Evolving a Repertoire of Controllers for a Multi-Function Swarm

Sondre A. Engebråten[1,2], Jonas Moen[1,2], Oleg Yakimenko[3], and Kyrre Glette[1,2]

[1] Norwegian Defence Research Establishment, P.O. Box 25, 2027 Kjeller, Norway.
Sondre.Engebraten@ffi.no
[2] University of Oslo, P.O. Box 1080, Blindern, 0316 Oslo, Norway
[3] Naval Postgraduate School, 1 University Circle, Monterey, CA 93943, USA

Abstract. Automated design of swarm behaviors with a top-down approach is a challenging research question that has not yet been fully addressed in the robotic swarm literature. This paper seeks to explore the possibility of using an evolutionary algorithm to evolve, rather than hand code, a wide repertoire of behavior primitives enabling more effective control of a large group or swarm of unmanned systems. We use the MAP-elites algorithm to generate a repertoire of controllers with varying abilities and behaviors allowing the swarm to adapt to user-defined preferences by selection of a new appropriate controller. To test the proposed method we examine two example applications: perimeter surveillance and network creation. Perimeter surveillance require agents to explore, while network creation requires them to disperse without losing connectivity. These are distinct application that have drastically different requirements on agent behavior, and are a good benchmark for our swarm controller optimization framework. We show a performance comparison between a simple weighted controller and a parametric controller. Evolving controllers allows for specifying desired behaviors top-down, in terms of objectives to solve, rather than bottom-up.

Keywords: Swarm UAVs, MAP-elites, Evolutionary Robotics, Multi-function

1 Introduction

In a robotic swarm system, a large number of agents interact in order to conduct missions and solve specific tasks. Some swarm systems require collaboration, as a single agent will not be able to complete the task alone [1]. Other swarm systems might consider agents with competing interests or goals [2]. Yet, others might consider how heterogeneous teams of agents can collaborate in order to make use of the strengths of each agent [3].

For swarm systems it is common that the most interesting part of the behavior happens on a macro level, i.e. considering the swarm as one single system. For example; is the swarm, as a whole, able to solve the given task? Is the swarm able to optimize agent use for efficiency? This is different from a micro level, considering the individual behavior of each agent or platform. Swarm behaviors

DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

require some form of rules or controller for each individual agent. This can be a neural network [4], or a set of rules [5], or even a hybrid of the two [6].

An unsolved problem in the swarm literature is the top-down design of swarm behaviors or controllers, or automatic controller synthesis. This is a problem, as the high-level behavior is very dependent on the low level behavior, but the relation is not easily predictable. Evolutionary robotics attempts to address this issue by evolving controllers, rather than designing them by hand [7]. We propose to expand on this idea by evolving not just a single controller, but a set of controllers for controlling a swarm. Previous works have evolved sets or repertoires of controllers for robotics application [8], but these have been focused on single robot application. We propose to expand this to swarms of robots, in order to generate a varied set of primitives for swarm control.

We are also investigating the potential to tackle multiple tasks or application simultaneously. Imagine a swarm, not limited to a singular task, but solving multiple tasks such as perimeter surveillance and communication network. This could be approached from a resource or task allocation perspective [9], but this paper chooses to view this as a problem of generating a suitable controller with an internal notion of priority between the tasks. This is related to multi-modal learning, multi-task learning and multi-function learning. All of these attempt to solve multiple tasks at once, but with slight variations. For example, in multi-modal behavior learning [10, 11], the challenge is to evolve a game-playing agent for multiple sequential tasks. The agent has to learn both how to solve the individual tasks and when to change from one behavior to another in order to succeed. This is significantly more challenging, as it also requires handling potentially conflicting knowledge in the controller. Further, this can also be related to the challenge of learning two related, but potentially conflicting tasks. For this approach, modularity in the controller has been shown to improve performance [12]. In this paper the authors propose to explore whether it is possible to evolve controllers for multi-functional robotic swarms where the tasks are not sequential, but rather concurrent goals that all have to be satisfied at the same time.

Two applications will be considered in this paper: perimeter surveillance and network creation. The task of perimeter surveillance has been explored in previous work [13], and so has the use of swarms to maintain communication networks [14]. The challenge is now to consider both of them at once. Both perimeter surveillance and network creation have their own specific requirements in terms of movement and behavior. As such, it is expected that it will be hard for a single controller to perform well on both tasks.

For this paper, on the evolution of multi-function swarms, we propose two controller types: a weight-based controller and a parametric controller. The weight based controller is, as the name suggests, a simple weighting of input components or forces. A simple weighting of input forces is similar to the motor schema [15]. The parametric controller has a more complicated and powerful controller description, capable of describing a wider array of controller behaviors. Our controllers were inspired by the use of artificial potential fields and

artificial physics [16]. Artificial potential fields have previously been employed in collision avoidance [17–19]. Using artificial potential fields for collision avoidance can be viewed as a problem of weighting a number of independent forces from objects of potential collision risk. Essentially, this becomes a weighted forces problem where the net force from all the potential collision risks should point in a collision-free direction. In many cases this works well, assuming the environment is not too cluttered or the collision risks too many. For the controllers examined in this paper, we propose a similar approach, using several contributions from interesting objects and weighting these to provide a controller for a robotic swarm. In essence, each application provides some input to the controller and have certain requirements on the behavior or movement of the platform for optimal performance. Concurrently handling requirements from multiple applications is a challenging problem - one not yet fully explored in the literature.

Using a single controller for multiple tasks could allow for development of more complex behaviors. A hand-coded strategy may be able to solve either of the given tasks alone, but describing how to handle complex interaction between the tasks and the requirements that each task brings to the behavior may be too hard for conventional methods. For this reason, we chose to explore the option of evolving controllers using the MAP-elites algorithm [8, 20]. It is also important to realize that this is different from the work on hybrid controllers, as we chose to approach the controller design as a single monolithic problem rather than trying to solve it through decomposition [6, 21, 22]. Using MAP-elites will lessen the need for bootstrapping or incremental evolution, which is often required in evolutionary robotics [23, 24].

Both the weighted and parametric swarm controllers are optimized on the tasks of perimeter surveillance and communication network creation, using MAP-elites to generate a large repertoire of possible solution candidates. This is related to multi-objective optimization where, rather than a single best solution, a good approximation of a Pareto front is sought. The advantage of this approach is that not only one, but many interesting solutions may be examined. This may also provide a better insight into the problem and contributing factors to the generated solutions by illuminating the search space [20]. Both the weighted and parametric swarm controllers can be considered direct controller architectures, which have been shown to have higher performance than indirect encodings [25].

2 Simulator Setup and Swarm Model

Each platform or agent is modeled as a point mass with independent limits on acceleration and velocity. This makes it possible to simulate a wide variety of platform types including non-holonomic ground and aerial vehicles. While modeling platforms as a point mass is a major simplification, compared to real-world dynamics, it is suitable for these experiments. Here, we wish to examine the dynamics of the swarm as a whole, and the high-level behaviors generated. We are less interested in the exact motion of individual agents; as such, this is a viable model for the swarm. The bounds on acceleration and velocity for

these experiments are $1m/s^2$ and $10m/s$ respectively. The agents operate in a 1000x1000m area. Each platform is controlled by setting a velocity setpoint $\mathbf{v}_{sp}$, this can be considered the outer control loop. The goal of the inner-loop controller is to change the acceleration to match the velocity setpoint, i.e minimize the norm $||\mathbf{v} - \mathbf{v}_{sp}||$. This is done by a simple proportional controller. The cascaded control architecture allows setting a position setpoint as well, which is then be transferred to a velocity setpoint. For interaction between platforms or agents we assume that each agent is able to localize itself in a global and shared coordinate frame. Our controller require the agents to be able to communicate their position to neighbors. This effectively gives each agent accurate direction and heading to neighboring agents within communication range.

The details can be found in the full source code available online [4]

3 Velocity Setpoint Controllers

Our two proposed velocity setpoint controllers receive 4 inputs, from which a single controller output is generated. The controller inputs are:

1. Direction and distance to closest neighbor
2. Direction and distance to second closest neighbor
3. Direction and distance to third closest neighbor
4. Direction to the least-visited neighboring field (square)

To find the least visited square surrounding the agent, a histogram over visits to each area is collected amongst the agents. This is based on a Moore neighborhood model, i.e least of the eight surrounding squares. If two or more squares have the same visitation count, one is chosen at random. The current implementation uses a shared blackboard structure, but in principle, there is nothing that requires this information to be globally known to all agents. A local history of visitations could be used in place of this structure.

The 4 controller inputs are coded as a difference vector $\mathbf{F}_i$ $(i = 1, 2, 3, 4)$, or relative vector to the agent's current position. For example; if the input is directly returned as the output, the net result is that the agent moves towards the neighboring agent or the least visited field surrounding it. This is different from other similar works, where sensors are directional, covering a slice of the agents total view.

Each of the inputs are weighted - a process which vary depending on the type of controller, and accumulated/summed to generate a single output direction. This direction can also be is the velocity setpoint for the agent. As such, the weights directly influence the speed of the agent at any given time.

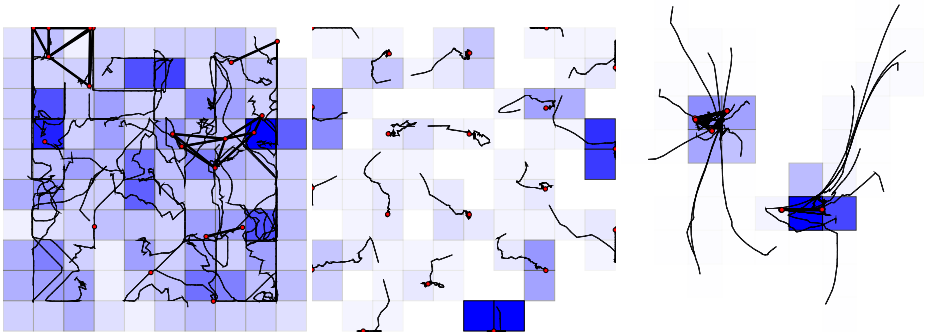
[4] `https://github.com/ForsvaretsForskningsinstitutt/Paper-towards-multi-function-swarm`

3.1 Weighted controllers

Using the forces defined in Section 3, a simple controller could be generated by defining the controller output as a weighted sum of the inputs. This is inspired by artificial potential fields [17–19]. This allows the generation of a varied set of behaviors such as clustering, avoidance, gather all, and more. This requires a single weight parameter for each given force resulting in a total of 4 parameters for a single controller.

$$\mathbf{v}_{sp} = \frac{1}{4}\sum_{i}^{N} \frac{\mathbf{F}_i}{||\mathbf{F}_i||} * w_i \tag{1}$$

$\mathbf{F}_i$ is a relative vector between the agent and the sensed object or the force direction. w_i is the weight for a given force vector. $\mathbf{v}_{sp}$ is the combined controller output, given all the input forces, which is fed to the inner-loop controller. As can be seen from Eq. 1, this controller does not use the distance-part of the input, and relies solely on the input directions. As such, the weights w_i directly influences the velocity of an agent at any given time.



Exploration Repulsion Clustering

Fig. 1. Example of hand designed controllers using the weighted controller structure. Darker squares have been more frequently visited. From left to right, exploration, repulsion and clustering type behavior. Based on manual manipulation of the weight controller, it is hard to get the right tradeoffs to get controllers balancing these traits. Videos of these behaviors can be found at `https://www.youtube.com/playlist?list=PL18bqX3rX5tQN2HKdHSCna8ysbX9lUeSM`

The weighted controller is able to define simple behaviors such as attraction and repulsion. Hand coded examples of this can be seen in Fig. 1. The weights used to generate these behaviors are [-0.5,0,0,1] for exploration, [-1,0,0,0] for repulsion and [0,0,1,0] for clustering. The first three of the weights are used to weigh the contribution from the three nearest neighbors, in order of distance. The final weight is used to specify the attraction or repulsion towards the least

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

frequently visited square of the eight that surrounds the agent. The exploration controller with weights [-0.5,0,0,1] attempts to move towards the least visited area, while keeping the closest neighbor agent away.

3.2 Parametric controllers

A potential issue with a simple weighted controller is the inability to describe a behavior keeping a given distance from another agent. This type of behavior might be very useful, e.g. when robots are to created and maintain a stationary grid. To address this problem we propose to use a parametric function instead of a simple weight.

$$g_i(d_i) = -t_i * 2 * (d_i - c_i) * e^{-(d_i - c_i)^2/\sigma_i^2} \tag{2}$$

$$a_i(d_i) = k_i * \left(\frac{2}{1 + e^{-(d_i - c_i)/\sigma_i}} - 1 \right) \tag{3}$$

$$w_{p,i}(d_i) = a_i(d_i) + g_i(d_i) \tag{4}$$

$$\mathbf{v}_{sp} = \frac{1}{4} \sum_{i}^{4} \frac{\mathbf{F}_i}{||\mathbf{F}_i||} * w_p(d_i) \tag{5}$$

The parametric weight function $w_{p,i}(d_i)$ consists of two components, $a_i(d_i)$ and $g_i(d_i)$. This function gives a weight that depends on the distance d_i to the sensed object. In other words, the contribution of the force to the velocity of the agent can be made to vary with the distance to the sensed object. $a_i(d_i)$ is responsible for static repulsion/attraction forces, while $g_i(d_i)$ account for distance holding at a predefined distance. Fig. 2 is an example plot of $w_p(d_i)$.

$g_i(d_i)$ or the gravity well enables holding a distance c_i to an object. This is based on a normal distribution with a mean of c_i. The outputs to the platform consists of a velocity setpoint, rather than a position setpoint; as such we use the derivative of the normal distribution as part of our parametric function. In order to approach a distance and stop, we need a function with a variable zero crossing point, which is accomplished through c_i. In addition, the parameters σ_i and t_i allow for the adjustment of the width or range of the force, and the strength of the force respectively. t_i allows for both repulsive and attractive behaviors around center c_i by inverting the sign of the function.

$a_i(d_i)$ contributes a fixed attractive or repulsive force across a greater area. This allows for pure attraction or repulsion behaviors, which can be useful for collision avoidance or exploration. This component is based on a Sigmoid activation function and exhibits a jump from $-k_i$ to k_i around the center-point c_i. The transition between the two values is smooth, which is important for stability once the agent is close to the center point c_i.

Together, $a_i(d_i)$ and $g_i(d_i)$ make the weight $w_{p,i}(d_i)$ for a given input i in the parametric controller - we call this the Sigmoid-well function (Fig. 2). It consists of two parts; a Sigmoid for general attraction/repulsion from objects and a

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

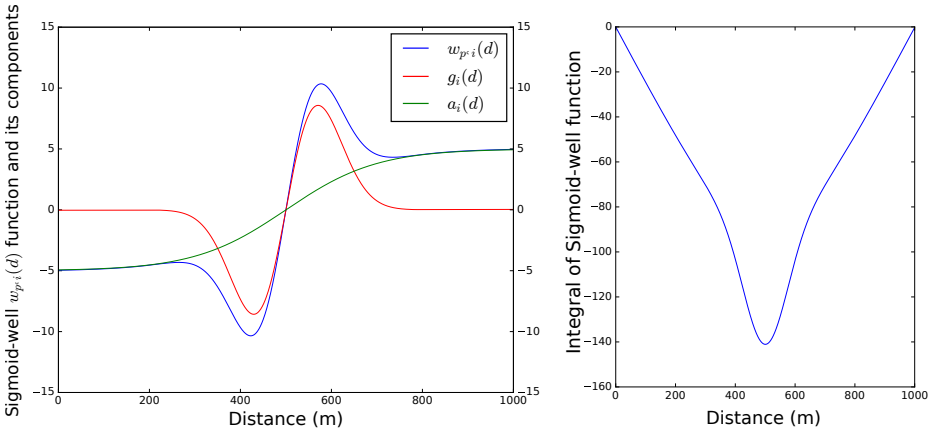


Fig. 2. The Sigmoid-well function (Eq. 4) is shown in the left part of the figure. The green line represents the Sigmoid component $a_i(d_i)$. The red line is the gravity well component $g_i(d_i)$. Added together they form the blue line: the Sigmoid-well function $w_{p,i}(d_i)$. The right part of the figure depicts the integral of the Sigmoid-well function $w_{p,i}(d_i)$, which has a clear strong attraction (minimum) around a center $c_i = 500.0$ The remaining parameters are $t_i = -0.1$, $k_i = 5.0$ and $\sigma_i = 100.0$.

gravity well component $g_i(d_i)$ for keeping a given distance. This function combines the weights from the simple controller, through the scaling of the Sigmoid function (parameter k_i) and the ability to describe hold at a distance c_i through a gravity well component. Furthermore, it is easily optimizable and described by 4 real-coded values for each force contribution, or 16 for our complete controller with 4 input forces. Fig. 2 show the individual contributions given distance, and the combined resulting weighting $w_{p,i}(d_i)$

Table 1. Parameters for hand coded parametric example controllers

	k_i	c_i	σ_i	t_i
Exploration	[-2,0,0,3]	[150,150,150,1000]	[100, 100, 100, 100]	[0,0,0,0]
Combination	[-2,1,0,3]	[150,150,150,1000]	[100, 100, 100, 100]	[-0.1,-0.1,-0.1,0]
Network	[-1,1,0,0]	[150,150,150,1000]	[100, 100, 100, 100]	[-0.1,-0.1,-0.1,0]

This controller allows for defining a "hold at a distance" behavior (Fig. 3). Parameters for controllers in Fig. 3, can be seen in Tab. 1. Both the network focused behavior and the controller featuring a combination of network and exploration focus exhibit clear lattice structure. This is made possible by the parametric controller architecture. Onwards we consider k_i, t_i, c_i and σ_i as vectors $\mathbf{k}$, $\mathbf{t}$, $\mathbf{c}$ and $\boldsymbol{\sigma}$ - each a vector of 4 real-coded values.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

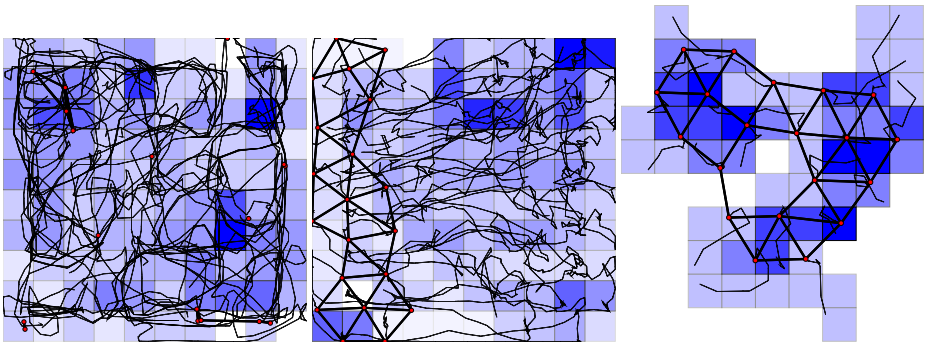


Fig. 3. Example hand designed controllers using the parametric controller structure. From left to right are examples of exploration focus (left), combination of exploration and network creation (middle) and a controller that results in a static network (right). Videos of these behaviors can be found at `https://www.youtube.com/playlist?list=PL18bqX3rX5tQN2HKdHSCna8ysbX9lUeSM`

4 Methods

4.1 Fitness and characteristics

For these experiments, two behavioral characteristics were used: exploration median and network coverage. First, exploration median is calculated by accumulating all distinct visitations to each square/bin in the area during the simulation. Then the median across all the bins in the search area is calculated. The median is normalized for agent count, speed and size of the grid. This gives a measure that is independent of the number of agents and the simulator setup. This can also be considered a percentage of the maximal median possible to achieve, given a number of agents.

The second metric, network coverage, is calculated by first finding the largest group of connected agents. In this context, connectivity is defined by a simple range test, which for these experiments was defined as within a fixed range of 200m. The behaviors adapt to this distance through the evolution of the controller, as such the controllers evolved are specific to this connectivity distance. Once the largest set of connected agents is determined, the area their communication radius cover is used as a characteristic dimension. This number is also normalized for the greatest possible coverage a swarm of agents can achieve. Since some of the agents will cover overlapping areas, the maximum area possible to cover is scaled by a factor of 0.5.

Finally, fitness is a metric related to the movement or energy use for the swarm as a whole. Fitness is defined as:

$$\text{fitness} = \frac{2}{1+b} \tag{6}$$

DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

Where b is proportional to an average agent's speed during a simulation run. Without loss of generality, this quantity can be approximated deterministically based on the parameters for the controller. Specifically, for the weighted controller b is the norm of the weight vector ($||\mathbf{w}||$), and for the parametric controller b is the sum of norm t_i and norm k_i ($||\mathbf{k}|| + ||\mathbf{t}||$).

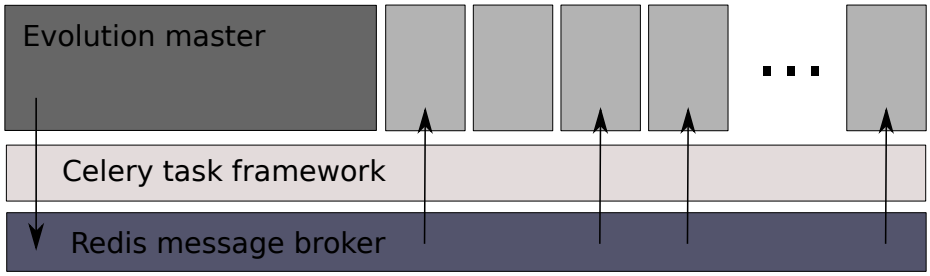


Fig. 4. Overview of our evolutionary framework in Python using Celery and Redis message broker. The evolution master generates the candidate controllers to be tested and maintains the repertoire of controllers, and the worker threads evaluate candidate solutions and return a log of the experiment for review by the master.

4.2 MAP-elites

Both controllers are defined as a finite sequence of real-coded values. For the weighted controller, a vector of 4 real-coded values is used. For the parametric controller, a vector of 16 real-coded values is used. MAP-elites uses only a mutation operator for permutation of individuals; this is implemented as an additive Gaussian variation with mean of 0 for all parameters. The standard deviation of the Gaussian mutation is 10.0 for the weighted controller, and the weights are clamped between -100.0 and 100.0.

For the parametric controller, the real-coded genome has a different interpretation. We have 16 real-coded values. For each input force there is a scaling weight for the Sigmoid function (k_i), scaling weight for the gravity well strength (t_i), center distance for the gravity well (c_i) and spread or range for the gravity well (σ_i). These form the vectors $\mathbf{k}$, $\mathbf{t}$, $\mathbf{c}$ and $\boldsymbol{\sigma}$ - each a vector of 4 real-coded values. The range and mutation used for the individual elements of these vectors can be found in Tab. 2

This research utilizes a parallel version of MAP-elites. The original MAP-elites algorithm specifies that, at each iteration a single individual is selected, mutated, evaluated and then placed back in the appropriate cell. The parallel version is similar, but works on a batch of individuals, selecting 200 individuals at a time, mutating, evaluating and placing them back in their cells. Each individual in the repertoire is one potential controller. The controller is evaluated by simulating a swarm, where each agent is controlled by the given controller.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

Table 2. Range and mutation parameters for the parametric controller

Param.	Min	Max	Mut. std. dev.
k	-100.	100.	10.0
t	-1.0	1.0	0.1
c	100.0	1000.0	100.0
$\boldsymbol{\sigma}$	0.0	100.0	10.0

Our experiments use an initial population (generated randomly) of 200 individuals and up to 200 generations or batches of 200 individuals per experiment. This evaluates some 40200 possible solutions. We run our experiments using task parallelism, this is briefly outlined in Fig. 4.

5 Results

5.1 Weighted controller experiments

Preliminary experiments used only a single simulation per controller; which was insufficient and caused significant noise in the final repertoire. For this reason we simulate each controller five times, varying the starting positions and random seed, in order to get a better estimate of the controller performance. Fig. 5 shows the resulting repertoire after 1, 10, 100 and 200 epochs (batches of evaluations). In this figure, the resultant exploration-median values are placed in 10 bins (shown along the vertical axis), and network-coverage values - in 100 bins (shown along the horizontal axis). Bins for which MAP-elites found a controller has a gradient color from white to dark blue, where dark blue indicates a high fitness value. Bins for which no solution was found are black.

Some of the evolved behaviors from the evolution are shown in Fig. 6. Evolved behaviors that explore a lot, without providing a large network coverage, typically cluster most or all the agents into groups that traverse the area. This can be seen in the left part of the figure. Behaviors that are more balanced between the applications, keep the agents further apart while exploring a small part of their surroundings. The controllers with the greatest network coverage accomplish this by standing still and balancing clustering and repulsion. It should be noted that this latter swarm behavior poses quite a challenge for the weighted controller, as there is no explicit support for holding a distance to another agent.

5.2 Parametric controller experiments

The result for the same set of experiment for the parametric controller are shown in Fig. 7. Compared to Fig. 5 there is less noise (almost none) and a more clearly defined structure to this repertoire. This is due to the fact that the parametric controller is more reliable and consistent, compared to the weighted controller.

Similar to Fig. 6, Fig. 8 shows some examples of parametric controllers generated by our approach. These are more structured, compared to the weighted controllers and feature partial lattice patterns. It is important to note that the controller has no incentive to generate stable or highly structured behaviors. A

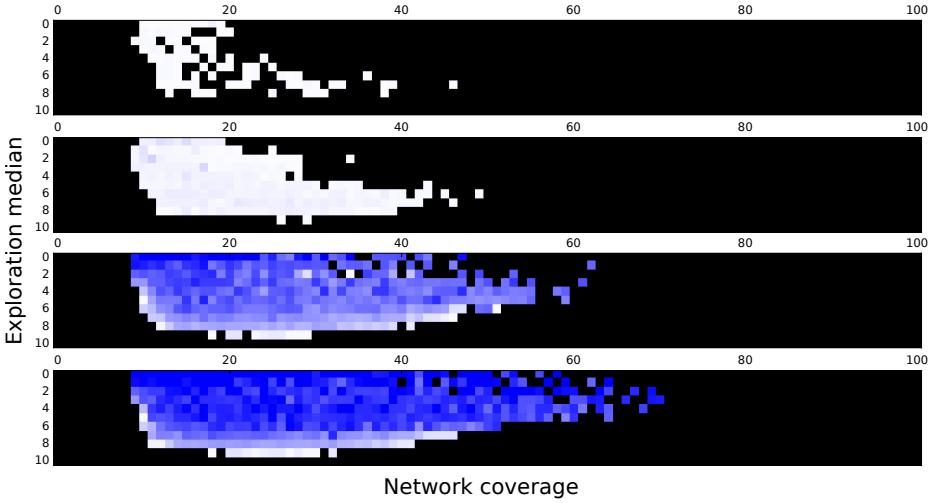


Fig. 5. Evolving a weighted controller. Fig. shows epoch 1, 10, 100 and 200 with 5 evaluations per controller

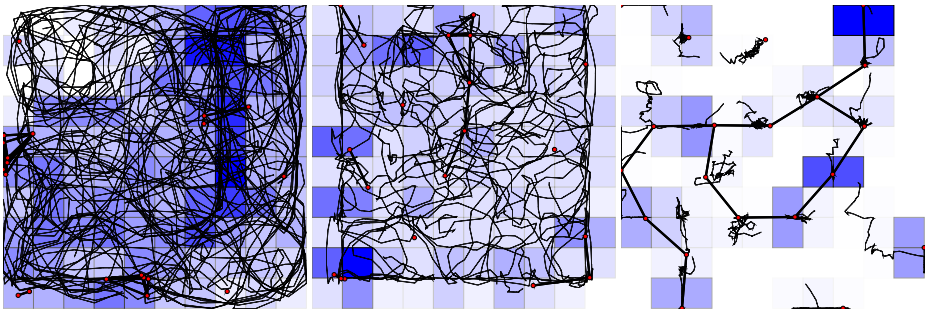


Fig. 6. Example evolved controllers using the weighted controller structure. From left to right it is possible to see: an exploration focused controller, a combination of exploration and network focused controller, and a pure network focused controller. Notice the difference in coverage and the links between the individual agents. Videos of these behaviors can be found at `https://www.youtube.com/playlist?list=PL18bqX3rX5tQN2HKdHSCna8ysbX9lUeSM`

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

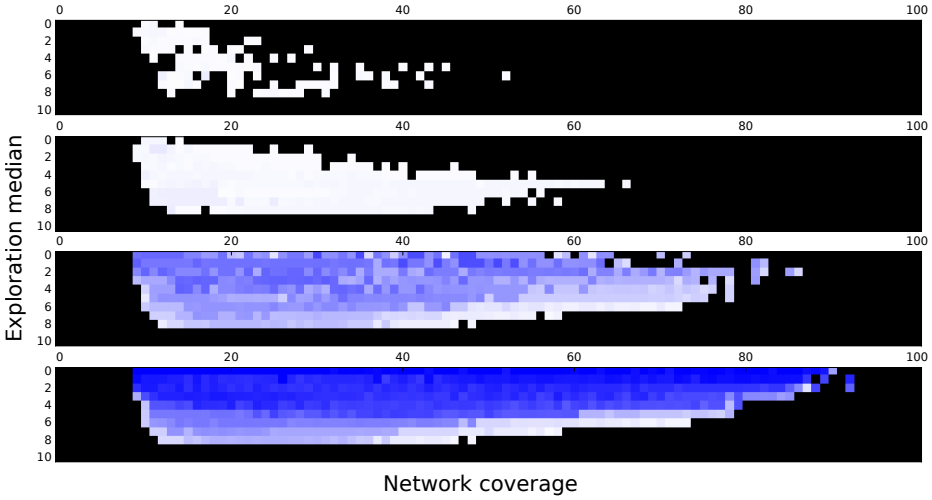


Fig. 7. Evolving parametric controllers. Figure shows epoch 1, 10, 100 and 200 with 5 evaluations per controller.

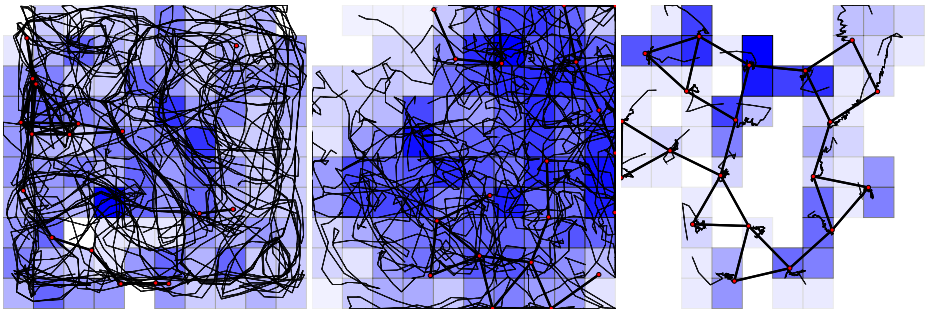


Fig. 8. Example evolved controllers using the parametric controller structure. From left to right, controllers were selected based on exploration performance, a combination of exploration and network performance, and solely large network coverage. Videos of these behaviors can be found at `https://www.youtube.com/playlist?list=PL18bqX3rX5tQN2HKdHSCna8ysbX9lUeSM`.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

less direct encoding might allow for more structured behaviors - this is an area with potential for future improvement.

6 Discussion

6.1 Comparison weighted and parametric

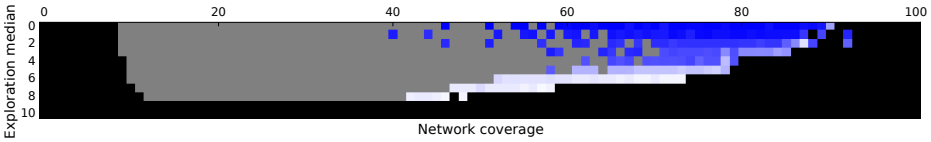


Fig. 9. The difference between the parametric controller and weighted controller repertoires at final epoch. Grey bins indicate where both controllers have a solution.

When comparing the result from the weighted and parametric controller it is clear that the weighted controller is inferior to the parametric controller. This is highlighted in Fig. 9, which shows the difference between the repertoire from the parametric controller, after subtracting all the bins that the weighted controller also found solutions for. The difference is also documented in Tab. 3, where the number of unique solutions to each controller type is shown.

Table 3. Comparison between the weighted and parametric controller. The unique column are solutions found with one controller type but not the other. Relative average fitness % is the average fitness achieved, compared to the best fitness found across all experiments.

	Solutions	Fill %	Unique	Fitness %
Weighted	429	38.6	14	96.7
Parametric	608	54.7	193	89.3

The results for the weighted controller suggest that this representation is not powerful enough to describe a controller that is good at both the exploration and the networking at the same time (Fig. 5). While it successfully manages to fill up 38.6% of the characteristics space, it is unable to do very well on the task of creating a stable communication network. As mentioned earlier, creating a stable network requires the ability to keep a given distance to neighboring agents, this is challenging for the weighted controllers as the description does not explicitly allow for hold-distance type primitive and the only way to achieve this is to make use of the boundary of the simulation area. As such, the problem with creating a stable network is exaggerated in cases where there are too few agents to cover the entire area of operations. This was the case for all the experiments described previously.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

In order to extend this architecture, we propose to use a different weight, or more precisely: a different non-scalar parametric weight function. This allows for controllers with the ability to hold a distance, effectively enabling the application of communication network maintenance. The parametric controller manages to fill 54.7% of the characteristic space, which is a 41.7% improvement over the weighted controller. From the results, it is also possible to see signs of another issue with the weighted controller approach. Compared to the weighted controller, the MAP or repertoire generated with the parametric controller is much more consistent in terms of fill. This is connected to the MAP-elites algorithm itself, the evaluation of the individuals, as well as the lack of an explicit hold distance primitive. Fitness and behavioral characteristics are evaluated post-test, based on the log for the given test. For instance, the networking characteristic considers two snapshots from the simulation; one at the end of the test and one in the middle of the test. By calculating the networking characteristic based on just these two snapshots, the metric may be susceptible to noise. In addition, we are simulating these agents, and initial conditions such as initial position, speed or even just random seed may vary. This shows an interesting challenge with the MAP-elites algorithm.

The MAP-elites algorithm is highly elitist. Within a characteristic bin, the only solution that will survive or be kept, is the best performing solution, according to fitness. Similarly, if a solution is found for an empty bin, this solution is kept, always. While these two traits may be beneficial for exploring or illuminating the search landscape, they are also detrimental to determining the exact shape of this landscape given noisy measurements. In essence, any and all variance in the fitness or characteristic function will be amplified by the MAP-elites algorithm. This is an issue that is not yet fully addressed. We introduced multiple tests for each controller in order to reduce the variation in test results. However, this comes at a cost in terms of time and processing power. Further, this may not fully solve the problem.

Consider two swarm controllers: one is unreliable, with a large spread in performance with a mean of μ. The other, a controller with a lesser spread, and with a mean of $\mu + 1$. Comparing the two, we would prefer a controller with lesser spread and a higher/better mean. However, as the MAP-elites algorithm is greedy it may very well choose the worse solution, with the higher spread and lesser mean. The worse solution only has to get lucky once, in order to beat the better and more reliable controller. A similar issue can be seen with an elitist genetic algorithm. However, we believe that this is a lesser concern for a genetic algorithm, as the population gives lesser performing solutions a chance while the MAP or repertoire does not. This suggests that care should be taken in order to avoid noise in the evaluation of candidate solutions, if the resulting repertoire is to be as accurate as possible.

7 Conclusion and Future Work

We have shown that it is possible to automatically synthesize swarm controllers for a multi-function swarm system. Our experiments showcase the ability of the

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

proposed framework to generate a large variety of controllers that can allow for finely tuned optimized behaviors for any requirements presented by a human operator. The behaviors evolved could also be used as swarm primitives with a different type of high level controller choosing the appropriate controller for the task. Evolution was done in a top-down approach, where only the skeleton of the controllers was specified, and the goals for the swarm as a whole. This also presents a contribution towards the issue of automatically generating low-level controllers from high level goals or requirements.

A focus of this study was to allow for further expansion through testing on real-world unmanned aerial vehicles by making sure that the controller inputs and outputs use only local information and are compatible with current state-of-the-art unmanned aerial vehicles. As such, a natural starting point for future work would be to conduct real-world tests on a robotics swarm platform.

It would also be possible to expand the presented framework to optimize more complex controller architectures. For instance, instead of a simple weighted sum or parametric function, a neural network could be used. With a more complex controller structure it could also be possible to include more complex tasks; for instance tasks that require sequential actions, or tasks that require agents to segment into smaller groups for optimal performance.

We would also like to conduct a more thorough investigations into the effects of noise on the MAP-elites algorithm as this poses some interesting challenges for divergent evolution.

References

1. Gross, R., Dorigo, M.: Towards group transport by swarms of robots. International Journal of Bio-Inspired Computation 1(1-2), 1–13 (2009)
2. Mitri, S., Floreano, D., Keller, L.: The evolution of information suppression in communicating robots with conflicting interests. Proceedings of the National Academy of Sciences 106(37), 15786–15790 (2009)
3. Ducatelle, F., Di Caro, G.A., Gambardella, L.M.: Cooperative self-organization in a heterogeneous swarm robotic system. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 87–94. ACM (2010)
4. Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S.M., Christensen, A.L.: Evolution of collective behaviors for a real swarm of aquatic surface robots. PloS one 11(3), e0151834 (2016)
5. Krupke, D., Ernestus, M., Hemmer, M., Fekete, S.P.: Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 413–420. IEEE (2015)
6. Duarte, M., Oliveira, S.M., Christensen, A.L.: Hybrid control for large swarms of aquatic drones. In: Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems. MIT Press, Cambridge, MA. pp. 785–792 (2014)
7. Nolfi, S., Bongard, J.C., Husbands, P., Floreano, D.: Evolutionary robotics. (2016)
8. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature 521(7553), 503–507 (2015)

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1007/978-3-319-77538-8_49

9. Berman, S., Halász, Á., Hsieh, M.A., Kumar, V.: Optimized stochastic policies for task allocation in swarms of robots. IEEE Transactions on Robotics 25(4), 927–937 (2009)
10. Schrum, J., Miikkulainen, R.: Evolving multimodal networks for multitask games. IEEE Transactions on Computational Intelligence and AI in Games 4(2), 94–111 (2012)
11. Schrum, J., Miikkulainen, R.: Evolving multimodal behavior with modular neural networks in ms. pac-man. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 325–332. ACM (2014)
12. Ellefsen, K.O., Mouret, J.B., Clune, J.: Neural modularity helps organisms evolve to learn new skills without forgetting old skills. PLoS computational biology 11(4), e1004128 (2015)
13. Basilico, N., Carpin, S.: Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 610–615. IEEE (2015)
14. Hauert, S., Zufferey, J.C., Floreano, D.: Evolved swarming without positioning information: an application in aerial communication relay. Autonomous Robots 26(1), 21–32 (2009)
15. Balch, T., Arkin, R.C.: Behavior-based formation control for multirobot teams. IEEE Transactions on robotics and automation 14(6), 926–939 (1998)
16. Spears, W.M., Spears, D.F., Heil, R., Kerr, W., Hettiarachchi, S.: An overview of physicomimetics. In: International Workshop on Swarm Robotics. pp. 84–97. Springer (2004)
17. Vadakkepat, P., Tan, K.C., Ming-Liang, W.: Evolutionary artificial potential fields and their application in real time robot path planning. In: Evolutionary Computation, 2000. Proceedings of the 2000 Congress on. vol. 1, pp. 256–263. IEEE (2000)
18. Park, M.G., Jeon, J.H., Lee, M.C.: Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In: Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium on. vol. 3, pp. 1530–1535. IEEE (2001)
19. Lee, M.C., Park, M.G.: Artificial potential field based path planning for mobile robots using a virtual obstacle concept. In: Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on. vol. 2, pp. 735–740. IEEE (2003)
20. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
21. Duarte, M., Oliveira, S., Christensen, A.L.: Hierarchical evolution of robotic controllers for complex tasks. In: Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on. pp. 1–6. IEEE (2012)
22. Duarte, M., Oliveira, S.M., Christensen, A.L.: Evolution of hybrid robotic controllers for complex tasks. Journal of Intelligent & Robotic Systems 78(3-4), 463 (2015)
23. Uchibe, E., Asada, M.: Incremental coevolution with competitive and cooperative tasks in a multirobot environment. Proceedings of the IEEE 94(7), 1412–1424 (2006)
24. Mouret, J.B., Doncieux, S.: Incremental evolution of animats behaviors as a multiobjective optimization. From Animals to Animats 10 pp. 210–219 (2008)
25. Tarapore, D., Clune, J., Cully, A., Mouret, J.B.: How do different encodings influence the performance of the map-elites algorithm? In: Genetic and Evolutionary Computation Conference (2016)