Distributed Spatial Control, Global Monitoring and Steering of Mobile Agents*

Diana F. Gordon and William M. Spears
AI Center, Naval Research Laboratory
Washington, D.C. 20375
gordon@aic.nrl.navy.mil

Oleg Sokolsky and Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

Abstract

In this paper, we combine two frameworks in the context of an important application. The first framework, called "artificial physics," is described in detail in a companion paper by Spears and Gordon [13]. The purpose of artificial physics is the distributed spatial control of large collections of mobile physical agents. The agents can be composed into geometric patterns (e.g., to act as a sensing grid) by having them sense and respond to local artificial forces that are motivated by natural physics laws. The purpose of the second framework is global monitoring of the agent formations developed with artificial physics. Using only limited global information, the monitor checks that the desired geometric pattern emerges over time as expected. If there is a problem, the global monitor steers the agents to self-repair. Our combined approach of local control through artificial physics, global monitoring, and "steering" for self-repair is implemented and tested on a problem where multiple agents form a hexagonal lattice pattern.

1. Introduction

The objective of this research is the distributed control of large numbers of mobile physical agents to form regular geometric configurations, e.g., to act as sensing grids. During formation, the configurations are monitored by a global observer to detect whether there is a significant increase in the number of pattern violations over time. Our combined approach of distributed local control and global monitoring enables spatio-temporal coordination of the agents. The agents may range in scale from neurons, nanobots, or micro-electromechanical systems (MEMS) to micro-air vehicles (MAVs) and satellites. The example considered here is that of a swarm of MAVs whose mission is to form a hexagonal lattice, which creates an effective sensing grid. Essentially, such a lattice will create a virtual antenna or synthetic aperture radar to improve the resolution of radar images. A virtual antenna is expected to be an important future application of MAVs. Currently, the technology for MAV swarms (and swarms of other micro-vehicles such as micro-satellites) is in the early research stage. Nevertheless we are developing the control software now so that we will be prepared.

We assume agents can only sense and affect nearby agents; thus the problem is one of "local" control. The method for local control should be based on principles such as self-assembly, fault-tolerance, and self-repair. These principles are precisely those exhibited by natural systems. This leads us to look at the laws of physics for ideas on distributed control. To explore this, we have developed a general framework for distributed control in which "artificial physics" (AP) forces control agents. We use the term "artificial" because although we are motivated by natural physical forces, we are not restricted to only natural physical forces. The agents aren't really subject to real forces, but they can *act* as if the forces are real. Thus the agent's sensors will have to be able to see enough to allow it to compute the forces to which it is reacting. The agent's effectors should allow it to respond to this perceived force. For details on AP, see Spears and Gordon [13].

We see at least two advantages to AP. First, in the real physical world, collections of small entities yield surprisingly complex behavior from very simple interactions between the entities. Thus there is a precedent for believing that complex control can be achieved through simple local interactions. This is required for very small agents (such as nanobots), since their sensors and effectors will necessarily be primitive. Two, since the approach is largely independent of the size and number of agents, the results should scale well to larger agents and larger sets of agents.

AP addresses the problem of distributed agent control via local rules. This approach, which also includes fault-tolerance and local self-repair mechanisms [13], may be inadequate for handling major unanticipated events. For ex-

* This research was supported in part by ONR N00014-97-1-0505 and ONR N00014-99-WR20010 as part of the ONR Semantic Consistency MURI, as well as NSF CCR-9619910, ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466.

ample, if a swarm of MAVs is flying in formation, fault-tolerance and/or local self-repair capabilities could enable recovery from minor air turbulence. Nevertheless, intentional or unintentional corruption of the MAVs' control software, severe environmental conditions, or widespread mechanical failures could conceivably result in an unrecoverable problem maintaining the desired geometric formation. Therefore, we also include a global observer that monitors the progress of the formation, using the Monitoring and Checking (MaC) framework, which is described in detail in Kim *et al.* [6]. We do *not* make the strong assumption that the global observer can see the pattern – because this assumption may be infeasible for large numbers of widely distributed agents. We only assume that the observer can communicate with the individual agents. Each agent sends an alert if it fails to satisfy its local evaluation measure. The global observer collects the alerts and issues a general alarm if the local alerts are too frequent for too long. The general alarm might be sent to people nearby to persuade them to intervene and manually solve the problem by sending commands to the agents. Here, we assume that the general alarm suggests the need for "steering" (i.e., self-repair to recover from problems). In our approach to steering, the global observer broadcasts to the agents a global parameter change for self-repair. This restores progress toward the desired geometric configuration.

The novelties of this paper are: (1) the combination of AP with MaC, (2) the introduction of a steering method for self-repair when MaC detects a failure, and (3) experimental results that validate the usefulness of this combined approach in the context of hexagonal lattice formations. The paper begins by presenting the artificial physics framework. This is followed by a description of how AP can be used to generate hexagonal lattices. We then describe the MaC framework, and apply it to monitor the progress of forming hexagonal lattices. Finally, we present a method for steering that adjusts global parameters for self-repair. The paper concludes with some initial results, followed by related work and ideas for future research.

2. Artificial Physics: A Framework for Distributed Multiagent Control

Our artificial physics approach treats agents as physical particles, though their actual size may range from nanobots to satellites. A simple but realistic physical simulation of the particles' behavior was built. Particles exist in two dimensions (we see little difficulty in generalizing to three dimensions) and are considered to be point-masses. Each particle i has position $p = (x, y)$ and velocity $v = (v_x, v_y)$. We use a discrete-time approximation to the continuous behavior of the particles, with time-step Δt. At each time step, the position of each particle undergoes a perturbation Δp. The perturbation depends on the current velocity $\Delta p = v\Delta t$. The velocity of each particle at each time step also changes by Δv. The change in velocity is controlled by the force on the particle $\Delta v = F\Delta t/m$, where m is the mass of that particle and F is the force on that particle. An additional simple frictional force is also always included, for self-stabilization.

Given the initial conditions and some desired global behavior, we must define what sensors, effectors, and force F laws are required such that the desired behavior emerges. We explore this for hexagonal lattices.

2.1. Creating Hexagonal Lattices

This subsection explains the construction of hexagonal lattices, e.g., for MAV sensor grids. For MAVs, the initial conditions are assumed to be similar to those of a "big bang" – the MAVs are released from a canister dropped from a plane, then they spread outwards until a desired geometric configuration is obtained. This is simulated by using a two-dimensional Gaussian random variable to initialize the positions of all particles (MAVs). Velocities of all particles are initialized to be 0.0, and masses are all 1.0 (although the framework does not require this). An example initial configuration for 150 particles is shown in Figure 2.1.

Figure 1. The initial universe at $t = 0$.

Since MAVs (or other small agents such as nanobots) have simple sensors and primitive CPUs, our goal is to provide the simplest possible control rules that require minimal sensors and effectors. At first blush, creating hexagons would appear to be somewhat complicated, requiring sensors that can calculate range, the number of neighbors, their angles, etc. However, it turns out that only range information is required. To understand this, recall an old high-school geometry lesson in which six circles of radius R can be drawn on the perimeter of a central circle of radius R (the fact that this can be done with only a compass and straightedge can be proven with Galois theory). Figure 2 illustrates this construction. Notice that if the particles (shown as small circular spots) are deposited at the intersections of the circles, they form a hexagon.

The construction indicates that hexagons can be created via overlapping circles of radius R. To map this into a force law, imagine that each particle repels other particles that are closer than R, while attracting particles that are further than R in distance. Thus each particle can be considered to have a circular "potential well" around itself at radius R – neighboring particles will want to be at distance R from each

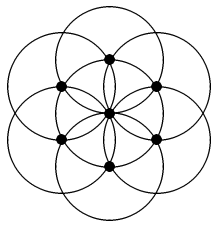

Figure 2. How circles can create hexagons.

other. The intersection of these potential wells is a form of constructive interference that creates "nodes" of very low potential energy where the particles will be likely to reside (again these are the small circular spots in the previous figure). Thus the particles serve to create the very potential energy surface they are responding to!

With this in mind we defined a force law $F = Gm_i m_j / r^2$, where F is the magnitude of the force between two particles i and j, and r is the range between the two particles. The "gravitational constant" G is set at initialization. The force is repulsive if $r < R$ and attractive if $r > R$. Each particle has one sensor that can detect the range to nearby particles. The only effector is to be able to move with velocity v. To ensure that the force laws are local in nature, particles can not even see or respond to other particles that are greater than $1.5R$ in distance. [1]

The initial universe of 150 particles (as shown in Figure 2.1) evolves, using this very simple force law. For a radius R of 50 we have found that a gravitational constant of $G = 1200$ provides good results (these values for R, G, and the number of particles remain fixed throughout this paper). Figure 3 shows the system after 35 time steps.

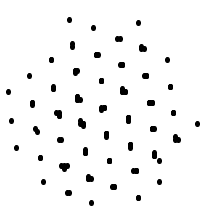

Figure 3. The 150 particles form a good hexagonal lattice by $t = 35$.

There are a couple of important observations to make about Figure 3. First, a reasonably well-defined hexagonal lattice has been formed from the interaction of simple local force laws that involve only the detection of distance to nearby neighbors. Also, the perimeter is not a perfect hexagon, although this is not surprising, given the lack of global constraints. However, many hexagons are clearly embedded in the structure and the overall structure is quite hexagonal. The second observation is that each node in the structure can have multiple particles (i.e., multiple particles can "cluster" together). Clustering is an emergent property that provides increased robust (fault-tolerant) behavior, because the disappearance of individual agents from a cluster will have minimal effect.

[1] The constant 1.5 is not chosen randomly. In a hexagon, if a nearby neighbor is further than R away, it is $\geq \sqrt{3}R$ away. We wanted the force laws to be as local as possible.

2.2. Discussion

The artificial physics framework offers a number of advantages. For one, it enables large numbers of agents to self-assemble into geometric lattices. Here, we have shown the method for assembling hexagonal lattices. With a minor extension (the introduction of a "spin" attribute), agents can also self-assemble into square and other lattices. Fault-tolerance from clustering is another advantage of the AP framework. Furthermore, in Spears and Gordon [13], it is shown that there is an effective offline evaluation measure of lattice quality that averages the angular error throughout the lattice. This is useful during program development. Finally, Spears and Gordon [13] present effective local self-repair methods that can fill gaps in the lattice (empty nodes) and reduce the angular error.

Although AP has the desirable attributes of enabling self-assembly, fault-tolerance, and local self-repair, it cannot address all problems that the agents might encounter. In particular, although the offline measure of lattice quality provides assistance during program development, it relies on measuring angles and making geometric comparisons between agents that are far apart in the lattice. As stated earlier, we do not want agents to have to measure angles, and we cannot assume sensors that detect other agents beyond the visibility range. Therefore we require a simpler *online* measure of lattice quality. Furthermore, although the local self-repair methods are effective for repairing empty nodes and global flaws in angles (such as those detected by the angular error measure), they are not capable of restoring the lattice after severe disturbances that distort the shape of the perimeter. An example of a potential hazard for an MAV is air turbulence. MAVs are expected to be small (less than six inches in length, width, and height), slow (traveling 22-45 miles per hour), and light (50-70 grams). This translates into a low Reynolds number, which implies that for practical purposes inertia can be ignored and the MAVs will be especially vulnerable to air turbulence [7]. Our solution is to add Monitoring and Checking.

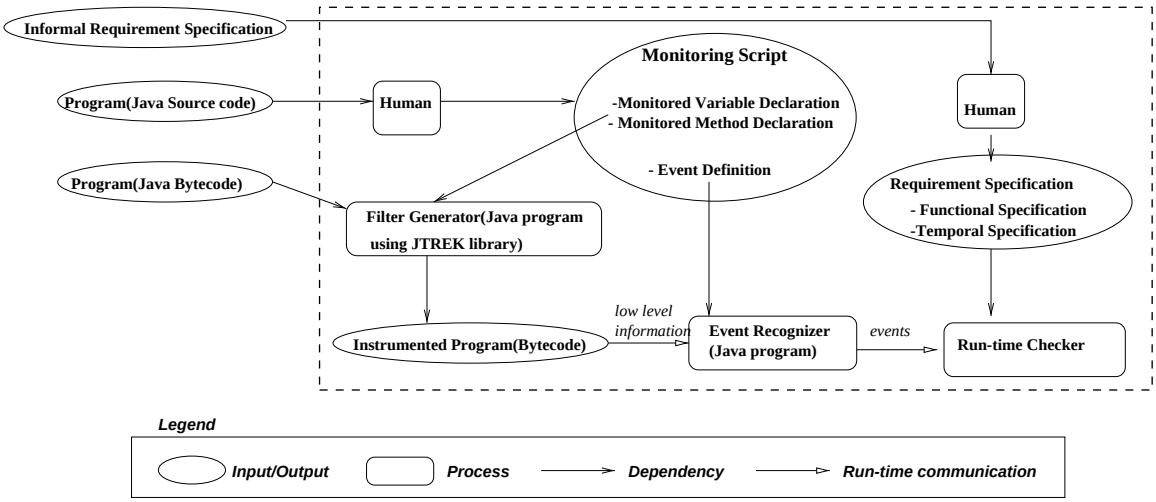

Figure 4. Overview of the MaC framework.

3. A Framework for Global Monitoring

The Monitoring and Checking (MaC) framework (see Figure 4) aims at run-time assurance monitoring of real-time systems. The current implementation is in Java, though the framework is generic and can apply to any language. The framework includes two main phases: (1) before the system is run, its implementation and requirement specification are used to generate run-time monitoring components; (2) during system execution, information about the running system is collected and matched against the (user-generated) requirements.

During the first phase, MaC provides a mapping between high-level events used in the requirement specification and low-level state information extracted during execution. They are related by means of a *monitoring script*, which describes how events at the requirements level are defined in terms of monitored states of an implementation. For example, in the requirements we may want to express the event that the agents are `tooClose`. The implementation stores the information about proximity in a variable `distance`. In an execution state, this variable has a particular value. The monitoring script in this case can define the event `tooClose` as `(distance > 0.25*R) && (distance < 0.75*R)`. This definition of `tooClose` captures the notion that if neighboring particles are $\leq 0.25R$ apart then we permit this because they are in the same cluster (node); however, if they are not in the same cluster then we want them to be approximately R apart.

The monitoring script is used to automatically generate a *filter* and an *event recognizer* for run-time monitoring. The filter is a set of program fragments that are inserted into the implementation to instrument the system. Instrumentation is performed statically directly on the code (bytecode in the case of Java). Instrumentation is automatic, which is made possible by the low-level description in the monitoring script. The essential functionality of the filter is to keep track of changes to monitored objects and send pertinent state information to the event recognizer.

The monitoring script is also used to automatically generate the event recognizer. The event recognizer detects, according to the monitoring script, occurrences of high-level events from the data received from the filter. The purpose of the event recognizer is to deliver events to a *run-time checker*, described below.

Also, during the first phase the user formalizes the system requirements in a *requirements specification*. The requirements in this specification are defined in terms of events (which are defined in the monitoring script). A run-time checker is produced automatically from the requirements specification. The purpose of the run-time checker is to determine at run-time whether the system is satisfying its requirements.

In summary, during the first phase the user defines a requirements specification and a monitoring script. The requirements specification defines what the user expects of the system. The monitoring script provides event definitions necessary for the requirements specification. From the monitoring script, a filter and event recognizer are automatically generated, and from the requirements specification, a run-time checker is automatically generated.

During the second (run-time) phase, the instrumented implementation is executed while being monitored. The filter sends relevant state information to the event recognizer, which detects events. These events are then relayed to the run-time checker, which checks adherence to the user-desired requirements.

```
Requirement Specification HexagonPattern

import event MAValert, startPgm;

Auxiliary Variables :

 var long currInterval;
 var int  count0, count1, count2;
 var int  prevAverage, currAverage;

Alarm Definition :

 property NoPattern =
   (currAverage > prevAverage*1.15 + 100) &&
   (prevAverage != -1);

Auxiliary Variable Definitions:

 event startPeriod = (time(MAValert) -
   currInterval > 10000);
 startPgm -> {
   currInterval = time(startPgm);
   count0 = 0;
   prevAverage = -1;
   currAverage = -1; }
 startPeriod -> {
   currInterval = currInterval + 10000;
   prevAverage = currAverage;
   currAverage = (count0+count1+count2)/3;
   count2 = count1;
   count1 = count0;
   count0 = 0; }
 MAValert -> {
   count0 = count0 + 1; }
```

Figure 5. MEDL requirement specification.

3.1. The Monitoring Language

We give a very brief overview of two languages: one to describe monitoring scripts (i.e., what to observe in the program), and the other to describe the requirements specification (i.e., the requirements that the program must satisfy). For more details on the logical framework of these languages, see Kim *et al.* [6].

The language for monitoring scripts is called the **P**rimitive **E**vent **D**efinition **L**anguage (PEDL). Requirement specifications are written in the **M**eta **E**vent **D**efinition **L**anguage (MEDL). The primary reason for having two separate languages in the MaC framework is to separate implementation-specific details of monitoring from the requirements specification. This separation ensures that the framework is portable to different implementation languages and specification formalisms, while providing a clean interface to the designer of monitors. For example, if we wish to retarget our system from programs written in Java to C++, then all we would need to modify is the syntax of PEDL, leaving MEDL unchanged.

```
Monitoring Script MAVpattern

export event MAValert, startPgm;

Monitored Entities :

 monobj int Hexagon.R;
 monmeth void EmulateMAV.main(String[]);
 monobj double Mav.run().distance;

Event Definitions :

 event startPgm =
   startM( EmulateMAV.main(String[]) );
 event tooClose =
   (Mav.run().distance > 0.25*Hexagon.R) &&
   (Mav.run().distance < 0.75*Hexagon.R);
 event tooFar =
   (MAV.run().distance > 1.25*Hexagon.R) &&
   (MAV.run().distance < 1.5*Hexagon.R);
 event MAValert = tooClose || tooFar;
```

Figure 6. PEDL script.

The design of PEDL, the language for writing monitoring scripts, is based on the following two principles. First, we encapsulate all implementation-specific details of monitoring in PEDL scripts. Second, we want event recognition to be as simple as possible. The name of the language reflects the fact that the main purpose of PEDL scripts is to define primitive events that can be referenced in requirement specifications.

The requirements that need to be monitored are written in MEDL. Like PEDL, MEDL is based on a logic of events. This logic has a limited expressive power. For example, one cannot count the number of occurrences of an event, or talk about the ith occurrence of an event. Because we need additional expressive capabilities such as counting for the requirements specifications, MEDL allows the user to define auxiliary variables. Updates of auxiliary variables are triggered by events. For example, `MAValert -> count0 = count0 + 1` can be interpreted as stating that the occurrence of event `MAValert` triggers the system to increment the auxiliary variable `count0`. MEDL also allows the definition of complex events using expressions of primitive events and auxiliary variables.

Correctness of the system is described in terms of *alarms*, which are events that should *never* occur. Alarms are defined in terms of events and/or auxiliary variables.

4. Global Monitoring and Steering of Hexagonal Lattice Formations

Our approach assumes that one agent acts as a global observer to monitor the formation. The observer might be the plane that dropped the MAVs. This global observer uses MaC to determine whether the desired pattern of agents is forming as expected. We do not make the strong assumption that the global observer can see the pattern. This assumption may be infeasible for large numbers of widely distributed agents. Instead, we only assume that the observer can communicate with the individual agents.

This approach to monitoring is based on the observation that in the hexagonal lattice, each neighbor of an MAV is either at a fixed distance R (adjacent node), or very close to the MAV in question (same node). If the pattern is not fully formed, there are MAVs that have neighbors in other locations, and this can be detected as a violation of the pattern. Intuitively, we should expect that as the pattern forms, the number of such violations should decrease.

We call the requirement being specified a *property*. An implementation-independent MEDL specification of the property just described is shown in Figure 5. The primitive event `MAValert` (abbreviated "alert") denotes a spatial misplacement of some neighbor of an MAV. For the purpose of counting alert events, time is divided into intervals. Auxiliary variable `count0` is used to count the number of violations (alerts) of the pattern in the current interval. When an interval elapses, the number of alerts over this interval and the previous two are averaged. In other words, averaging is done over a sliding window of size three. The reason for averaging is to reduce the variance in alert numbers. This average is compared with the average obtained at the end of the previous interval. If a significant increase in the number of violations is detected (measured as `currAverage` $>$ `prevAverage*1.15 + 100`, which is an empirically determined threshold), then an alarm `NoPattern` is sent as notification of a pattern formation problem.

The AP-MaC combination has been implemented in Java. Monitoring is applied to a distributed emulator of MAV deployment. Each MAV is represented as a separate instance of class `MAV`, based on the standard Java class `Thread`. When the thread in an MAV is run, it continuously executes the AP positioning algorithm and queries its neighbors for their positions. A local variable `MAV.run().distance` in the `run()` method of the `MAV` class is used to hold the distance from the currently queried neighbor. `Hexagon.R` is the variable for the desired hexagon radius R. The monitoring script in PEDL for this implementation is shown in Figure 6. It defines event `MAValert` in terms of the value of the variable `MAV.run().distance`. Event `MAValert` is defined to occur if a neighbor MAV is `tooClose` or `tooFar`. By declaring the variable as a monitored entity, the script instructs the filter to report all updates of this variable so that they can be compared with the acceptable range of values described in the script.

Figure 7. Formation after a blast.

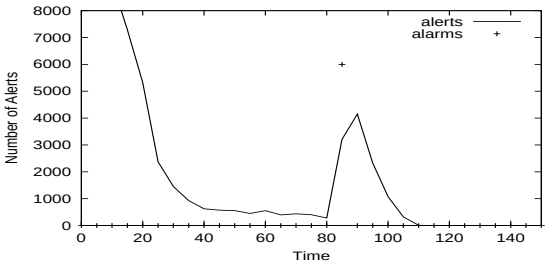

Figure 8. Alerts and alarms, with blast.

5. Experimental Results

We tested the combined AP-MaC implementation with 150 simulated MAVs. AP was used to form the lattice, and MaC determined whether the frequency of `MAValerts` (alerts) was increasing. Whenever the frequency increased significantly, a `NoPattern` alarm was issued. The requirements specification and monitoring script used were those shown in Figures 5 and 6, respectively.

Under normal conditions, the number of alerts gradually decreases as the hexagonal lattice is formed, and no alarms are issued. Now that we have a method for monitoring, to test this method we need to subject the MAVs to an unexpected yet severe environmental condition that disrupts the formation. We have implemented this as a blast (i.e., an explosion that causes a gust of wind), which is applied to one side of the lattice after it has been formed. The effect of the blast on an MAV is inversely proportional to the square of the MAV's distance from the center of the blast. In particular, this force is $F = 100Gm_i/{r_i}^2$, where m_i is the mass of MAV i and r_i is its distance from the center of the blast. Figure 7 shows the formation after a blast has been injected. Figure 8 shows the profile of alerts and alarms resulting from a blast. The number of alerts de-

creases as the hexagonal lattice is formed. However after the blast, injected around time 80, the number of alerts increases enough to set off an alarm. This profile is typical. In this figure, if an alarm is absent its magnitude is 0; if an alarm is issued, its magnitude is 6000. This is done for graphical convenience. In reality, alarm `NoPattern` has no magnitude; it is binary-valued.

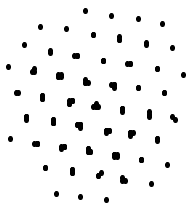

Figure 9. Formation after steering.

Although the number of alerts decreases after a sufficiently long time following the blast, from visual inspection we often find that the formation is not a well-defined hexagonal lattice. Local flaws can be fixed with local self-repair methods, but the concavity in the overall shape resulting from the blast typically persists. Therefore, steering is required. Steering consists of global parameter adjustment to compensate for the blast. In particular, the blast occurs, which sets off an alarm. After issuing an alarm, the global monitor broadcasts a global command to all MAVs to temporarily suspend repulsion. In other words, for a brief period of time all MAVs are told to assume that the force F is attractive only. During this time all of the MAVs gravitate toward each other. Also during this time alarms are suppressed. After the specified time period (in our experiments this lasted three intervals, i.e., one window of time), repulsion is resumed. The alarms are suppressed for another three intervals, however, to give the multiagent system time to settle down. Figure 9 shows the formation after steering.

Table 1. Shape improvement from steering.

	mean
$s_{orig} - s_{pre}$	1.8
$s_{orig} - s_{post}$	0.6

A typical profile of alerts and alarms with a blast followed by steering looks the same as in Figure 8 with no steering. To evaluate the effectiveness of steering, a useful *offline* measure of the quality of the overall shape is s = the size of (number of nodes per side in) the largest perfect embedded regular hexagon in the formation. This measure is applied to the original formation (s_{orig}), the post-blast pre-steering formation (s_{pre}), and the post-blast post-steering formation (s_{post}). Table 5 shows the differences $s_{orig} - s_{pre}$ and $s_{orig} - s_{post}$, averaged over 10 independent experiments. Smaller differences are better. Using an exact Wilcoxon rank-sum test, we find that there is a statistically significant difference between the two means ($p < 0.001$). Therefore the experimental results indicate that steering is an effective method for recovering a good lattice. In conclusion, our method of global monitoring and error recovery appears promising.

6. Related Work

Others have examined physical simulations of self-assembly. Schwartz et al. [12] have investigated the self-assembly of viral capsids. Winfree [14] has investigated the self-assembly of DNA double-crossover molecules on a 2D lattice. Both Schwartz et al. and Winfree are restricted to using plausible models of natural physics, since they are investigating the self-assembly of small natural particles. AP, however, is not bound by this restriction.

AP is also closely related to the work of Carlson et al. [2], which investigates techniques for controlling miniature agents such as micro-electromechanical agents and nanobots. Their work relies heavily on the use of a global controller that can impose an external potential field that agents can sense. Since we rely primarily on local force interactions, the work by Carlson et al. could be complementary to our work.

AP is similar to the "potential field" and "behavior based" approaches to robotics. Potential field (PF) approaches [4, 5] are used for robot navigation and obstacle avoidance. Like AP, PF approaches model a goal position as an attractive force, while obstacles are modeled with repulsive forces. PF computes force vectors by taking the gradient of an entire potential field. In AP, however, each particle directly computes the force vector that applies to its current position – the potential field is never computed. AP thus has lower computational overhead.

Behavior based approaches (e.g., [1]) derive vector information in a fashion similar to AP. However, behavior based approaches do not make use of potential fields and forces. Rather, they deal directly with velocity vectors. This distinction is significant for two reasons. First, AP can mimic natural physics phenomena more easily since it deals directly with forces. Second, unlike behavior based approaches, AP has the potential of being analyzable with conventional physics techniques.

There is also research related to MaC. Although most research in verification does not address correctness at execution time, recently several research efforts have begun to address run-time monitoring. Yet they all differ from the MaC framework. For example, in [3, 11], only the bus activity can be monitored. In our opinion, instrumentation of a variety of key points in the system allows us to de-

tect violations faster and more reliably. Several monitoring approaches concentrate on a reduced class of safety properties, e.g., [10, 8]. By contrast, MaC can monitor all safety properties. Another novelty of our work is that it addresses properties with spatial constraints. Previous system verification methods have focused almost exclusively on verifying temporal properties.

Finally, our combined approach, which includes local rules for self-assembly of distributed agents into a geometric formation and global monitoring and steering, is unique. We were motivated to use decentralization (local rules) as the basis of our approach because agents, such as MAVs, may have severe cost and weight limitations, thereby posing extreme restrictions on the range and number of sensors and the processing power. The primary disadvantage of our approach over purely decentralized approaches is the requirement of a global observer. Nevertheless, it is not possible to recover from severe disturbances to a formation without some central repository of information. To minimize the amount of global information, we only require that the global observer collect alerts from the agents. Furthermore, our approach includes a novel and successful method for formation repair.

7. Conclusions and Future Work

In this paper, we have combined two frameworks. The first, called artificial physics, is used for distributed spatial control of large collections of mobile agents via local artificial forces. The second framework is for global monitoring of the agent formations. Furthermore, we have added a steering capability for self-repair. From our experimental results, we can see that the combined approach is effective and potentially useful. We plan to explore a variety of steering methods.

Another future direction will be to explore alternative geometric configurations (e.g., continue the direction of Spears and Gordon [13]) and alternative requirements specifications and monitoring scripts. The current property being monitored (i.e., that the MAVs must not be between $0.25R$ and $0.75R$ or between $1.25R$ and $1.5R$) does not *necessarily* enforce a hexagonal lattice. The exploration of more sophisticated property requirements would be valuable. Also, as PEDL scripts become more sophisticated, we will need to address issues related to the expression of qualitative spatial relations, such as those in Mukerjee [9].

In conclusion, we have presented an approach to distributed spatial control, global monitoring, and steering of collections of agents that is independent of the number and size of the agents. This combined framework has potential applicability to a wide range of problems, including geometric formations for MAV sensing grids, a virtual space telescope, nanotechnology for MEMS, fleets of autonomous underwater vehicles, and configuring micro-satellites for better reception and transmission. This approach enables self-assembly of complex multiagent systems through artificial physics along with monitoring and self-repair to handle unanticipated severe events. Therefore, our novel combined approach takes us one step closer to the autonomous coordination of spatially distributed multiagent systems. When the technology for MAVs and other physical agents ripens to the extent that we have swarms of micro-agents, we would like to test our method on the actual vehicles.

References

[1] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):1–15, 1998.

[2] B. Carlson, V. Gupta, and T. Hogg. Controlling agents in smart matter with global constraints. In *AAAI-97 Workshop on Constraints and Agents*, 1997.

[3] M. Diaz, G. Juanole, and J.-P. Courtiat. Observer - a concept for formal on-line validation of distributed systems. 20(12):900–913, Dec. 1994.

[4] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[5] J. Kim and P. Khosla. Real-time obstacle avoidance using harmonic potential functions. In *Proc. IEEE International Conference on Robotics and Automation*, pages 790–796, 1991.

[6] M. Kim, M. Viswanathan, H. Ben-Abdallah, S. Kannan, I. Lee, and O. Sokolsky. Formally specified monitoring of temporal properties. In *Euromicro Conference on Real-Time Systems, to appear*, 1999.

[7] J. McMichael and M. Francis. Micro air vehicles – toward a new dimension in flight. *http://www.darpa.mil/tto/mav/mav_auvsi.htm*, 1997.

[8] A. K. Mok and G. Liu. Efficient run-time monitoring of timing constraints. In *IEEE Real-Time Technology and Applications Symposium*, June 1997.

[9] A. Mukerjee. Neat vs scruffy: A survey of computational models for spatial expressions. In P. Olivier and K. P. Gapp, editors, *Computational Representation and Processing of Spatial Expressions*, 1998.

[10] S. Sankar and M. Mandal. Concurrent runtime monitoring of formally specified programs. In *IEEE Computer*, pages 32 –41, March 1993.

[11] T. Savor and R. E. Seviora. An approach to automatic detection of software failures in real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 136 –146, June 1997.

[12] R. Schwartz, P. Shor, P. Prevelige, and B. Berger. Local rules simulation of the kinetics of virus capsid self-assembly. *Biophysics*, 75:2626–2636, 1998.

[13] W. Spears and D. Gordon. Using artificial physics to control agents. In *IEEE International Conference on Information, Intelligence, and Systems*, 1999.

[14] E. Winfree. Simulations of computing by self-assembly. *DIMACS: DNA-Based Computers*, June 1998.