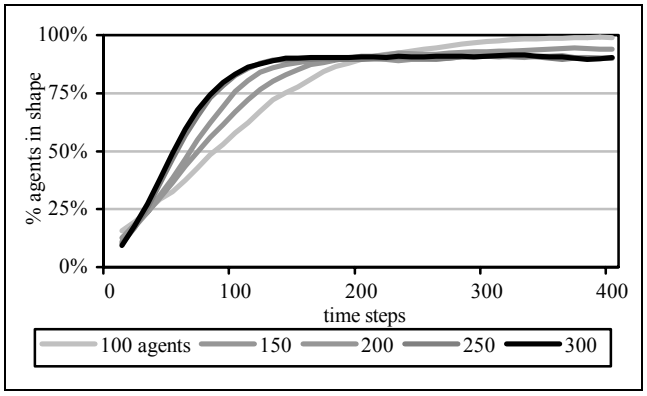


Figure 5: Rate of shape formation for different agent densities.

agent world, each pixel corresponds to a unit square area. In all cases, exactly 12 agents clustered in a common 5x5 area are seeded with compatible initial positions to trigger the initial round of trilaterations. Proximity sensors have a range of 5.0 units. Agents move in discrete steps of 2.0 units, but error can cause actual step size to vary.

Figure 4 shows several examples of formations by mobile agents, and also shows that the same shape can be maintained at different densities with no modification of agent behavior. The gas expansion rules cause agents to always disperse evenly at any density.

The stabilized shapes took about 300 time steps to complete, depending on agent density. Figure 5 shows the percentage over time of agents who have coordinates and are in the shape while forming a 50x50 square in a 80x80 world. Rate of shape formation increases as the number of agents increases from 100-300. With higher agent density, there are more interactions, so coordinate systems propagate faster and time to stabilization is reduced.

Coordinate System Accuracy and Robustness

We measure quantitatively the overall level of agreement among all agents on a similar coordinate system. To do this, we calculate the *variance* of the consensus coordinate system, by first computing the mean global coordinate system over all agents (i.e. average location of the origin) and then computing the variance of the distance between each agent's local coordinate system and the mean coordinate system. Lower variance signals more agreement between different agents' coordinate systems.

In addition, we test how the distributed model is affected by practical hardware limitations, in particular *proximity sensor errors* and *movement error*. Proximity

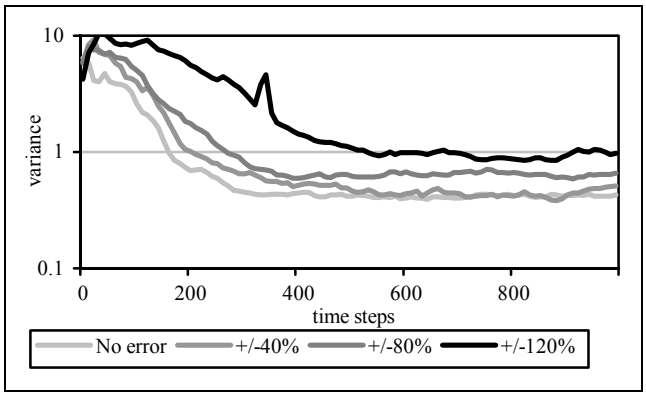


Figure 6: Coordinate variances degrade gracefully with increasing proximity sensor errors (note: log y axis).

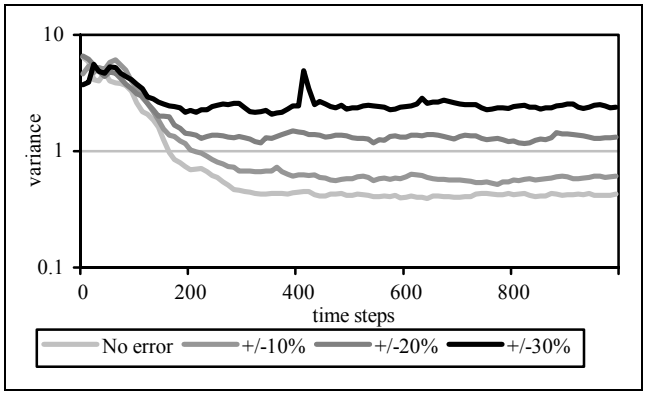


Figure 7: Effect of increasing movement error (log y axis).

sensors are inherently imperfect, and external effects such as atmospheric disturbances further impair their accuracy. We simulate sensor error by adding to each sensed distance d^{PS} a uniformly distributed random error e^{PS}. With movement error, each step an agent takes may put it slightly farther or nearer than it thinks. Thus, an agent may drift from its perceived coordinates over time. We simulate movement error by adding a uniform random error e^{MV} to the true distance moved d^{MV}.

Figures 6 and 7 show coordinate system variances for different levels of sensor and movement errors. In these experiments, agents formed a 50x50 square centered in a 80x80 world. For each agent, $r = w = 10$. In general, setting $r = w$ was a sensible choice. Trials showed that setting $r > w$ performed worse. If the window size does not at least span the entire interval between readjustments, then updates will not be utilizing all available information. In particular, $r - w$ out of every r trilaterations will be wasted. Setting $r < w$ gave empirically similar results to $r = w$. Proximity sensor error is expressed as a percentage of the sensor range. Since the sensor range is 5 units, a sensor error of ±40% implies that e^{PS} is a random variable on the interval [-2,2]. Similarly, movement error is a percentage of the step size.

For sensor and movement error, coordinate system variance settles to a stable value after about 400 time steps. Agents achieved variances under 0.39 units^2 for

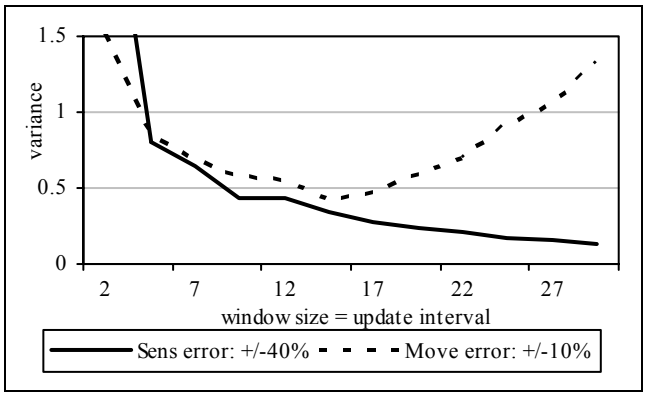


Figure 8: Variance after the swarm has stabilized. Isolated sensor error of ±40% vs. isolated movement error of ±10%.

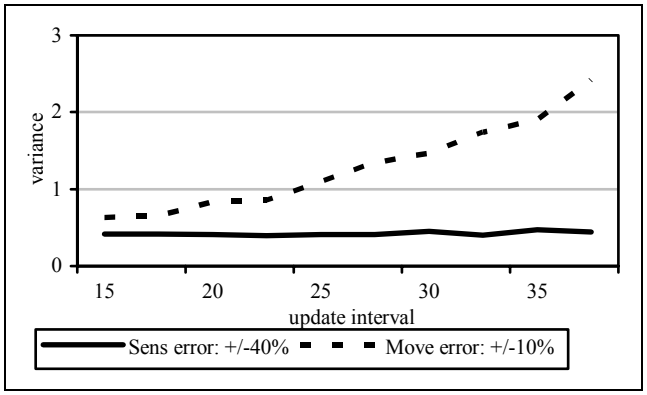


Figure 9: Large update intervals degrade performance on trials with movement error but do not affect trials with sensor error.

proximity errors up to ±20%, and under 0.57 units^2 for movement errors of ±10%. Performance degrades gracefully as the magnitude of error increases for both sensor and movement error. However, it is also clear that SHAPEBUGS is more tolerant of sensor error than movement error. For example, experiments with a sensor error of ±80% can still reach a variance of less than 0.65 units^2, but a movement error of only ±20% results in a variance of 1.30 units^2.

This suggests a fundamental difference between the way sensor error and movement error affect coordinate system agreement. Indeed, figure 8 shows that increasing w and r causes the stabilized variance to continually decrease when there is only sensor error. However, with movement error, stabilized variance reaches a minimum at $w = r = 15$ and then increases again. To understand why this pattern occurs, we note the following: First, both sensor and movement error are sources of random noise with expected values of zero at each time step. Thus, increasing w and averaging across more time steps has the effect of filtering out this noise. This explains the downward trend for sensor error and the initial downward trend for movement error. However, movement error has the added characteristic that it is cumulative. While sensor error is independent across time steps, movement error can add up and cause coordinate drift over time. Thus, as the interval r between updates increases, the drift worsens, as there are more steps for it to accumulate. Figure 9 demonstrates this by holding constant $w = 5$ and varying r. Movement error worsens with increasing r while sensor error is unaffected. This may explain the delayed upward trend in variance for the movement error experiments. With movement error, the goal is to set w and r to strike a balance between minimizing noise and minimizing drift.

Adaptation and Self-Repair

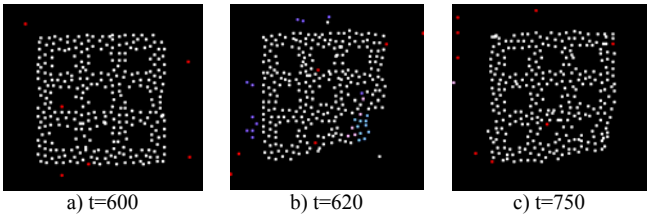
We have shown that SHAPEBUGS can form various shapes and adapt to sensor/movement error. Here we describe experiments aimed at testing the ability to recover from large scale errors such as 1) accidental misplacement of large numbers of agents and 2) regional death or influx of agents. We show that the coordinate system can restabilize and that agents adjust to influx and death without any explicit detection or monitoring for failures.

A challenge at the macro level is for an entire group of misinformed agents to stabilize in relation to the aggregate entity. For example, if the terrain under an entire group of agents shifts, those agents will be fragmented from the aggregate entity but will agree on a common faulty coordinate system. The aggregate entity should be able to overcome these regional failures.

To test this case, we first allowed agents to stabilize into the aggregate shape. Then, we selected a large region of agents and uniformly displaced their coordinate systems. Figure 10 shows experiments on a swarm that has formed into a grid pattern. Agents in the lower right corner had their perceived coordinates shifted 15 units down and to the right, and the swarm was allowed to reconverge. In figure 10b the displaced agents start to form another grid at the shifted coordinates, but as they interact with their neighbors from the original grid, they correct the error, and revert to the original shape (Fig. 10c).

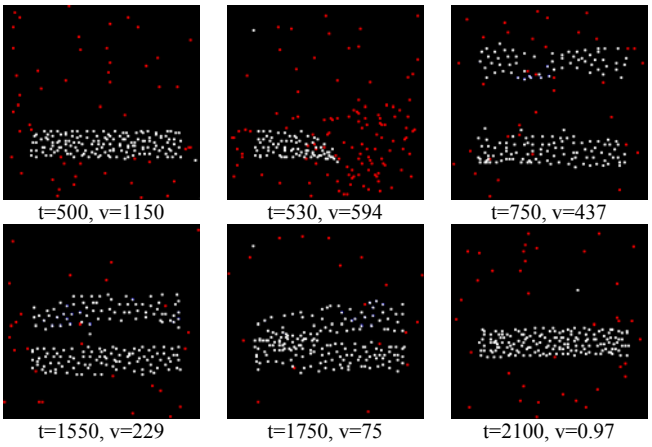
Figure 11 illustrates a more extreme example where the shifted agents become completely detached from the original shape. Here, agents in the right half of a 90x15 rectangular bar experience a 50 unit downward (y-axis) shift in their perceived coordinates. This causes the variance of the agents' perceptions of their coordinate system to rise sharply as the displaced agents start forming a new complete shape above the original (t=530). However, as agents randomly break free from one shape and run across to mingle with the other, the two distinct coordinate systems slowly drift towards each other. Variance decreases slowly at first, but drops quickly around step 1750 when the two shapes finally meet. Here, the agents are able to interact much more freely, so reconvergence accelerates during these final steps.

Figure 12 demonstrates the repair feature. Spaces opened up by agent death are quickly filled by neighboring agents. We removed the agents outlined in the square in figure 12a, and the gap was quickly repaired. The efficiency of repair varies with the type of shape. For instance, repair is slow if the shape has a bottleneck. When the right half of the barbell in Figure 12 dies, repair



a) t=600 b) t=620 c) t=750

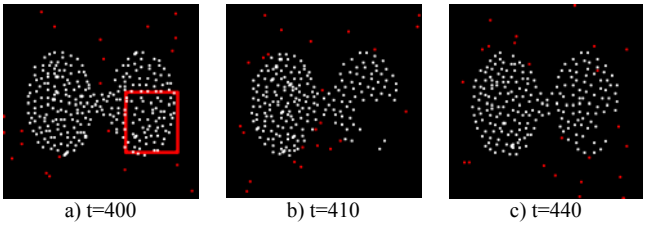
Figure 10: Recovery from distorting a region of the shape.



t=500, v=1150 t=530, v=594 t=750, v=437

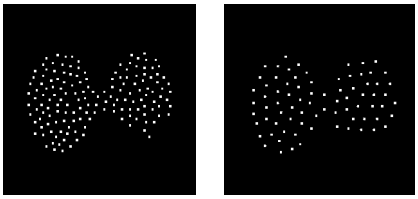
t=1550, v=229 t=1750, v=75 t=2100, v=0.97

Figure 11: Reconvergence of a horizontal bar (v = variance)



a) t=400 b) t=410 c) t=440

Figure 12: Repairing a region after agent deaths



neighborhood=5 neighborhood=8

Figure 13: Lattice formation for different neighborhood sizes

takes about 250 steps. By contrast, repair only takes 80 steps for similar death in a square shape with the same area. Also, formation and repair will not finish if agent density is so low that agents can space themselves evenly to the limit of their communication range. In this range, the gas expansion laws no longer push the agents. The agents settle into a static, lattice-like formation (Fig. 13).

Conclusion

This work shows an effective strategy for construction and preservation of a complex aggregate entity using large numbers of simple decentralized agents. By composing two processes, 1) trilateration and 2) gas expansion movement, agents can self-organize into arbitrary user-specified shapes. The resulting structure can self-repair and restabilize in cases of agent death and displacement, and can overcome large degrees of sensor and movement error. In addition, the ability to operate in spite of common hardware limitations make SHAPEBUGS a model that could be feasibly implemented with real robots.

SHAPEBUGS can be further improved by addressing certain aspects such as imperfect orientation control and active recruiting of agents instead of random wandering. In addition, our goal is to extend SHAPEBUGS to allow the swarm to move in formation. We believe that the gas expansion model will allow us to achieve goal-directed movement in a way that allows the system to move around obstacles yet retain (repair) swarm shape.

References

[1] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, E. Bonabeau. *Self-organization in Biological Systems*, Princeton University Press, 2002.

[2] D. Payton, M. Daily, R. Estkowski, M. Howard, C. Lee. Pheromone Robots. *Autonomous Robots,* 11, 3: 319-324, 2001.

[3] F. Mondada, A. Guignard, M. Bonani, D. Floreano, M. Lauria, SWARM-BOT: From Concept to Implementation, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[4] C. Unsal, J. Bay. Spatial Self-organization in Large Populations of Mobile Robots. *International Symposium on Intelligent Control,* August, 1994.

[5] C. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model, *SIGGRAPH*, 1987.

[6] M. Mamei, M. Vasirani, F. Zambonelli. Experiments in Morphogenesis in Swarms of Simple Mobile Robots. *Applied Artificial Intelligence,* 18, 9-10: 903-919, 2004.

[7] A. Kondacs. Biologically-inspired Self-assembly of 2D Shapes, Using Global-to-local Compilation. *International Joint Conference on Artificial Intelligence,* 2003.

[8] K. Stoy, R. Nagpal. Self-reconfiguration Using Directed Growth. *International Symposium on Distributed Autonomous Robot Systems*, June, 2004.

[9] N. Gordon, I. Wagner, A. Brucks. Discrete Bee Dance Algorithms for Pattern Formation on a Grid. *International Conference on Intelligent Agent Technology*, 545, 2003.

[10] W. Spears, D. Spears, J. Hamann, R. Heil. Distributed, Physics-Based Control of Swarms of Vehicles. *Autonomous Robots 17*, 137-162, 2004

[11] Swarm Development Group: http://www.swarm.org