Evolving Test Environments to Identify Faults in Swarm Robotics Algorithms

Hao Wei
Department of Computer Science
University of York
York, UK
hw967@york.ac.uk

Jon Timmis
Department of Electronics
University of York
York, UK
jon.timmis@york.ac.uk

Rob Alexander
Department of Computer Science
University of York
York, UK
rob.alexander@york.ac.uk

***Abstract*—Swarm robotic systems are often considered to be dependable. However, there is little empirical evidence or theoretical analysis showing that dependability is an inherent property of all swarm robotic system. Recent literature has identified potential issues with respect to dependability within certain types of swarm robotic algorithms. There appears to be a dearth of literature relating to the testing of swarm robotic systems; this provides motivation for the development of the novel testing methods for swarm robotic systems presented in this paper. We present a search based approach, using genetic algorithms, for the automated identification of unintended behaviors during the execution of a flocking type algorithm, implemented on a simulated robotic swarm. Results show that this proposed approach is able to reveal faults in such flocking algorithms and has the potential to be used in further swarm robotic applications.**

Keywords—swarm robotics; genetic testing method;

I. INTRODUCTION

Swarm Robotics is the study of the design of groups of robots that operate without relying on any external infrastructure or on any form of centralized control [1]. Work by Winfield [2, 3] has raised concerns about the reliability of swarms in certain conditions, raising doubts over the assumption that swarm robotic systems are inherently reliable. There are significant issues in the reliable and controllable performance of a swarm in complex tasks. Issues such as communication, line of sight and failing units are examples of such problems.

Due to the emergent behaviors in swarm robotic systems, ensuring certain types of behavior emerge is challenging, and ensuring that certain types of behaviors do not emerge especially so. Software testing is a process, or a series of processes, designed to ensure that computer code does what it was designed to do and, conversely, that it does not do anything unintended [4]. There is very little literature on testing swarm robotic systems [5], yet the complexity of controller generation, coupled with coping with emergent properties of the system would indicate a complex testing strategy may well be required. Automated ways of testing swarms could potentially save significant amounts of time and identify subtle faults in the systems operation.

One of the goals of software testing is to automate, as much as possible, thereby significantly reducing its cost, minimizing human error and make regression testing easier [6]. Testing swarm robotic systems can be seen as having two levels, the code level and behavior level. For code level testing, the method designed for testing the code of single robotic system might be used for testing the code of swarm robotic systems. However, for behavior level testing, unlike single robotic system or centralized control multi-robotic system, the behavior of a swarm robotic system is not explicitly described by the behavior of the components of the system, and is therefore difficult to predict and test.

In this paper, we propose an automated testing method, based on a genetic algorithm approach. The approach generates test cases (specifically, the environment in which the swarm is executed) and monitors the movement of the swarm as it moves through the environment. Undesired behaviors will be recorded and used for assessing whether the swarm algorithm (or e.g. the design and implementation of the individual robots) has faults.

The remainder of this paper is organized as follows. Section II describes flocking rules and defines metrics for flocking behavior and failure taxonomy. Section III shows the approach for solving the problem. Section IV gives the experimental design of evolutionary computation approach. Section V gives experimental results and finally Section VI concludes the study and outlines future works for the research.

II. PROBLEM DESCRIPTION

In this paper we focus on the problem of finding undesired behavior in a specific swarm algorithm: flocking. As different kinds of swarm behaviors follow different rules, different types of test cases are needed for testing different swarm robotic systems. Flocking is an emergent behavior from a group of agents which are following a limited set of rules. By means of example in this paper, we use a simple flocking algorithm, based on Boids [7] to illustrate our approach.

A. Flocking Rules

Agents in Boids use three rules to achieve basic flocking behavior:

- Separation: steer to avoid crowding local flockmates;
- Alignment: steer towards the average heading of local flockmates;

978-1-5090-4601-0/17/$31.00 ©2017 Crown

- Cohesion: steer to move toward the average position of local flockmates.

In order to make the flocking behavior more interesting, the following two rules can be added:

- Obstacle avoidance: steer to avoid obstacles in the environment;
- Goal seeking: steer towards the direction of the goal.

B. Metrics for flocking behavior

There are various studies around flocking behavior, but most of them only develop algorithms that produce flocking behaviors and then test the behavior through manual observation of the swarm [8, 9, 10]. Few of them define metrics for measuring the performance (quality) of flocking behaviors. However, based on several papers, for example [8, 11, 12], we can conclude that good flocking behavior should have at least some of the following properties:

- The agents of the swarm should always face approximately the same direction;
- The agents or flocks that meet should stay together;
- The swarm should neither lose no agents nor separate into different swarms;
- The agents should remain close to each other;
- The agents should not collide with each other or obstacles;
- The agents should be able to reach the target.

The focus of this study is to evaluate overall behavior at the swarm level and not at an individual level, therefore we will ignore failures of single agents, e.g. motor failures. Hence, we assume that agents in the swarm will not be damaged by colliding with other agents or the obstacles, and therefore will ignore the property 5. For property 6, if the swarm is unable to reach the target in a given time, we will treat this as a total failure for the whole swarm.

There are a limited number of metrics described in the literature, which can be used to assess, in part, some of the above properties. In this paper, we use two: angular order to access property 1 and positional order to access property 2-4.

1) Angular Order

The angular order of a swarm can be used to indicate whether the agents are moving in the same direction. In [13], a mathematical model is proposed for the measurement of the angular order of self-aligned objects. By combing it with the model proposed in [14], an equation for calculating the angular order (ψ) of a group of objects can be derived - see equation 1:

$$\psi = \frac{1}{N}\left|\sum_{n}^{N} e^{i\theta_n}\right|, \tag{1}$$

where N is the total number of objects in the group, θ_n is the facing direction of the *nth* object in the group, where $\theta \in [-\pi, \pi]$, and i is the imaginary unit (complex number, which $i^2=-1$).

The value of the angular order can vary between 0 and 1. The value 0 means that the group is in a completely disordered state while 1 means that the group is perfectly aligned and is in a completely ordered state.

In a two-dimensional case, the angle describing the facing direction of an object is θ and $\theta \in [-\pi, \pi]$.

2) Positional Order

The positional order of a swarm shows whether the swarm is in a steady state or not. When a swarm is in a steady state, the distance between each agent and its neighbors is close enough that the agent is attracted to the centre of its neighborhood and is far enough that the agent can avoid colliding with its neighbors.

There are several metrics which can provide a mathematical measurement of positional order, such as the social entropy developed by Shannon [15], cohesion radius developed by Gu et al. [16], and the deviation energy developed by Antonelli et al. [17]. For this paper, we employ the social entropy metric for positional order measurement. In future work we plan to find other suitable metrics for measuring positional order.

Shannon's social entropy can be used to measure the positional order of a swarm by setting the maximum distance (h) between the individuals in the same cluster. For a given cluster, an agent r is considered to belong to this cluster if and only if in this cluster there exists another agent for which the distance between this agent and agent r is less than the maximum distance h. Shannon's information entropy $H(h)$ of a cluster with a maximum distance h is defined as:

$$H(h) = -0.5\sum_{i=1}^{M} p_i log_2(p_i) \tag{2}$$

where p_i is the proportion of the individuals in the *ith* cluster and M is the number of clusters for a given maximum distance h.

The value of H(h) varies from 0 to ∞ for a given h. A swarm with its H(h) equals 0 means that this swarm is in a steady state. The larger the value of H(h), the less steady.

C. Failure Taxonomy

Our approach will attempt to find undesired behaviors that occur during the execution of a swarm. Hence, we need a criterion to categorize failures. In swarm robotics, works such as [3], address failure modes of a single robot in the swarm, but few papers discuss taxonomies of failures for swarm behavior.

In this paper, we propose a failure taxonomy based on various causes of the failures. When a swarm is moving in an environment, the main reason of splitting up the swarm is the obstacles. When the agents meet obstacles, their velocity (both speed and moving direction) will be affected, which might cause them to lose track of the rest of the swarm. Our taxono-

my is therefore in terms of how agent speeds and directions change when a swarm fails by splitting splits.

Specifically, when a split occurs, we compare the speed and moving direction of these two clusters. We classify the nature of the split in terms of relative speeds (is the slower cluster at >= 50% of faster cluster speed, or not) and relative angle (in 10 degree increments until 90 degree, and then 90 degree plus). Hence, there will be 20 different types of failures.

We also apply the following rules:

- A cluster is only treated as a swarm if the number of agents in this cluster is larger than or equal to N (in our paper, we use 3, 5 and 7);
- During an experiment, if no swarm reaches the target, a "total failure" has occurred;
- We are only considering the last cluster which is split from the swarm;

III. METHOD

In the field of automated testing, two of the most common testing methods are random testing and testing using genetic algorithms (GA) [18]. Random testing is based on random search and is a well-known automatic testing technique [19], which can be effective at finding software bugs [20]. Yet, it is also well-known that random testing only finds simple bugs and provides low test coverage [21]. GAs provide an intelligent exploitation of a random search, and widely used to solve optimization problems [22]. They have been applied to automated software testing in conventional software applications [23] and to evolve control algorithms in swarm robotic systems [24], but to date not in testing for swarm robotics.

A. Genetic Testing Method

A GA evolves solutions by selecting, reproducing and mutating a population over many generations [25]. To use a GA, we need to design a good chromosome (the representation of each individual in the population), define an appropriate fitness function, and set appropriate GA parameters.

1) Chromosome

The term *chromosome*, for a GA, refers to a candidate solution to a problem [22]. In this study, each test case can be treated as a chromosome. In our testing approach, a test case is an environment containing obstacles in different locations so that the performance of a swarm can be tested. In all cases, the task of the swarm will be move from the starting point to the destination.

In this study, we use cellular representation [26] to represent our chromosomes. In a cellular representation, an object is represented by using directions for constructing it rather than using direct descriptions. The cellular representation for our test cases is composed of single descriptors. A single descriptor specifies an obstacle according to a simple rule. Each descriptor has five parameters (*x, y, l, w, o*) which specify its central position (*x, y*) in the environment, the length *l*, the width *w*, and *o* the orientation of the descriptor.

TABLE I. PARAMETER DESCRIPTIONS FOR GA

Name of Parameter	Description	Candidates
Population size	Number of chromosomes	[1, ∞]
Parent Selection method	How to select chromosomes	All Parent-BestHalf 2BestParent-2WorseAway 2RandomParent-2WorseAway
Crossover type and probability	A genetic operator and its usage probability	Single-point-crossover Double-point-crossover Uniform-crossover
Mutation probability	The probability of mutation	[0, 1]
Number of Generation	The number of generations before the evolution ends	[1, ∞]

If there are *N* obstacles in each test case, a random test case can be generated according to the following rules:

- Randomly generate *N* single descriptors (obstacles).
- Process single descriptors in the order they are generated, and place a corresponding obstacle in the environment.
- If part of the obstacle is outside the edge of the environment, split this part from the obstacle, but leave the descriptor for this unchanged.
- If an obstacle is totally inside another obstacle, regenerate its descriptor.
- If adding the obstacle means that there is no path between the starting point of the swarm and the destination, regenerate its descriptor.

2) Fitness Function

The fitness value of a test case in this study represents the performance of the swarm in this simulation with a given chromosome. The purpose of the testing method is to find unintended behaviors or failures in a swarm; this means that the worse a swarm performs in a test case the better (more fit) the test case is.

If the swarm flocks to the destination with the given time, this indicates that the swarm performs a perfect flocking behavior in this test case. If the swarm can not reach the destination within the given time, or the robots reach the destination separately (the robots are not moving in flock), this indicates a total failure. Hence, the fitness value of a test case can be calculated using angular order and positional order as follows:

$$FV = H / \psi \tag{3}$$

where ψ is the angular order of the flock and *H* is the positional order of the flock. In order to make the calculation simple, if the calculated fitness value is larger than 1, we'll set it to

1. The range of fitness value is therefore from 0 to 1, where 0 indicates a perfect flocking and 1 indicates a total failure.

3) Parameter Analysis

In this study, there are five parameters in our genetic algorithm which are population size, parent selection method, crossover type and probability, mutation probability, and number of generations in evolution. Table I shows the description and potential candidates for these five parameters.

We employed a parameter robustness technique from Spartan [27, 28] which allows the assessment of the sensitivity of parameters in the simulation. This technique is performed by changing each parameter individually with all other parameters remaining the same. For each parameter, simulation results for different values will be compared to determine whether a scientifically significant behavioral alteration has occurred. From the simulation results, we can also determine at which value of the parameter the genetic algorithm performs best.

IV. SIMULATION AND EXPERIMENTAL SETUP

In order to simulate the experiments for our case study, we use foot-bot in ARGoS, an open source robot simulator focusing on the simulation of large heterogeneous robot swarms [29]. The following subsection talks about the flocking algorithm which is applied to foot-bots in order to achieve flocking behavior in ARGoS.

A. Control Algorithm for Flocking Behavior

Agents in a swarm robotic system typically have limited sensors and do not share global knowledge. In this paper, the flocking algorithm we used to achieve flocking behavior is developed based on the algorithm proposed in [12]. In that algorithm, no agent (robot) has access to either the goal direction or to alignment information. However, the algorithm only works in an environment without obstacles, and is also unable to move towards a goal.

We improved this flocking algorithm by adding obstacle avoidance and goal seeking rules (see section 2.1). In order to apply obstacle avoidance, a way needs to be established to allow a foot-bot to distinguish the difference between its flockmates and obstacles. This can be achieved by using the omni-directional camera provided by ARGoS. The omni-directional camera can tell the position of a light source with respect to the centre of the foot-bot. Foot-bots in ARGoS are able to emit light of different colours with their LEDs. All foot-bots in the swarm can be set to emit light of the same colour, and then each foot-bot can use its omni-directional camera to find out the position of other foot-bots. Finally, a light source can be provided at the destination, which emits light having different colour from foot-bots. Foot-bots will be able to move towards the destination following the light, and therefore a goal seeking rule can be applied to our control algorithm.

The improved flocking algorithm should be able to reach the goal while avoiding obstacles in the environment by sharing global knowledge. However, as stated in [12], the flocking behavior accomplished is limited in the sense that the swarm could not stay cohesive the whole time.

B. Genetic Algorithm Procedure

The following procedure shows how to generate test case using the genetic algorithm:

Step 1: Initialize population: randomly generate an initial population using the rules stated in section 3.1.1 for generating test cases.

Step 2: Compute fitness: evaluate the fitness value of each test case.

Step 3: Select parents: follow the parent selection method to choose parent test cases.

Step 4: Crossover: Use a crossover procedure to produce offspring, if the produced offspring doesn't contain a clear path from the starting point to the destination, redo Step 4.

Step 5: Mutate: Allow the offspring to mutate with mutation probability p_m, if the produced offspring doesn't contain a clear path from the starting point to the destination, redo Step 5.

Step 6: Check for termination: Terminate the algorithm if we've now run N generations (where N is a parameter).

C. Experimental Setup

In this study, the environment in which foot-bots are tested is a closed 6m×6m squared space. For each experiment, 10 foot-bots will be tested. The starting point of the swarm is at the upper left corner of the environment, and the goal is placed at the lower right corner.

V. EXPERIMENTAL RESULTS

In this section, we present the experimental results of parameter selection for the genetic algorithm. Then results of the genetic testing method and random testing method are compared. We refer to a test case generated by our automated testing method as an evolved test case, and test case generated

TABLE II. PARAMETER VALUES OF GENETIC ALGORITHM

Name of parameters	Potential Candidates	Best Candidates
Number of genes	[1, 15]	6
Population size	[2, 50]	20
Parent selection method	AllParent-best half 2BestParents-2WorseAway 2RandomParents-2worseAway	2BestParents-2WorseAway
Crossover operator	Single-point-crossover Double-point-crossover Uniform-crossover	Single-point-crossover
Crossover probability	[0, 1]	0.1
Mutation probability	[0, 1]	0.05
Number of generations	[1, 100]	50

randomly as a random test case.

A. Parameter turning for the Genetic Algorithm

First, we present the parameters selected for the GA. A robustness technique is performed by perturbing parameter individually, using a 'one at a time' approach [27] – when evaluating the different values of one parameter, all other parameters will remain unchanged. For each candidate value of each parameter, 20 evolutions will be executed (The number of 20 is chosen by using the Consistency Analysis technique provided by Spartan for reducing the uncertainties during the experiments). The final results for all potential candidates will be compared using the Vargha-Delaney A-Test [30] to determine if a scientifically significant behavioral alteration has occurred, if yes, then the candidate which performs best will be selected.

Table II shows the parameter values determined according to the analysis from Spartan. All the parameter values in Table II were only optimized for a 6m×6m environment with a swarm of 10 foot-bots.

B. Comparison of Severity of Failure

In this subsection, we will compare the performance of the simulated swarm, which is equipped with the flocking algorithm mentioned in section IV.A, in both evolved and random test cases. Our means of comparison is the fitness score. We propose a null hypothesis:

H_0: the use of evolved test cases makes no difference to the ability to identify the severity of failures when compared to a random testing strategy.

At the beginning of the experiments, 5000 random test cases are produced. In order to keep the fitness evaluations between random testing method and genetic testing method the same (at 5000), and given that we will run 50 generations of the GA, 100 evolved test cases should be generated. The swarm is executed in both evolved test cases and random test cases, and the performance in each test case is calculated and recorded. When comparing the results, 100 random test cases will be randomly chosen.

Table III shows the result details of applying Mann-Whitney U-Test to the experimental results, the average (mean) fitness of the solutions (*avg*), the standard deviation (*std*), the fitness of the best solution (*best*), sum of ranking (*SoR*), mean of ranking (*MoR*), and the U-value. The p-value of the test is 0 and this result is significant at $p<0.05$, which means that the medians of the fitness values of both groups of test cases is different.

Figure 1 presents a box-and-whisker plot which shows the distribution of the fitness values of the test cases generated by genetic testing method and random testing method. From the graph, it is clear that there is no overlap in spreads (75% of random test cases are less fit than 75% of evolved test cases), therefore both mean and median of the fitness value of evolved test cases is higher than those of random test cases. Hence, the null hypothesis H_0 is rejected.

TABLE III. RESULT DETAILS OF MANN-WHITNEY U-TEST.

	avg	std	best	SoR	MoR	U-value
Evolved	0.8775	0.1123	1	13425.5	134.26	1624.5
Random	0.7253	0.0897	0.9	6674.5	66.74	8375.5

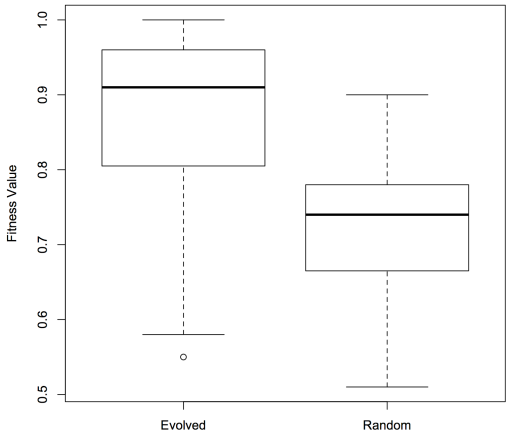

Fig. 1. The fitness value of evolved test cases compared with that of random test cases

C. Comparison of Diversity of Failure Type

The purpose of our testing method is not only to discover as many instances of undesired behaviors as possible, but also to discover as many distinct types of undesired behaviors as possible. In order to measure the diversity of failures found by our testing methods, 10 independent evolutions are carried out. For each evolution, there are 20 initial populations and the evolution continues for 80 generations. Figure 2 shows that the number of failure types won't keep increasing as the number of generations increases. The number of failure types will reach the peak after certain numbers (between 20 and 40) of generations. Hence, to keep the computing cost at a minimum, the evolutions in this subsection will run for 20 generations. We propose another null hypothesis:

H_0: the use of evolved test cases makes no difference to the ability to identify types of failures when compared to a random testing strategy.

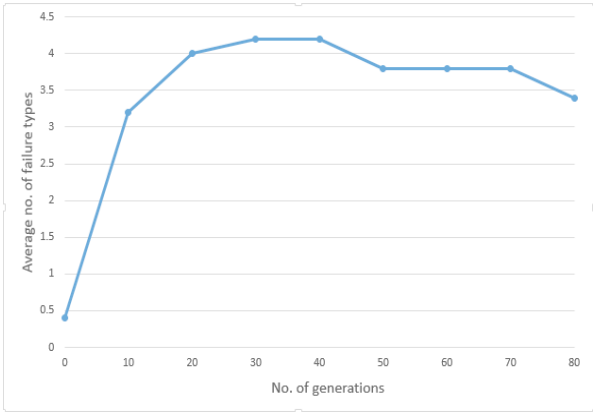

Fig. 2. A line graph shows the average number of failure types of 10 independent evolutions in 80 generations.

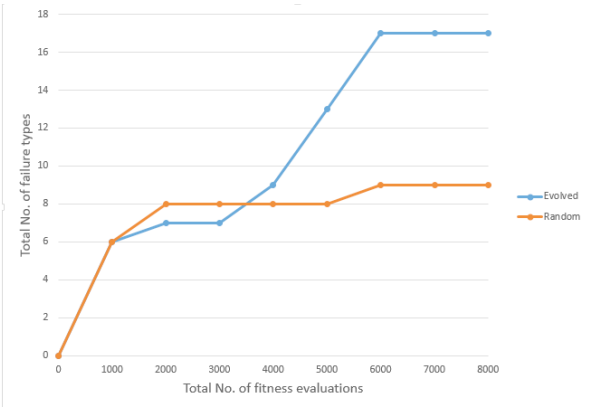

Fig. 3. A line graph shows the total number of failure types during 8000 fitness evaluations.

In this subsection, 8000 random test cases are generated. To keep the fitness evaluations between two testing methods the same, 400 evolved test cases are produced. As the population size for each evolution is 20, therefore 20 independent evolutions are carried out.

For failure taxonomy, failures are categorized into 20 categories, formed by combining two speed categories and 10 angle categories. The two speed categories are "<=50% of mean robot speed" and ">50% of mean robot speed". Angles are measured with respect to the mean heading of the swarm members. The angle categories are in the range 10-90 degrees with an increment of 10 degrees, along with a tenth "90 degrees+" category.

Figure 3 shows the total failures type found for both evolved and random test cases. In this graph, we assume that the executions of 20 different evolutions are independent and parallel. As we discussed in subsection II.C, there are 20 different types of failures in total. Figures 3 shows that evolved test cases discover 17 different types of failures, while random test cases discover only 9. Table IV shows the specific failure types discovered by both evolved and random test cases. From the table, it is clear that the set of failure types discovered by random test cases is a subset of the set of failure types discovered by evolved test cases.

One precondition for all the experimental results above is that a cluster will only be treated as a swarm if the number of agents in this cluster is larger than or equal to 5. We also carried out all the experiments in this subsection when a cluster is treated as a swarm if the number of agents in it is larger than or equal to 3 and 7. From the experimental results, even though the number of failure types for both evolved and random test cases are changing, the set of failure types discovered by random test cases is always a subset of that discovered by evolved test cases.

TABLE IV. FAILURE TYPES DISCOVERED

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Evolved			■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Random					■	■	■	■	■	■	■	■	■							

Hence, the experimental results from all three sets of experiments show that after a certain number of fitness evaluations and under the criterion we developed for failure taxonomy, the evolved test cases not only cover all the failure types which random test cases identified, but also identify more types of failures. The null hypothesis H_0 in this subsection is rejected.

D. Discussion

According to the experimental results from subsection V.B and V.C, the evolved test cases lead to worse swarm performance and cover more failure types than random test cases, suggesting that they are better tests when testing flocking algorithm in section IV.A.

VI. Conclusion

In this paper, we have proposed an automated test generation method for the identification of undesired behaviors during the execution of a swarm robotic system. A genetic algorithm was applied to generate test cases that represents the environment. The evolved test cases were compared with random test cases in simulation experiments. The evolved test cases lead to more severe and more diverse failures, the evolutionary approach thus produced a better quality of test than random did. By analyzing the failures found during the tests, there might be a chance of improving the swarm control algorithm.

In the future, this automated testing method will also be applied on physical devices. Moreover, metrics for different swarm algorithms can be established and applied for the development of corresponding automated testing methods. Those new developed testing methods will be evaluated to find out whether the approach proposed in this paper is adaptive for testing other swarm behaviors.

References

[1] Dorigo, M., Birattari, M., & Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, *9*(1), 1463.

[2] Winfield, A. F., Harper, C. J., & Nembrini, J. (2004). Towards dependable swarms and a new discipline of swarm engineering. *International Workshop on Swarm Robotics*. Springer Berlin Heidelberg.

[3] Winfield, A. F., & Nembrini, J. (2006). Safety in numbers: fault-tolerance in robot swarms. *International Journal of Modelling, Identification and Control*, *1*(1), 30-37.

[4] Myers, G. J., Sandler, C., & Badgett, T. (2011, September 23). *The art of software testing*. John Wiley & Sons.

[5] O'Grady, R., Groß, R., Christensen, A. L., & Dorigo, M. (2010). Self-assembly strategies in a group of autonomous mobile robots. Autonomous Robots, 28(4), 439-455.

[6] Ammann, P., & Offutt, J. (2008, January 28). *Introduction to software testing*. Cambridge University Press.

[7] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics*, *21*(4), 25-34.

[8] Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on automatic control*, *51*(3), 401-420.

[9] Kwong, H., & Jacob, C. (2003). Evolutionary exploration of dynamic swarm behaviour. *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. IEEE.

[10] Lindhé, M., Ogren, P., & Johansson, K. H. (2005). Flocking with obstacle avoidance: A new distributed coordination algorithm based on voronoi partitions. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE.

[11] Xiong, N., He, J., Yang, Y., He, Y., Kim, T., & Lin, C. (2010). A survey on decentralized flocking schemes for a set of autonomous mobile robots. *Journal of Communications*, *5*(1), 31-38.

[12] Moeslinger, C., Schmickl, T., & Crailsheim, K. (2009). A minimalist flocking algorithm for swarm robots. *European Conference on Artificial Life*. Springer Berlin Heidelberg.

[13] Mogilner, A., & Edelstein-Keshet, L. (1996). Spatio-angular order in populations of self-aligning objects: formation of oriented patches. *Physica D: Nonlinear Phenomena*, *89*(3), 346-367.

[14] Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., & Shochet, O. (1995). Novel type of phase transition in a system of self-driven particles. *Physical review letters*, *75*(6), 1226.

[15] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, *5*(1), 3-55.

[16] Gu, D., & Hu, H. (2008). Using fuzzy logic to design separation function in flocking algorithms. *IEEE Transactions on fuzzy Systems*, *16*(4), 826-838.

[17] Antonelli, G., Arrichiello, F., & Chiaverini, S. (2010). Flocking for multi-robot systems via the null-space-based behavioral control. *Swarm Intelligence*, *4*(1), 37-56.

[18] Ammann, P., & Offutt, J. (2008). *Introduction to software testing*. Cambridge University Press.

[19] Bird, D. L., & Munoz, C. U. (1983). Automatic generation of random self-checking test cases. *IBM systems journal*, *22*(3), 229-245.

[20] Forrester, J. E., & Miller, B. P. (2000). An empirical study of the robustness of Windows NT applications using random testing. *Proceedings of the 4th USENIX Windows System Symposium*. Seattle.

[21] Godefroid, P., Klarlund, N., & Sen, K. (2005). DART: directed automated random testing. *ACM Sigplan Notices*. ACM.

[22] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

[23] Srivastava, Praveen Ranjan, and Tai-hoon Kim. "Application of genetic algorithm in software testing." International Journal of software Engineering and its Applications 3.4 (2009): 87-96.

[24] Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T. H., Baldassarre, G. & Gambardella, L. M. (2004). Evolving self-organizing behaviors for a swarm-bot. Autonomous Robots, 17(2-3), 223-245.

[25] Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

[26] Gruau, F. (1994). Automatic definition of modular neural networks. *Adaptive behavior*, *3*(2), 151-183.

[27] Alden, K., Read, M., Timmis, J., Andrews, P. S., Veiga-Fernandes, H., & Coles, M. (2013). Spartan: a comprehensive tool for understanding uncertainty in simulations of biological systems. *PLoS Comput Biol*, *9*(2), e1002916.

[28] Alden, K., Read, M., Andrews, P. S., Timmis, J., & Coles, M. (2014). Applying spartan to understand parameter uncertainty in simulations. *The R Journal*, *6*(2), 1-10.

[29] Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., et al. (2011). ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.

[30] Vargha, A., & Delaney, H. D. (2000). A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics, 25(2), 101-132.