RBE595 Proposal

Discovering Novel Swarm Behaviors through Color and Robot Capabilities

Claypool
Computer Science
Worcester Polytechnic Institute
Worcester, USA
smclaypool@wpi.edu

Enyedy
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
ajenyedy@wpi.edu

Hosea
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
rdhosea@wpi.edu

***Abstract*—Emergent behavior research in swarms has primarily been based on binary proximity sensors. This research has provided a list of emergent behaviors that can be expected from simple, evolving swarm systems. We base our research on the work outlined in Brown et al. [1], which describes the process of discovering which emergent behaviors can occur given a limited capability model of a robot. We plan to expand on their research by adding a color to each robot representing a heterogeneous 'type', and changing the binary sensor to an N-bit sensor detecting the color of the visible robots where N is the total number of colors. Our goal is to determine if, given this simple capability model, these robots can exhibit self-segregation based on their color.**

I. Introduction

Swarm systems have a variety of applications which are starting to beginning to become possible with modern technology. Tasks such as construction, exploring, search and rescue. Often, these tasks are implemented by trying to create a set of rules to control a swarm, or predicting the behavior of a swarm given a set of rules. Brown et. al. take a different approach, instead asking "given a set of *capabilities*, what are the possible emergent behaviors that can occur?". This is an important question as it provides a lower-bound for defining the required complexity for a robot in a swarm to observe a desired emergent behavior.

In nature, the emergent behavior of "self-segregation" is often important for creating complex behaviors from heterogeneous structures. For example, cockroaches exhibit preferential aggregation based on their smell [2]. Ants also exhibit a form of self-segregation when organizing their young - ant brood that require more attention or food are placed on the perimeter to allow for easy-access whereas ants that do not require attention are placed in the center. [3]. Segregation is also present at the molecular level accounting for phenomena such as organ formation.

Previous research has been done to model the segregation of robots based on role. One approach to this is to apply a "differential adhesivity" model where heterogeneous types of robots exhibit a different levels of adhesion and cohesion based on their type [4]. This model takes inspiration from the mechanisms that allow cell types to self-segregate in biological systems. Other models have been created that determine if self-segregation is possible through limited capability models and simple 'if-then-else' statements with a similar goal of defining a complexity lower-bound for self-segregation [5].

We propose a solution to this problem using a capability model similar to Brown et. al. where each robot has a differential drive motor controlled by a shallow neural network. We augment that capability model of Brown et. al. by adding a 'color' parameter to each robot, and changing the binary line-of-site sensor to an N-bit sensor reporting the color of the observed robot. Our goal is to determine whether a swarm of robots can develop self-segregation behavior based on this limited capability model.

II. Proposed Work

We propose a system using simple differential drive robots as described in Brown et al. [1] and shown in Fig. 2, but with the added capabilities of sensing colors and defining their colors. This color field for each robot will be implemented as a 1-hot encoded vector. Initially, this color field will simply be a 2 bit vector as we will initially test just two groups of robots. The sensor will be adjusted to report an N-bit vector corresponding to the color of the robot seen. In the case that not robot is in the line of sight of the sensor, it will report a vector of all zeroes. To simplify the neural network, we will have collapse the input into 2 bits regardless of the number of classes in the simulation. The first bit will be a 1 if a robot of the same type is detected by the sensor, and the second bit will be a 1 if a robot of the second type is detected by the sensor. This new configuration can be seen in Figure 1

III. Proposed Experiments and Expected Outcomes

We will follow a similar construction to Brown et. al. to test our hypothesis. We will perform an evolutionary search over the possible neural network states and prioritize novel behaviors, as described in Figure 3.

Rather than clustering the observed statistics through unsupervised methods such as K-medoids, we will rank our observed behaviors using a measure of segregation, such as the intersection of convex hulls based on Santos et. al. [6]. The intersection of convex hulls summarizes the convex shape of each cluster of robots as a convex hull, and measures the

Identify applicable funding agency here. If none, delete this.

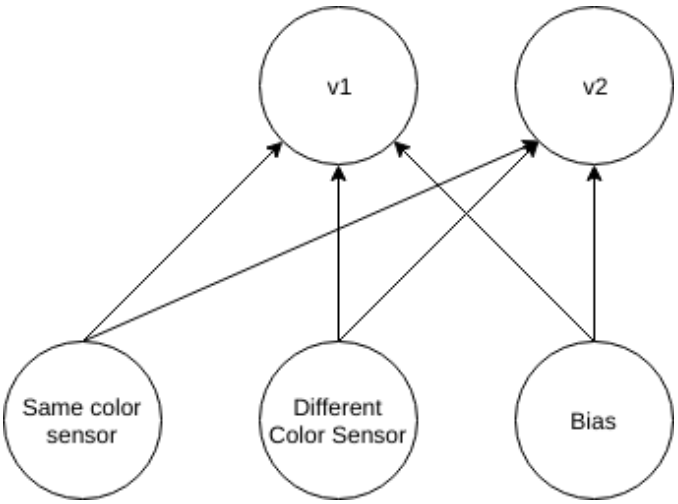
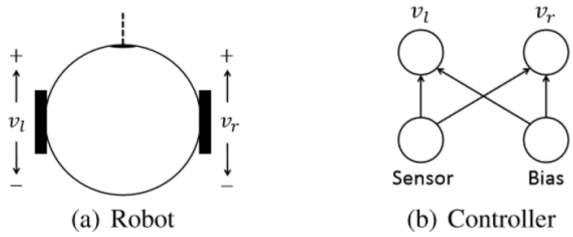


Fig. 1. Neural Network Controller



(a) Robot (b) Controller

Fig. 2. Robot model from [1]

overlap between the different classes. This means that, if the classes are each perfectly separated, there will be no overlap between the nulls of each class, so the overlap will be zero.

Once we can rank the models by their separation, we will visually inspect the top results to categorize their behavior and separation mechanisms.

IV. Weekly Schedule

To complete our objectives, we will have weekly meetings on both Tuesdays and Wednesdays at 6pm. At these meetings we will convene to review our progress over the past week and plan our work for the upcoming week. We set up a Gantt chart to plan our weekly progress, shown in Figure 4

Algorithm 1 `NovelBehaviorDiscovery`

Require: environment $\mathscr{E}$, capability model $\mathscr{C}$, and controller model $\mathscr{W}$

```
P ← InitializePolicies(W(C))                    ▷ Generate initial population P_0
archive ← InitializeArchive(P)
while stopping criterion not met do
    for each policy p_i in population P do
        f_i ← ExtractFeatures(p_i, E)           ▷ extract features by evaluating policy
        n_i(t) ← Novelty(f_i, archive)          ▷ evaluate novelty
        if addToArchive(⟨p_i, f_i(t)⟩) then
            archive.add(⟨p_i, f_i⟩)             ▷ store individual in novelty archive
        end if
    end for
    P ← Update(P, f)        ▷ update population using a GA with fitness replaced by novelty
end while
K ← Cluster(archive)        ▷ Cluster on archive and return K representative behaviors
return K                    ▷ Return cluster representatives as taxonomy
```

Fig. 3. Novelty search as seen in [1]

Task	Difficulty	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Develop Problem Statement	1 of 5							
Set up Robots in ARGoS	3 of 5							
Set up Neural Network	4 of 5							
Write Buzz movement code	3.5 of 5							
Testing	3 of 5							
Analysis of Results	5 of 5							
Final Presentation	3 of 5							
Final Report	4 of 5							

Legend
Overdue
Expected
Done
In Progress

Fig. 4. Gantt Chart

V. Implementation

A. Individual Argos Simulations

We use Argos to virtually test different neural network weight configurations, as described in Figure 1. We create a BUZZ template file with the weights defined as the top as `W1 = {W1}` etc. This allows our python program (discussed in Section V-B) to replace these values for each population. For exaple, if the current population has the weights `[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]`, then the first weight in this file will be replaced with 0.1, the second weight will be replaced with 0.3 and so on. Argos will then be able to use this newly created buzz controller to control the behavior for each robot.

The ''neural network'' code is implemented with if statements. If a bot detects a robot in front of it with the *same* group type, it will set the wheel speed of its left wheel to $weight_1 \times K$ and the right wheel to $weight_2 \times K$ where K is a constant scaling factor. Similarly, if the robot sees a robot of a *different* type directly in front of it, it will set the left wheel speed to $weight_3 \times K$ and the right to $weight_4 \times K$. Finally, if no robot is seen, then the current bot will set the left wheel to $weight_5 \times K$ and the right to $weight_6 \times K$. This implementation is identical to the vectorized version discussed in the introduction.

Each of the simulations will run for 5,000 time steps. At each time step, we record the position and wheel speeds of each bot for the current iteration. Further processing of these values is done in in the python code discussed in Section V-B.

B. Novelty Search

Our novelty search implementation closely that of Brown et. al. [1]. We implement this algorithm, previously shown in Figure 3, in Listing 1. This follows an identical layout - first, a seed population is chosen with uniform weights (although we did experiment with an intentionally ''good'' seed population which we believed would already shown some group segregation). On each iteration, the current most-fit population is permuted. This means that each weight is either increased or decreased and a new population is created from this mutation(discussed further in Section V-C). Then, each of these new populations is simulated in Argos and the locations at each time-step are recorded.

After each simulation, we create a *behavior vector* similar to Brown et. al.5. This is to summarize the performance of

Table 1 Behavior vector feature descriptions

Name	Equation	Name	Equation
average speed	$\frac{1}{N}\sum_{i=1}^{N}\|v_i\|_2$	scatter	$\frac{1}{R^2 \cdot N}\sum_{i=1}^{N}\|x_i-\mu\|^2$
ang. momentum	$\frac{1}{R \cdot N}\sum_{i=1}^{N}(v_i \times (x_i-\mu))$	group rotation	$\frac{1}{N}\sum_{i=1}^{N}\left(v_i \times \frac{x_i-\mu}{\|x_i-\mu\|}\right)$
radial variance	$\frac{1}{R^2 \cdot N}\sum_{i=1}^{N}\left(\|x_i-\mu\|-\frac{1}{N}\sum_{i=1}^{N}\|x_i-\mu\|\right)^2$		

Fig. 5. Behavior Vector, Table 1 in Brown et. al. [1]

each population so that we can evaluate the novelty of the parameter weights.

Listing 1. Search algorithm (Python)

```
# Details at
# github.com/SaahilClaypool/
    swarm_novelty_search/blob/master/
    search.py
def search():
    print("iteration, population, weights
        , segregation")
    seed = Observation([0.5, 0.5, 0.5,
        .5, .5, .5])
    population = [seed]
    archive = []
    stop = False
    it = 0
    max_it = 100
    while(max_it != 0):
        it += 1
        max_pop = -1
        for idx, p in enumerate(
            population):
            _features = p.getMeasures()
            global PRIOR_WEIGHTS
            PRIOR_WEIGHTS.append(p.
                weights)
            if (shouldAddToArchive(p, p,
                archive)):
                archive.append(p)

        population = updatePopulation(
            population, archive)

        max_it -= 1

    seg = most_segregated(archive)
    print("most segregated\n", seg)
```

C. Time Complexity

The obstacle in our implementation is the time complexity involved in the evoluationary search. By increasing the number of weights in our model from 4 to 6, we increased the search space from 4 dimmensions to 6. In our original implemntation, we were fully permutating each weight vector. That is, for each population we would test *all possible combinations* of increased or decreasing the weights in the vector. For example, if the weights of a two dimmensional vector were `[.1, .2]`, and the step size[1], then a single permutation of this population would create the following new populations

- `[.0, .1]`
- `[.0, .2]`
- `[.0, .3]`
- `[.1, .1]`
- `[.1, .3]`
- `[.2, .1]`
- `[.2, .2]`
- `[.2, .3]`

Which is $3^2 - 1 = 8$ new populations. This is because each value can either increase, decrease, or stay the same. So, for each new weight added, three new populations can be generated. This meant that when we created a 6-weight neural network, each population could generate a potential of $3^6 - 1$ populations, or 728 new populations. Each population takes roughly 10 to 30 seconds to simulate in our implementation, causing each generation of the evolutionary search to take roughly 6 hours on a relatively powerful machine.

Thus, we needed to adjust our implementation to reduce this time complexity. We did this with two different approaches. Our first approach was to return to a 4 weight model, but change the semantics of our simulation such that weights 1 and 2 were used only if a robot of the *same* time was visible directly infront of the current robot. Otherwise, the other two weights would be used whether there was no robot infront of the current robot, or a robot of the other type. This made the bots effectively blind to bots of the wrong type, but from our experimenting, it seems that well-segregated swarms would often just turn away from swarms of the oher type. Thus, the swarms of the other type would not be in 'line of sight' for more than a single time step, and thus the weights controlling the behavior when a bot of the other type is visible were rarely used. So, we combined these weights with the weights controlling the behavior when no bot is visible.

Our second heuristic was to *not* compute all combinations of the weights for the current population, but rather just increase each weight individually to create new populations. This reduces the number of new populations generated each generation from $3^W - 1$ to $3 \times W - 1$. In the case of our 6-weight example, this reduces the number of permutations from 728 to just 17. This makes each generation less effective, but made each iteration exponentially faster. This allowed us to run the same experiment with all 6 weights being evaluated. In the results, we will discuss the differences discovered by these two approaches.

1) Fitness Search vs Novelty Search: Why do we think fitness is a better fit for this problem

[1]We use a step size of .1, and bound each weight between 0 and 1.

D. Segregation Metrics

What does it mean to be segregated?

REFERENCES

[1] D. S. "Brown, R. Turner, O. Hennigh, and S. Loscalzo, *Discovery and Exploration of Novel Swarm Behaviors Given Limited Robot Capabilities*, pp. 447–460. Cham: Springer International Publishing, 2018.

[2] J.-M. Ame, C. Rivault, and J.-L. Deneubourg, "Cockroach aggregation based on strain odour recognition," *Animal behaviour*, vol. 68, no. 4, pp. 793–801, 2004.

[3] A. B. Sendova-Franks, S. R. Scholes, N. R. Franks, and C. Melhuish, "Brood sorting by ants: two phases and differential diffusion," *Animal Behaviour*, vol. 68, no. 5, pp. 1095–1106, 2004.

[4] M. Kumar, D. P. Garg, and V. Kumar, "Segregation of heterogeneous units in a swarm of robotic agents," *IEEE transactions on automatic control*, vol. 55, no. 3, pp. 743–748, 2010.

[5] P. Mitrano, J. Burklund, M. Giancola, and C. Pinciroli, "A minimalistic approach to segregation in robot swarms," *arXiv preprint arXiv:1901.10423*, 2019.

[6] V. Graciano, S. Luciano, C. Pimenta, and L. Chaimowicz, "Segregation of multiple heterogeneous units in a robotic swarm,"