Exerting Human Control Over Decentralized Robot Swarms

Zsolt Kira
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
Email: zkira@gatech.edu

Mitchell A. Potter
Navy Center for Applied Research in Artificial Intelligence
U.S. Naval Research Laboratory
Washington, DC 20375, USA
Email: mpotter@aic.nrl.navy.mil

***Abstract*—Robot swarms are capable of performing tasks with robustness and flexibility using only local interactions between the agents. Such a system can lead to emergent behavior that is often desirable, but difficult to control and manipulate post-design. These properties make the real-time control of swarms by a human operator challenging—a problem that has not been adequately addressed in the literature. In this paper we present preliminary work on two possible forms of control: top-down control of global swarm characteristics and bottom-up control by influencing a subset of the swarm members. We present learning methods to address each of these. The first method uses instance-based learning to produce a generalized model from a sampling of the parameter space and global characteristics for specific situations. The second method uses evolutionary learning to learn placement and parameterization of virtual agents that can influence the robots in the swarm. Finally we show how these methods generalize and can be used by a human operator to dynamically control a swarm in real time.**

I. Introduction

Swarm intelligence is characterized by simple agents that exhibit complex emergent behavior as a result of local interactions between themselves and their environment. These characteristics lead to robustness, flexibility, and simplicity of the control systems. Failure of an individual agent does not cause a catastrophic failure of the swarm because the duties of the failed member can be easily performed by another agent.

These same characteristics, however, lead to group behavior that can be very complex and hard to model, making attempts to control global behavior by the designer and operators difficult. Although some progress has been made in the development of techniques for engineering swarm behaviors [1], [2], little work has focused on *real-time* control of swarms after deployment. Real-time control is a particularly important issue because the locality and unpredictability of emergent behavior makes influencing the entire swarm after deployment both difficult and necessary. The main contribution of this paper is to introduce and explore the novel problem of post-design control of decentralized robotic swarms. We identify two possible flavors of control (top-down and bottom-up), present learning methods to address each form of control, demonstrate the approach in the context of a defense scenario, and analyze its generality.

The need for such real-time control of robots during operation is clear. Sometimes the swarm dynamics may result in undesirable behavior due to modeling or design error, or the situation changes such that a different group behavior is needed. There is a trade-off between the need for robot teams that are robust with respect to member failures and teams that can be influenced or controlled when necessary.

In the work described here swarms are controlled with *physicomimetics* [3]. In physicomimetics, forces similar to those existing in the real world such as gravity act upon agents and determine their movement. One can define agents with different masses, coefficients of friction, and so on, that are influenced by various force laws. The combination of these parameters can produce a wide variety of useful behaviors. This system has allowed for useful physics-based analysis of emergent behaviors and has been shown to be both robust and scalable [4]. It has been demonstrated on a number of applications such as chemical plume tracing and surveillance [5], [6]. Previous work has also looked at generalizing the system to include particle *types* and making the engineering of physicomimetics swarms easier by providing a graph-based representation of the various agent interactions that reduces the parameter space [7]. Note that although we use physicomimetics to control the interactions between agents, the mechanisms proposed for real-time control can be easily generalized to other swarm-based systems.

In this paper we present preliminary work that explores how an operator can influence the resulting swarm after deployment in a task that requires multiple behaviors. In particular, we look at two possible mechanisms of real-time control. The first mechanism involves a top-down approach whereby global swarm characteristics are defined and the parameter settings of the individual agents are optimized to achieve such characteristics. Instance-based learning is used to generalize a sampling of the space of possibilities with respect to the characteristics and store solutions in the form of parameter settings. The second control mechanism is a bottom-up approach that does not modify the behavior of the current agents. Instead, *virtual* agents that do not exist in the environment interact with the real agents via the same force law mechanisms and can therefore influence swarm behavior. We use evolutionary learning to learn placement and parameterization of these virtual agents. Finally, we show how these methods can be used by a human operator to dynamically

influence a swarm in real time and we analyze the generality of the approach.

II. Related Work

The concepts of predicting and controlling swarm behavior have been addressed in several categories of related work. First, in the emerging subfield of swarm engineering design-time methods have been developed for creating swarms that exhibit desirable behavior. Kazadi describes a mathematical swarm engineering methodology whereby a swarm condition is defined by an equation and global behavior is defined in the limit [1]. For example, they attempt a task where robots attach to each other to form chains, and each robot that is in a chain swarm has an associated rank number depending on its position in the chain. The swarm condition is the summation of these ranks, and can be used to control average chain length. This then leads to more analysis leading to the design of the actual local behaviors of the robots, determining when they should break off chains, and so on. Mamei et al. developed Co-Fields, a canonical unifying representation for many swarm domains [8]. The mathematical analysis therefore only has to be done for this representation versus a separate analysis for each domain, and a dynamical systems model is provided.

These latter methods of engineering local rules for robots given an analysis of global characteristics are related to the idea of swarm modeling. There are two distinct approaches: microscopic and macroscopic models. Microscopic models take an agent-level perspective and can use mathematical models or simulation in order to predict behavior. Because microscopic models can be computationally expensive, macroscopic models were developed to describe collective behavior at an average level and do not model individual behavior. Macroscopic models have been successfully applied to the prediction of foraging behavior in a swarm of Markov agents [9], [10]. However, these methods require analytic models which may not be available or possible to derive for some domains. In this paper, we use simulation to derive instance-based models of global characteristics. Unlike the previous work in modeling collective behavior, we consider the problem of using these models to allow real-time changes in swarm characteristics by a user, as opposed to modeling swarms to gain information during design-time.

There is also related work in human control of robot swarms. The issue of multi-agent telerobotics, or human control of a small group of robots, was explored before large swarms were possible [11]. In more recent work, Olsen and Wood look at a human-robot interaction methodology for modeling human control of semi-autonomous swarms, based on: how many robots can be controlled, the amount of time a robot is effective after user commands, and the time it takes to interact with the robot [12]. Perron and Zhang look at human coaching of a robotic multi-agent soccer team, and explore issues of the abstraction of the commands given and selection of individuals in groups [13]. Finally, McLurkin et al. describe philosophy and experiences with user interfaces and other issues for controlling large robotic swarms [14]. These papers deal with *direct* human control of all of the swarm members, while we propose novel methods of *indirect* operator influence over swarms that operate within the framework of local rules in a decentralized manner.

III. Framework

A. Physicomimetics

While the contributions presented in this paper are not restricted to any one swarm-control method, we use physicomimetics control in the work described here. Physicomimetics is a form of distributed multi-agent control where agents are treated as point-mass particles and their movement results from the interaction of artificial forces that act on these particles [3]. Robots exist in a physical world and their movement is constrained by actual physical forces, but they may also be acted on by physicomimetic forces that are translated into motor controls for pushing or pulling the robot in a particular direction. We used a force law from previous physicomimetics work inspired by Newton's law of universal gravitation [3], [4], [7]. The force between two particles i and j is:

$$F_{ij} = \begin{cases} -G\frac{(m_i m_j)^a}{r_{ij}^d} & \text{if } r_{ij} \in [0, R) \\ G\frac{(m_i m_j)^a}{r_{ij}^d} & \text{if } r_{ij} \in [R, C] \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where F_{ij} is the force acting on the two particles, C is the range of effect of the force, R is the range at which the force switches from repulsive to attractive, G is the gravitational constant, m_i and m_j are the masses of the two particles, r_{ij} is the distance between the two particles, and the exponents d and a are used to emphasize the effect of distance and mass respectively. To provide a larger range of possible system behaviors each particle is assigned a *type*, and force law parameters are specified for each particle-type pair that will potentially interact [7]. Forces between a pair of particles with unequal types need not be symmetric.

Speed and turn rate commands are issued to each robot on a discrete time interval. A velocity delta is first computed as $\Delta v = F/m$, where m is the mass of the particle associated with the robot and F is the summation of the physicomimetic forces acting on it. The robot's target velocity is then computed as $v_{t+1} = (v_t + \Delta v)(1.0 - c_f)$, where c_f is the particle's coefficient of friction and v_t is its current velocity.

B. Learning Methods for Top-Down and Bottom-Up Control

There are several potential ways that a designer and operator can control a swarm. The most common of these is a bottom-up approach in which the designer specifies the types of agents involved and their rules of interaction, in addition to the control systems. Desirable but more difficult is a top-town approach in which the designer has global behaviors (or more abstractly, *characteristics*) in mind, and the system takes these as input and derives local rules to achieve them. Recently, several engineering design cycles have been proposed to integrate these two approaches and ensure that the resulting emergent behavior is the desirable one [1], [2]. Note that these

approaches have been studied in the context of *design-time* decisions, where the interactions are decided ahead of time and the resulting system is then deployed and left to run autonomously.

As motivated in Section I, it may be desirable to influence swarms while they are running. In this case, top-down and bottom-up methods are both applicable as during design time. For example, one level of influence may be from a top-down perspective where the operator desires some global behavioral characteristics and can modify them in real-time. Another way to influence a swarm in real-time is by introducing new virtual agents that cannot affect their environment but that influence real agents via the same interaction mechanism. The distinction between real and virtual agents is especially pertinent when dealing with robots, as the introduction of new physical objects into the environment is not practical.

In this paper we frame these two methods within a machine learning perspective and describe the inputs, assumptions, and optimization problems that each method presents. The descriptions will be in the context of physicomimetics, hence the agents will be modeled as point-mass particles.

C. Defining and Sampling Global Swarm Characteristics

Given a particular environment and number and types of agents, we would like the user to be able to define and dynamically change high-level characteristics. These characteristics abstract individual lower-level parameters of the physicomimetic system, allowing for more intuitive control. The key issue here is to model the mapping between low-level parameters and the higher-level characteristics. We use a sampling-based approach that is fed into a learning system in order to build this mapping.

We have defined two such characteristics that are generally useful, particularly when the swarm of agents have assembled into a formation of some sort. The first characteristic, *swarm radius*, is the radius of a circle circumscribing a swarm. One might want to increase the radius, for example, so the agents spread out to cover some desired area, or reduce the radius so the agents can pass through a narrow corridor. First, the centroid C of a swarm of agents S is calculated using the following equations:

$$C_S = (\overline{x}, \overline{y}), \qquad \overline{x} = \frac{1}{|S|}\sum_{a \in S} a_x, \qquad \overline{y} = \frac{1}{|S|}\sum_{a \in S} a_y \quad (2)$$

where a is one particular agent, and a_x and a_y are the global coordinates of agent a. The swarm radius r_S is then calculated as the maximum deviation from this centroid by all of the agents:

$$r_S = \max_{a \in S}(\sqrt{(a_x - \overline{x})^2 + (a_y - \overline{y})^2}) \quad (3)$$

The second global characteristic, *maximum inter-agent distance*, is calculated as:

$$d_S = \max_{a,b \in S}(\|a - b\|) \quad (4)$$

where $\|.\|$ is the Euclidean norm. This measure could be useful if the user requires the agents to be separated by a specific maximum distance from each other, for example, to keep agents from moving beyond a limited communication range.

TABLE I
PARAMETERS AND RANGES FOR PARTICLES AND NEWTONIAN FORCE LAW

Particle		*Force Law*				
m	c_f	C	R	G	d	a
1.0	0.9	$[0, 350]$	$[0, C]$	$[0, 2400]$	1.0	1.0

In order to find particle and force law parameters that achieve a certain global characteristic value, we use simulation to sample the parameter space and feed this into a learning system to build a more generalized model. Specifically, we use the nearest-neighbor learning method, although other more complex algorithms such as K-nearest neighbor may be used. Lazy learning is well-suited to this problem, as it creates a local model based on nearby instances in the space and hence can deal with situations in which the target function changes drastically in different parts of the space, as might happen in an emergent system. During training, the inputs are the parameter settings and resulting global characteristic values obtained via simulation. The nearest-neighbor model uses the global characteristic values as the index, and stores a new instance if it is sufficiently far away from previously known instances. After learning, a query consists of a desired global characteristic value, and results in an output of the parameter settings.

Table I shows all of the parameters and their ranges. For the two characteristics defined here we have determined that the most important parameters are C, R, and G. We have given the others fixed values to reduce the size of the search space.

One problem that may arise during this process is that different characteristics can interfere with each other; that is, changing one characteristic may cause the current value of another characteristic to not be achievable. The instance-based method deals with this by finding the nearest neighbor in terms of both characteristic values, and hence if a user requests a change in one characteristic value, it may be that the other one is modified slightly to make it possible.

D. Performing Swarm Operations with Virtual Particles

Operations on swarms, in contrast to global characteristics, seek to influence the behavior of an existing swarm without modifying their parameters. Instead, new virtual particles are added that can influence the swarm members via the same interaction mechanisms.

As in the previous section, we assume that the designer has specified the number and types of agents in the environment, and their interactions with one another through physicomimetic force laws assigned to particle-type pairs. Given a desired operation, the learning problem is to determine how many virtual particles to use, where to place the particles relative to the swarm, and what their interaction parameters with the agents will be.

For example, in a defense scenario there may be three types of agents in the environment: defenders, attackers,

and protected resources. Defenders may be attracted to the resources they are protecting as well as to attackers to chase them away, but repelled by other defenders to avoid clustering in too small an area. In this scenario two useful operations we could perform with virtual particles would be SPLIT and FOLLOW. The SPLIT operation separates a swarm of agents into two separate clusters and the FOLLOW operation directs a swarm along a specified trajectory. For example, if all the defenders formed a single cluster and the user wanted to lead some of them away to chase an attacker, the SPLIT operation could be performed first to physically separate the chasers from the other defenders—enabling the user to only influence the chasers when performing the FOLLOW operation.

To initiate a SPLIT, the two clusters can be differentiated by a separating axis specified by the user. We denote the defender agents to the left and right of the axis as being members of clusters L and R respectively. Virtual particles can then perform the SPLIT operation by exerting forces that maximize inter-cluster distances while minimizing intra-cluster distances. The inter-cluster distance is the distance between the centroids of the two clusters. Formally,

$$d_{L_C,R_C} = \sqrt{(\overline{x}_L - \overline{x}_R)^2 + (\overline{y}_L - \overline{y}_R)^2} \tag{5}$$

where $(\overline{x}, \overline{y})$ is a cluster centroid as defined in equation 2. The intra-cluster distance is the average distance between the members and their respective cluster centroid, and is computed (e.g., for cluster L and its centroid C_L) as follows:

$$d_{L,C_L} = \frac{1}{|L|} \sum_{a \in L} \|a - (\overline{x}, \overline{y})_L\| \tag{6}$$

To design virtual particles capable of performing the SPLIT operation, the inter-cluster and intra-cluster distance metrics are used by an evolutionary algorithm to determine the fitness of various virtual particles solutions. We have also added two additional terms to the fitness computation: a linear penalty for being distant from the protected resource and a penalty for having uneven clusters (i.e. if one side has more members than the other). The final fitness function is a weighted product of these four terms. The weights were determined empirically and were 2.0, 2.0, 1.5, and 1.0 for the inter-cluster distance, intra-cluster distance, resource distance, and cluster unevenness, respectively. Measurements are taken and fitness computed after the virtual particles are introduced and the system has a chance to stabilize.

Each set of virtual particles is described by a real-valued "genome" shown in Table II. Note that there is a single mass and coefficient of friction for up to 4 independently-positioned virtual particles, and the evolving parameters for the Newtonian force law only determine the influence the virtual particles will have on defenders. The virtual particles are themselves not influenced, so they remain fixed at their initial evolved positions which represent (x, y) offsets from the centroid of the defensive swarm prior to the SPLIT. Genomes were evolved with a generational evolutionary model using ranked selection, a population size of 60, and an adaptive Gaussian mutation operator [15] applied to each gene.

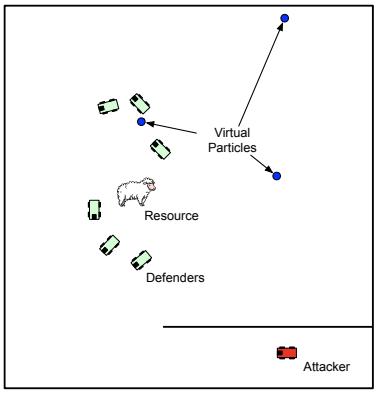

Fig. 1. Layout of the environment showing the six defenders, resource, attacker, and virtual particles implementing a SPLIT operation.

The FOLLOW operation only involves a single virtual particle positioned by the user, and its dynamics are simple enough to allow its other parameters to be designed by hand rather than resorting to evolutionary computation.

IV. Experimental Methodology and Results

A. Experimental Domain and Scenario

As in the work by Wiegand et al. [7], we focus on a defense domain where a swarm of robots protects a resource from incoming intruders. The domain shown in Figure 1 was implemented using the Player/Stage robotics simulator [16]. The protected resource is in the middle of the open space and the attacker comes from the bottom. Ideally, when the attacker approaches the resource it will be thwarted by the physicomimetic swarm as demonstrated in [7]. In our system, the operator can then decide to split the defensive swarm in half and lead one half toward the attacker. This is done by enacting the SPLIT and FOLLOW operations described previously. As the defenders pursue, the invader retreats to the narrow corridor it came from. This necessitates that the pursuing defenders decrease their swarm radius global characteristic in order to be able to fit through the corridor as a group. Once the invader is disabled, the operator can lead the pursuers back to join the defensive formation around the resource and adjust the swarm radius to provide a sufficient protective buffer.

B. Memory-based and Evolutionary Learning Results

In the scenario just described, the user needs to be able to control the radius of the swarm—decreasing the radius to maneuver the swarm into a narrow corridor and increasing the radius to provide a sufficient defensive buffer around a valuable resource. In order to achieve a range of radii, we explored, via simulation, a subset of the parameter space in the force law interactions between the particle types. We did this for two situations: one in which the defenders were surrounding the resource, and one in which the defenders were moving towards a goal (e.g. an attacker).

In the first situation, the largest determining factor of the radius is the force law between the defenders and the resource. As a result, we varied the R parameter of this force law while

TABLE II
VIRTUAL PARTICLE GENOME

particles	$(x, y) \times 4$	m	c_f	C	R	G	d	a
$[1, 4]$	$[-5.0, 5.0]$	$[0.1, 50.0]$	$[0, 1.0]$	$[0, 350.0]$	$[0, C]$	$[0, 2400.0]$	$[-5.0, 5.0]$	$[0, 5.0]$

keeping the C, G, d, and a parameters fixed. However, the forces between the defenders themselves also alters the swarm radius since the defenders must space themselves out if the repulsive forces between them are too large. This dynamic interaction is what makes it difficult to predict the global characteristic based on any one parameter alone. Hence, the simulation also varied the C, R, and G parameters of the force law between the defenders, while keeping the d and a parameters constant. Again, Table I shows the ranges for these parameters, and three separate parameter explorations were performed exploring different subsets of this space. This parameter exploration was also performed for the situation in which the defenders were pursuing a goal, but in that case the force law parameters between the defenders and the resource were not relevant and hence kept fixed. The resulting global characteristic values for many combinations of these four parameters were recorded during simulation, and the resulting data was fed into the instance-based learner.

Since in this domain we desired the defenders to spread themselves out in an approximately symmetric formation when surrounding the resource, we reduced the data set that was fed to the learning system based on this criteria. The nearest-neighbor learning system was trained on this reduced set of over three thousand points and the resulting library contained under one thousand representative cases. When the user then requests a particular global characteristic value, this library is queried and the low-level parameters are automatically configured based on the result. The system was tested by requesting six arbitrarily chosen values for the characteristics in the first situation, and measuring the actual resulting values when the parameters are applied to the swarm. This was done in simulation and on real MobileRobots, Inc. Pioneer 3-AT robots as shown in Figure 2, although only the three smallest values were tested on the real robots due to space limitations in our laboratory. Figure 3 shows the resulting error for both global characteristics, which was always significantly less than half a meter. Each point represents an average of ten runs, all starting from the same starting condition, with error bars indicating 95 percent confidence intervals.

In addition to adjusting the swarm radius in our defensive scenario, the user also needs to perform the SPLIT and FOLLOW operations. As previously mentioned, the FOLLOW operation is relatively simple and can be designed by hand. However, the SPLIT operation is more complex, involving multiple virtual particles, so we use evolutionary learning to determine appropriate particle parameters and positions to achieve the desired effect. Figure 4 shows the evolutionary learning curve for the SPLIT operation, performed while the agents are surrounding the resource. As can be seen, there are several sudden jumps in the learning curve corresponding to usage of more virtual particles, and the best performance used all four particles. The final evolved position of three of the virtual particles was previously shown in Figure 1.

Fig. 2. Six Pioneer robots in defensive formation around support column are about to receive request to change swarm radius.

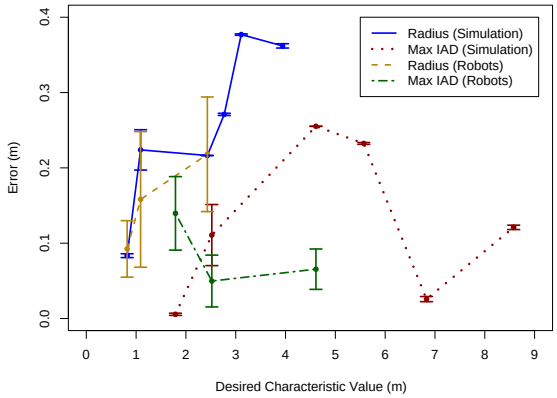

Fig. 3. Error between desired and achieved swarm radius and maximum inter-agent distance (IAD) for various characteristic values measured in simulation and on real robots.

Optimization of the parameters for the SPLIT operation was performed while the swarm was in a fixed radius from the resource. However, the learned solution is fairly generalizable, as shown in Figure 5, which plots the fitness during testing in which the swarm was placed (via the mechanism for changing global characteristics) in varying radii around the resource. As can be seen, the SPLIT operation performed well with radii within one meter of the trained radius. In some cases, such as when the swarm radius was set to the lowest value in the graph, the swarm members behaved completely differently than during training and clumped together; yet the SPLIT operation still achieved close to optimal fitness despite being trained on a swarm that was evenly dispersed in a circle around the resource.

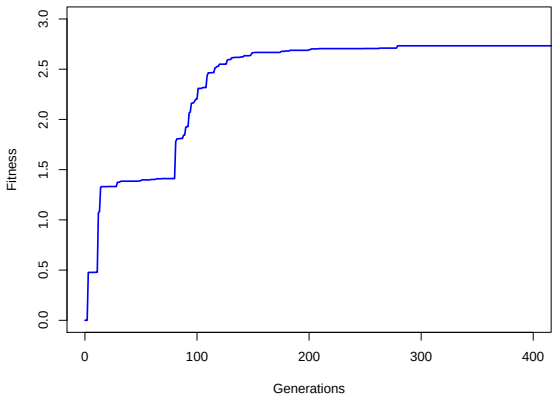

Fig. 4. Best-so-far fitness graph during evolutionary learning of SPLIT operation.

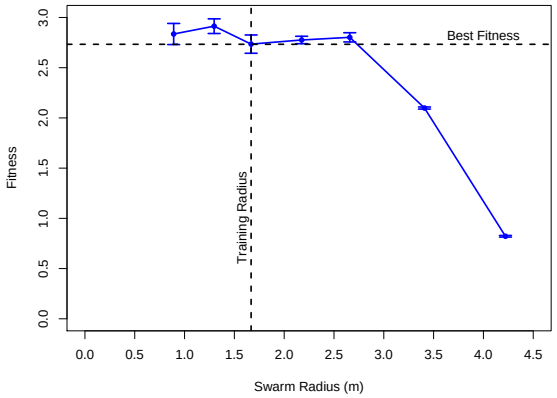

Fig. 5. Fitness results of SPLIT operation showing deviation from best fitness achieved during training when swarm radius is varied from the training radius.

V. Discussion and Conclusions

Engineering a swarm to perform desired behaviors is a difficult problem as emergent behavior can be complex and difficult to predict even with the simplicity of the control system. Although this problem is being heavily researched, little work has focused on enabling a human to have some element of control over the swarm after deployment, despite the clear desirability of this capability. We have shown that learning methods may be used to develop mechanisms for human control of swarms in real time. These methods can be applied to provide both top-down and bottom-up control.

Complexity was dealt with by simplifying assumptions and breaking down individual interacting elements, allowing one to ignore interactions that are not significant for the task. We do this at several different layers, including the abstraction of agent types via the generalized physicomimetic system, and at a higher level the separation of different situations (e.g. defending the resource versus chasing the attacker) during training. The latter abstraction limits situations for which learning must be performed. Despite this, we have shown in the results that there is some generalizability and robustness in the mechanisms, partially due to the robust physicomimetic representation. We have also shown that the instance-based model learned via simulation can be used to control real robots with comparable results. Another way in which we do this is by abstracting the features used in learning, for example, by making them dependent on a moving swarm centroid as opposed to static positions. This attempts to lessen the interaction between changes in other agents and the learned parameters. Future work will aim to increase the generalizability of the learned mechanisms to new unseen situations via more sophisticated learning methods. It would also be useful to characterize the situations in which a particular learned mechanism will or will not work, for example, by learning continuously during actual use.

Acknowledgment

This work was performed under Office of Naval Research Work Order N0001408WX20450.

References

[1] S. T. Kazadi, "Swarm engineering," Ph.D. dissertation, California Institute of Technology, 2000.

[2] A. F. Winfield, C. J. Harper, and J. Nembrini, "Towards dependable swarms and a new discipline of swarm engineering," in *Swarm Robotics: SAB 2004 International Workshop*. Springer, 2005, pp. 126–142.

[3] W. M. Spears and D. F. Gordon, "Using artificial physics to control agents," in *IEEE International Conference on Information, Intelligence, and Systems*. IEEE, 1999, pp. 281–288.

[4] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, "Distributed, physics-based control of swarms of vehicles," *Autonomous Robots*, vol. 17, pp. 137–162, 2004.

[5] D. Zarzhitskyi, D. Spears, D. Thayer, and W. M. Spears, "Agent-based chemical plume tracing using fluid dynamics," in *Formal Approaches to Agent-Based Systems, Third InternationalWorkshop, FAABS 2004*, ser. Lecture Notes in Computer Science, vol. 3228. Springer, 2005, pp. 146–160.

[6] W. Kerr and D. Spears, "Robotic simulation of gases for a surveillance task," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 2905–2910.

[7] R. P. Wiegand, M. A. Potter, D. A. Sofge, and W. M. Spears, "A generalized graph-based method for engineering swarm solutions to multiagent problems," in *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*. Springer, 2006, pp. 741–750.

[8] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: A physically inspired approach to motion coordination," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 52–61, 2004.

[9] K. Lerman, A. Galstyan, and T. Hogg, "Mathematical analysis of multiagent systems," 2004, e-print arXiv:cs/0404002v1 (arXiv.org).

[10] W. Agassounon, A. Martinoli, and K. Easton, "Macroscopic modeling of aggregation experiments using embodied agents in teams of constant and time-varying sizes," *Autonomous Robots*, vol. 17, no. 2-3, pp. 163–192, 2004.

[11] K. S. Ali, "Multiagent telerobotics: matching systems to tasks," Ph.D. dissertation, Georgia Institute of Technology, 1999.

[12] D. R. O. Jr. and S. B. Wood, "Fan-out: measuring human control of multiple robots," in *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, pp. 231–238.

[13] M. Perron and H. Zhang, "Coaching a robot collective," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4. IEEE, 2004, pp. 3443–3448.

[14] J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots," in *To Boldly Go Where No Human-Robot Team Has Gone Before: Papers from the 2006 Spring Symposium, Technical Report SS-06-07*. AAAI, 2006, pp. 24–31.

[15] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp. 1–29, 2000.

[16] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, 2005.