Optimizing Multiagent Area Coverage Using Dynamic Global Potential Fields

Rahul Rajan
U.S. Naval Research Laboratory
Washington, DC, USA
rahul.rajan@nrl.navy.mil

Michael Otte
University of Maryland
College Park, MD, USA
otte@umd.edu

Donald Sofge
U.S. Naval Research Laboratory
Washington, DC, USA
donald.sofge@nrl.navy.mil

***Abstract*—Multiagent coverage algorithms have been used in the context of search and rescue operations to determine optimal search patterns for a team of robots. Many solutions to the problem of area coverage have been discussed in the literature. Our approach covers a physics-inspired technique for global control of agents in a search space. However, more importantly we adopt an evolutionary approach to evolve a policy for dynamically changing the global control rules driving a team of agents. We implement this algorithm using the Robot Operating System and the Gazebo simulation platform.**

***Index Terms*—swarm, physicomimetics, search and rescue, coverage, potential fields, evolutionary algorithm**

I. Introduction

The need for more accurate, versatile, and inexpensive solutions for search and rescue has increased significantly over the years. As more and more natural disasters occur in populated areas and developing countries, we need affordable solutions that can be quickly deployed in any scenario to search for survivors and quickly alert human responders. The goal here is not to replace humans, but instead to assist them in their search. Increasingly, robots, specifically automated aerial vehicles aided with specialized sensors, have shown promise in recent years. While human rescuers extract survivors and get them to safety, aerial vehicles are used to search the area of interest and alert responders if they find something that could be a potential victim needing rescue.

Researchers are now looking into grouping these aerial robots together into teams and finding algorithms that enable them to collaborate or search in a way to optimize overall area coverage. Some approaches are top-down where computation is more centralized, while others are primarily bottom-up, where the agents themselves are more reactionary and rely on some form of communication with other agents and/or the environment to make decisions. Our approach can be implemented as either a bottom-up or top-down approach in principle, but we chose to implement the algorithm in a centralized manner as a proof of concept. Future work will cover a decentralized version of this approach.

II. Previous Work

A. Potential Fields

Potential field based methods have been used in robotics research for a long time. Typically, they've been used for obstacle avoidance tasks. The basic idea is that the search space is modeled as a potential field. Obstacles have high potential, which repel robots away, while targets or unexplored space may have lower potential. Khatib's team [1] discusses the idea of a time-varying potential field that changes based on the position of obstacles in the environment. This is especially useful in dynamic environments when obstacles or targets could be moving.

In the case of area coverage, potential field methods have emerged as a popular solution, especially for problems involving multiple agents. We define area coverage as the amount of area the team of agents has searched in a given amount of time. Many approaches have looked into modeling agents themselves as potential field generators that can repel other agents near them [2]. Howard and his team specifically designed a potential field that repelled mobile sensors (robots) from each other and the wall. Sydney implemented a technique that creates a temperature map, with an associated temperature gradient, across the search space [3]. The "heat" of different regions of the space changes based on the information gathered from that region, and the local temperature gradient determines the direction of the search agents. Thus, in this model, it is the search area itself moving the agents in different areas of the environment. A Lennard-Jones potential is combined with a temperature gradient to determine the resulting motions of the multi-agent system.

B. Stigmergy

Another important idea in the field of multi-agent area coverage is stigmergy via virtual pheromones. Stigmergy is the idea that agents can communicate with each other through the environment by dropping information and embedding it in the search space. We borrowed this idea in our previous paper where we combined ideas from stigmergy and force laws [4]. In this approach, agents dropped virtual pheromones in the environment with time-decaying weights. The current weight of a pheromone determines the repulsive force it places on nearby agents. Other work in stigmergy has looked into treating these virtual pheromones like trails [5] that serve as repellent regions which discourages agents from exploring the areas covered by the trails. Our approach aims for a similar concept, but our hope is that an evolutionary learning approach can learn this behavior over several generations. Some papers

actually use these trails more directly to guide exactly how agents move in the search space. Ranjbar-Sahraei discusses an algorithm $StiCo$ where agents move in fixed circles, marking their paths with a pheromone trail [6]. When an agent detects another agent's trail, it immediately switches directions to form a new circular path. Thus, this algorithm uses an approach where pheromones have very limited range and are only detectable when an agent goes over it. This is different from our previous approach with $PherPhys$ where the pheromones exert a force with a much longer range that tapers off with distance.

C. Neuroevolution

The idea of neuroevolution has been around for over 20 years, but it became very popular with the invention of NEAT (Neuroevolution of Augmented Topologies) [7]. This seminal work by Stanley introduced a novel way to use a fairly simple evolutionary algorithm to evolve both the topology and weights of a network simultaneously. This algorithm has been used in several fields as a viable alternative to traditional reinforcement learning approaches like Q-Learning. Since neuroevolution techniques like NEAT rely on a simple evolutionary algorithm under the hood, it is significantly less computationally intensive and is naturally parallelizable. Algorithms like NEAT have found success in tasks including walking and playing Atari games [8].

In terms of multi-agent systems, NEAT has shown promise in both homogeneous and heterogeneous swarms. Typically, heterogeneous swarms are evolved through a slightly modified version of NEAT that allows for co-evolution. This means agents are evolved using NEAT independently, but with the same fitness function. In many implementations, the agents will share their rewards equally. Thus, the members of the population are incentivized to develop their own independent strategies, but also to communicate with other agents in order to maximize their own reward [9]. In homogeneous situations, it is possible to evolve a single controller that is used by every agent in the population. Every genome is tested on all the agents simultaneously and the reward for the entire population is noted. Homogeneous neuroevolution has found success in a variety of swarm tasks, including foraging and searching. Ericksen and his team introduced an approach based on NEAT for evolving a general controller for a population of agents and showed that it outperforms popular foraging techniques [10].

III. Methodology

Our approach involves using a continuously changing global potential field to guide a multi-agent system to perform surveillance over a given search area. The agents themselves simply follow the gradient created by potential field generators distributed across the search space.

The remainder of the paper will delve deeper into how the algorithm works and what remains.

A. Features

The PhysField algorithm has 3 main parts — the global potential field, the agent, and the training node.

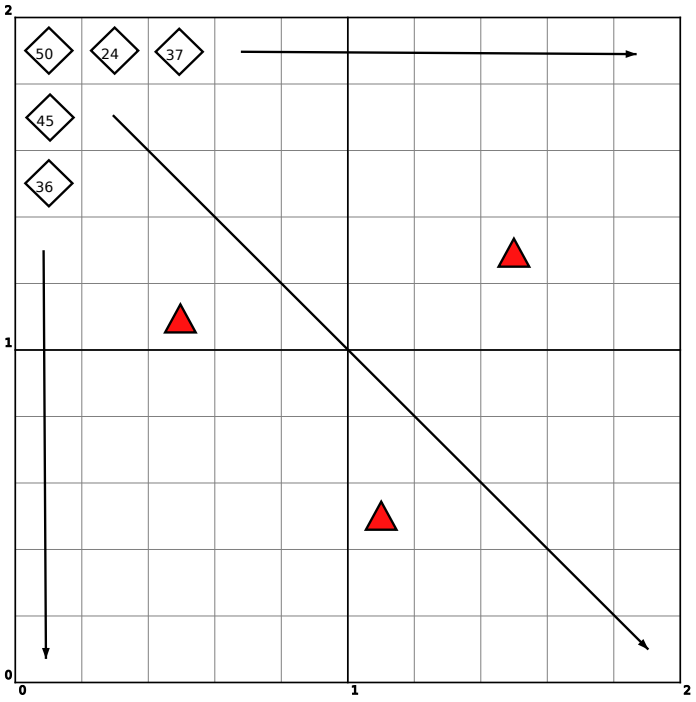

Figure 1. Potential Field Setup

The global potential field (GPF) consists of generators distributed at equally spaced intervals across the search space. Each generator has a different weight or strength which controls how the agents move through the space. A high positive weight would keep agents away, while a high negative weight would attract agents towards that area. Figure 1 shows an initial setup for the search space with the potential field generators laid on top. The diamonds enclosing a number represent the generators, while the red triangles represent the agents in the search space. The initial weights of the generators are randomly generated at the start of each experiment. Internally, these weights are saved in an array on a separate node that all agents can access simultaneously. The agents in the search space are represented as red triangles.

The Agent is responsible for following the field produced by the GPF, while also checking for collisions with obstacles, walls, and other agents. To prevent collisions with other agents, we employ a simple repulsive force between agents. This force follows the same inverse r-squared relationship found in Coulomb's law. When agents get a certain distance apart, the Agent will automatically employ this virtual force and adjust its velocity appropriately.

Finally, the training node serves to both monitor the experiment and change the potential field. This node utilizes a neuroevolution approach to train a neural network. This neural network is then used to change the distribution of potential field weights in the search space. The network inputs are the average distance to other agents, the previous weight (equivalent to the previous network output), and the (x, y) position of the potential generator. The output of the network is the new weight for that position in the search space. This

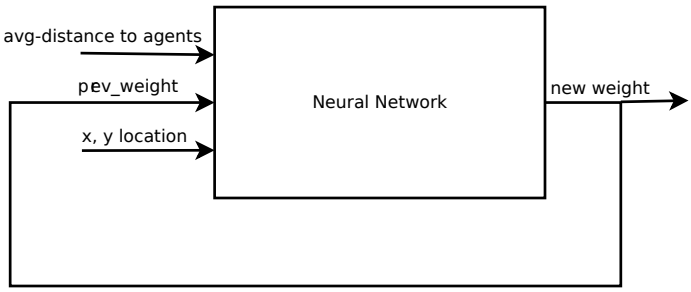

Figure 2. Potential Field Setup

setup is shown in Figure 2. The agents can then "call" this neural network to obtain the real-time weights of the search space and calculate the resulting velocity based on a potential formula.

B. Training

Neuroevolution has shown remarkable results in recent years, especially in evolving human-like behaviors such as walking, in simple robots. It has emerged as a promising alternative to reinforcement learning and is starting to show promise in multi-agent systems as well.

The basic premise of neuroevolution is to use a genetic or evolutionary algorithm to evolve a neural network. NEAT evolves both topology and weights for a feed-forward neural network simultaneously. This has been shown to be beneficial for complex problems such as walking or evolving a controller to play a game. In our case, we used a fairly simple fixed topology with 1 input layer, 1 hidden layer containing 3 nodes, and 1 output layer, and used an evolutionary algorithm to optimize the network weights. A sigmoid activation function was used to model nonlinearlity for all layers. Each individual in the evolutionary algorithm is modeled as a vector of weights (including bias bits). In subsequent generations crossover and mutations are incorporated to evolve a new population.

To record the fitness of each individual, or neural network, the overall area coverage of the agent team is recorded. The highest scoring individual would be the neural network that uses the previous field strength, average distance to agents, and an (x, y) position to determine a new potential field strength that optimizes coverage in the team. The goal is to evolve a neural network that adjusts the potential field such that agents are drawn towards unexplored regions before revisiting previous ones.

C. Functionality

The algorithm is based on an inverse r potential function. We slightly modified it to cap the maxima of the function at the weight determined by the neural network. We can then define the velocity at a given location by taking the gradient of the potential function $f(x, y)$ where k is a constant and x & y represent the x-distances and y-distances respectively.

$$f(x,y) = \frac{k}{x^2 + y^2 + 1} \tag{1}$$

$$|\nabla f(x,y)| = \left(\frac{2xk}{(x^2+y^2+1)^2}, \frac{2yk}{(x^2+y^2+1)^2}\right) \tag{2}$$

These equations define the motion of the agents according to the potential field generators and the field they generate. Each agent accesses the global map to get updated information on the state of the potential field. It then sums up the potential at its current location and calculates a resulting velocity in the direction of the gradient. The GPF will update all the generators in the search space with new weights at a fixed interval Δt. For our experiments we used 2 seconds. Additionally, to prevent erratic behavior, we limit the weight of a field generator between -100 and 100.

The agents themselves follow a set of rules to determine the safest path. First, they check if any other agents are a certain distance away. If there is anpther agent, it will immediately enable the Coulombic repulsive force and move away from the neighboring agent. If there are no obstacles, the agent will calculate its velocity according to the potential field and move in that direction.

D. Solutions to Key Issues

There are a number of key issues that needed to be addressed before the algorithm was viable. The first was keeping the agents confined in the search space. Without any special considerations, there is nothing keeping the agents from leaving the desired search space. To solve this problem, we added extremely high weight potential field generators around the search space to keep agents from crossing the boundaries. Another problem was the classic local minima problem. Because we use potential fields to guide agent motion, occasionally agents will fall into "traps", or areas with 0 potential. This is dangerous because it causes the agent to get stuck in a local minimum and thus it won't be able to keep searching. To get around this issue, we added a random vector to an agent's existing velocity to keep it moving within the search area.

IV. Implementation

We used Robot Operating System to handle the agent logic, including the behavior of the potential field, motion and modeling of the agents (with the $hector\_quadrotor$ package), and communication between the training algorithm and simulation code. We also used Gazebo for simulating the agents and modeling the environment. Gazebo blended seamlessly with ROS, providing an easy and flexible way to interface our code. The Gazebo environment is shown in Figure 3.

Additionally, we used Keras [11], a popular machine learning toolkit, with our own evolutionary algorithm implementation to run our neuroevolution approach for optimizing the PhysField algorithm. The training node works separately from the actual simulation. Our goal was for the training to serve as an "observer" that tests different neural networks and observes how they perform on the multi-agent simulation. As the observer gathers results, it will apply evolutionary operators like crossover and mutation to generate the next generation of individuals to evaluate.

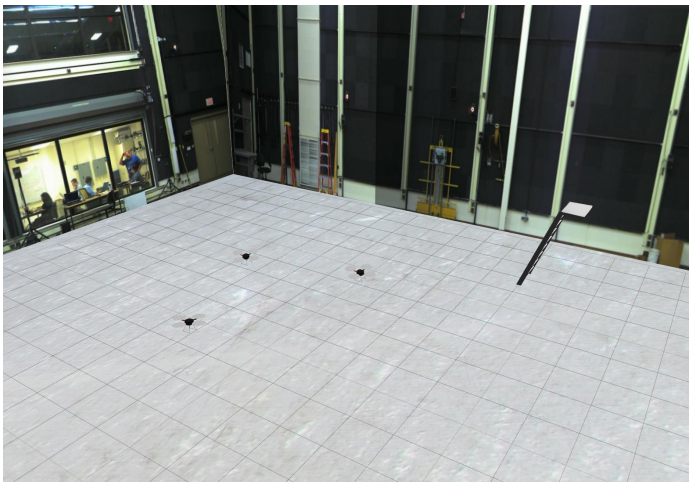

Figure 3. Model of NRL LASR's Prototyping High Bay in Gazebo, originally published in [4]

A. Distributed Simulation

For tests with experiments involving a large number of agents, we employed a Distributed Simulation framework which was first introduced in our previous paper [4]. This technique essentially allowed for several Gazebo instances to run in parallel on several machines. Each instance was responsible for keeping track of physics for a small number of agents and the framework used wired network connections between computers to merge the individual simulations together.

B. Agent Communication

We used Lightweight Communication and Marshalling (LCM), a UDP-based communication protocol, for any communication occurring globally such as position updates from the simulator. We also used LCM to communicate between machines for the distributed simulation framework.

Algorithm 1 Agent Movement Algorithm

function PHYSFIELD(P,$quad_x$, $quad_y$)
$dt \leftarrow 0.001$
$px \leftarrow 0$
$px \leftarrow 0$
for $i \leftarrow 0$ **to** $Rows$ **do**
  for $j \leftarrow 0$ **to** $Columns$ **do**
    $dx \leftarrow quad_x - i$
    $dy \leftarrow quad_y - j$
    $dP\_mag \leftarrow ((dx \hat{} 2 + dy \hat{} 2 + 1) \hat{} 2)$
    $px \leftarrow px + ((2 * k * pher\_weight)/(dP\_mag)) * dx$
    $py \leftarrow py + ((2 * k * pher\_weight)/(dP\_mag)) * dy$
  end for
end for
$vx \leftarrow (px * s + rand())$
$vy \leftarrow (py * s + rand())$
$Publish(vx, vy)$

C. Agent Control

The velocity of an individual agent is determined by the gradient of the potential function. We scale the x and y components to a safe and realistic UAV velocity. In this pseudocode, k represents the weight of the potential field generator at the location (i, j). We sum the resultant 'forces' from all the field generators in the search space to calculate the final velocity vector given to the agent. The algorithm that controls the velocity of the agents is shown in Algorithm 1. This function is called at a fixed interval by all agents in the environment. As described previously, it iterates over the grid space and calculates the distance between each cell and the agent. Using this distance and the current weight value in the cell, it will determine a force on the agent. The final force vector is converted into a velocity. The random component is added in at this point to keep agents in motion. This final vector is then published to the Gazebo simulation.

The Gazebo simulation and underlying physics engine takes in this velocity and automatically applies position updates to the agent model. Abrupt velocity changes are not possible in the simulation as the physics engine applies friction, gravity, and aerodynamics to the agent which keep the motion as smooth and realistic as possible.

V. OPTIMIZATION

In order to optimize average coverage, we evolved a neural network mapping environmental inputs to key parameters in the $PhysField$ algorithm. This allowed agents to dynamically change their behavior based on the current state of the environment. The goal of the network was to choose a set of parameter values that would maximize the area coverage potential of the team.

A. Evolving a Network

We trained a fixed topology feed-forward neural network using an evolutionary algorithm where the chromosomes are represented as weight vectors. Thus, the individuals being evaluated during each generation, represents a different network, and thus a different policy.

The environmental inputs into our network are the mean distance of a field generator to the other agents along with the previous strength and position of the generator. The output was next field strength that the generator should "produce".

Our evolutionary algorithm was trained with 50 individuals and run over 20 generations. The selectivity rate was 0.05 and the mutation rate was 0.01. Crossovers between individuals were incorporated as well with individuals being paired up using a Russian roulette approach. The fitness of each individual was evaluated by running a 2 minute simulation and recording the overall area coverage achieved by the team. The coverage by the entire team is used as the fitness value for each individual in a generation.

We are still working on improving this approach and conducting experiments to demonstrate its effectiveness in optimizing area coverage and potential target detection, especially in dynamic environments.

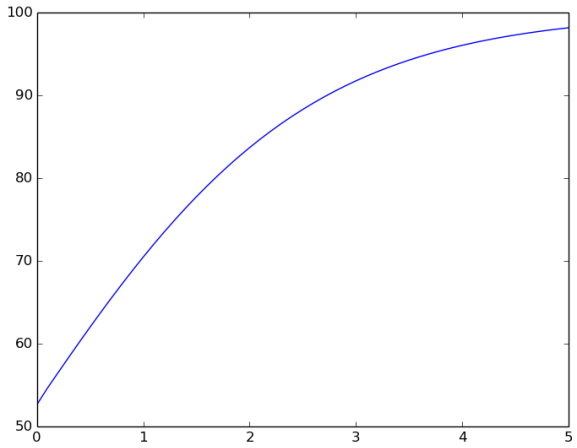

Figure 4. Coverage: 21

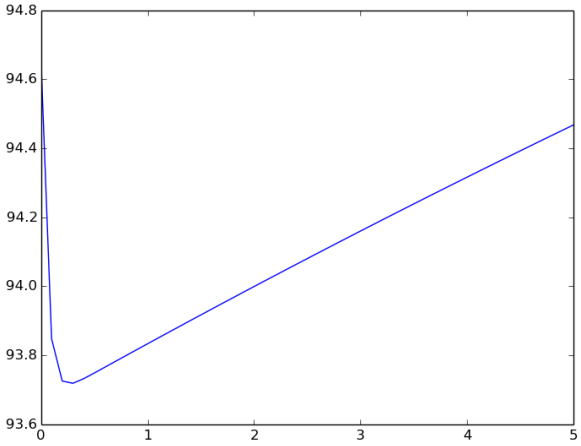

Figure 6. Coverage: 128

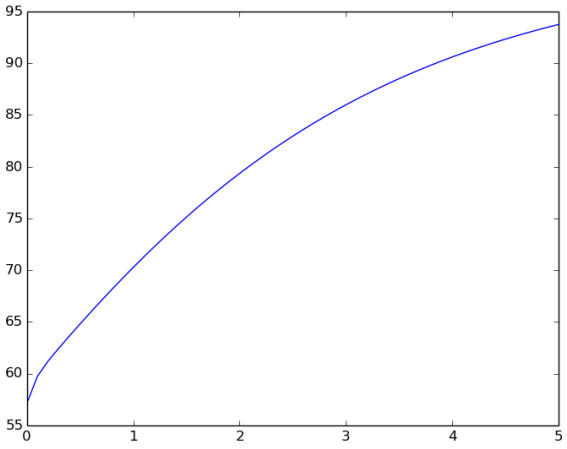

Figure 5. Coverage: 54

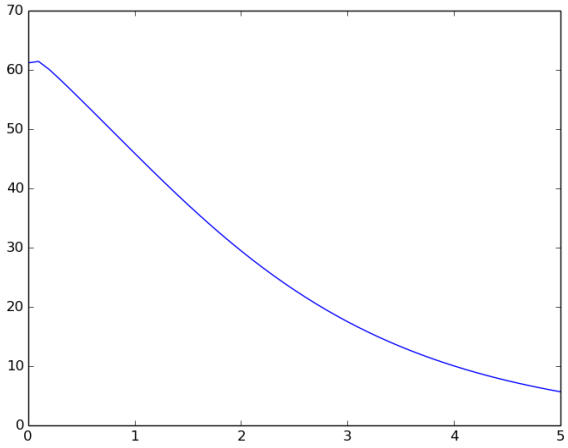

Figure 7. Coverage: 100

VI. Experiments

Due to the very lengthy training required to evolve a network with neuroevolution, in this paper we present preliminary results as well as initial neuroevolution data. The key relationship the neural network is designed to model is between agent distance and potential field weight. We tested different neural net weights, obtained a number of different types of functions modeling the relationship between average agent distance (X-axis) and the resulting weight (Y-axis) , and noted the resulting coverage obtained by the agents. Coverage in our experiments are defined as the total area (m^2) covered by all agents over a 2 minute period in a 20m x 20m search grid. In general, we found minor changes in the concavity and other features of the function caused significant changes to the overall coverage of the team, which makes training an optimal model very difficult.

Each of these graphs represent a different mapping between average agent distance and weight.

After looking at large number of networks, we found that on average weights which correspond to a downward correlation between agent distance and weight, as shown in Figures 7, 8, and 9, result in a higher overall area coverage. This makes sense as we want the repulsiveness of a certain region to decrease as an agent moves away and increase as the agent moves closer. Interestingly, we found that in some cases functions that decrease and increase at certain points like Figure 6 obtain a higher average are coverage compared to a normal decreasing graph like Figure 8.

When evolving the neural network itself, we had great difficulty determining the right set of parameters to obtain a progressive learning curve. Our best run involved 50 individuals/candidate neural networks, 20 generations, and 30 second fitness evaluations per individual. We ran this on a 5 agent Gazebo simulation and collected results on area coverage for

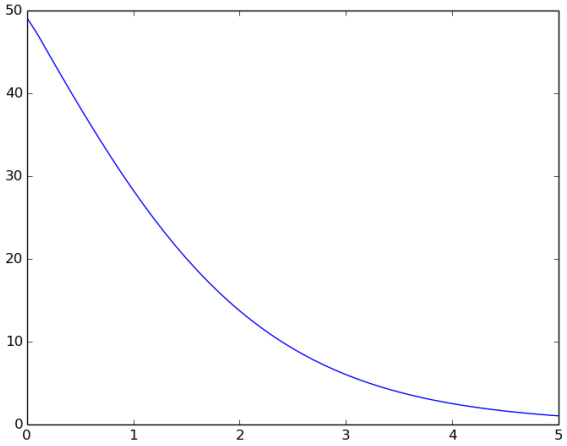

Figure 8. Coverage: 102

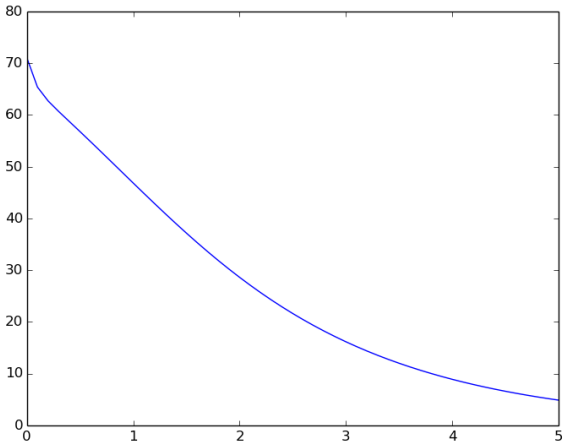

Figure 9. Coverage: 56

each candidate network every 30 seconds. The training began with an average coverage of $23.86m^2$ at generation 1 and ended with an average fitness/coverage of 31.9 m^2 over a 30 second evaluation period. We are still working on obtaining on a more optimal set of parameters for both the evolutionary algorithm as well as the potential field itself. However, we are confident in the progress of the learning process so far and will compare our approach against other area-coverage approaches in the near future.

VII. Conclusion

In this work we introduce both a new approach for achieving effective, dynamic area coverage in an unknown environment, and a learning method to optimize our approach using neuroevolution. Although our experiments are still ongoing due to the lengthy time needed for training, preliminary observations and data indicate that this research has promise and could perform extremely well after a sufficient number of generations. We believe our approach provides a unique perspective on the problem of multi-agent searching by allowing the environment itself to directly influence the motion of the agents in the space.

In the future, we will continue gathering data and statistics on our algorithm to prove that it is both effective and robust. Moreover, we will train the algorithm on different types of environments, with varying obstacle conditions in order to learn a more generalized solution to the area coverage problem using a time-varying potential field.

References

[1] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, pp. 396–404. New York, NY: Springer New York, 1990.

[2] A. Howard, M. J. Matarić, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Distributed Autonomous Robotic Systems 5* (H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, eds.), (Tokyo), pp. 299–308, Springer Japan, 2002.

[3] N. Sydney, D. A. Paley, and D. Sofge, "Physics-inspired motion planning for information-theoretic target detection using multiple aerial robots," *Autonomous Robots*, vol. 41, pp. 231–241, Jan 2017.

[4] R. Rajan, M. Otte, and D. Sofge, "Novel physicomimetic bio-inspired algorithm for search and rescue applications," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Nov 2017.

[5] J. L. Pearce, P. E. Rybski, S. A. Stoeter, and N. Papanikolopoulos, "Dispersion behaviors for a team of multiple miniature robots," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 1, pp. 1158–1163 vol.1, Sept 2003.

[6] B. Ranjbar-Sahraei, G. Weiss, and A. Nakisaee, "A multi-robot coverage approach based on stigmergic communication," in *Multiagent System Technologies* (I. J. Timm and C. Guttmann, eds.), (Berlin, Heidelberg), pp. 126–138, Springer Berlin Heidelberg, 2012.

[7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, June 2002.

[8] M. J. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, pp. 355–366, 2014.

[9] R. Miikkulainen, E. Feasley, L. Johnson, I. Karpov, P. Rajagopalan, A. Rawal, and W. Tansey, "Multiagent learning through neuroevolution," in *Advances in Computational Intelligence* (J. L. et al., ed.), vol. LNCS 7311, pp. 24–46, Berlin, Heidelberg:: Springer, 2012.

[10] J. Ericksen, M. Moses, and S. Forrest, "Automatically evolving a general controller for robot swarms," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Nov 2017.

[11] F. Chollet *et al.*, "Keras." https://keras.io, 2015.