Swarm Robotics Algorithms: A Survey

Don Miner

May 17, 2007

Abstract

Swarm robotics is a relatively new field that focuses on controlling large-scale homogeneous multi-robot systems. We survey six example swarm robotics control algorithms to give a brief overview of the current state of the cutting-edge. Experimental results show that swarm robotics algorithms are scalable, fault tolerant, robust and efficient. These favorable results stem from the algorithms being distributed, being decentralized and depending solely on local interactions. Most algorithms surveyed in this paper were implemented for the iRobot Swarm or the Swarm-bots Project and background for both of these projects is given. We conclude with directions that swarm robotics research should continue onto next and possible future work.

1 Introduction

Swarm robotics is a relatively new field that focuses on controlling large-scale homogeneous multi-robot systems. These systems are used to develop useful macro-level behaviors while being made of modules that are very simple in design and compact in size. These properties allow robot swarms to reach populations ranging from a dozen modules to hundreds of modules. The theme of simplicity and elegance resonates throughout swarm robotics research in both the designs of the robots as well as the algorithms that are devised for these systems. The algorithms are based on the idea that complex macro-level behaviors can emerge from simple local interactions between agents. This paradigm is often inspired from observations of social insects such as ants; ants do not have a centralized control, they are not very intelligent on an individual level but are still able to perform complex colony-level behaviors such as building bridges of ants, foraging for food, migrating, etc. Swarm robotics is focused on creating a system of simple modules from which complex behaviors emerge. In this paper we survey algorithms that control these robot swarms and their emergent behavior.

Robotic swarms have several advantages over their more complex individual robot counterparts and are the results of using many robots instead of just one. This is made possible by the simple design of the robot modules because they are often less expensive and easier to build. When comparing the capabilities of a robot swarm to the capabilities of an individual robot, it is best to view the swarm as an individual entity performing complex behaviors at the macro-level. The first improvement is an obvious one: robot swarms are able to cover more area than an individual robot. This is analogous to distributed search algorithms that are able to cover different parts of a search space at once. The second improvement over individual robots is swarm robots are fault tolerant because the swarm robotics algorithms do not require robots to depend on one another. If a single module fails, the rest of the swarm can continue performing its actions as if that module never existed. Meanwhile, an individual robot system may become worthless if there is a failure in a critical component. This type of robustness is an extremely important feature in complex or hostile environments. Another feature of robot swarms is their effectiveness scales well with the

number of members. Adding more robots is all that has to be done to increase the effectiveness of a swarm. The algorithms for swarms scale well and do not depend on the number of robots. On the other hand, it is not always clear how to improve the effectiveness of an individual robot system. Often times improvements in hardware require additional software upgrades, which is not the case with swarms. These properties make multi-robot systems suitable for several application domains.

Although swarm robotics research is still relatively new and has not produced a swarm that has been used in a practical application, several have been proposed in the literature. One algorithm covered in this paper deals with the common robot task of mapping an environment [8]. A swarm of robots could disperse in an environment and cover different locations at once. The maps of the environment that are developed are superimposed on one another to provide a detailed, accurate and extensive map. Pettinaro et al. [7] suggests a few more practical applications, such as foraging for objects or patrolling in a vast environment. Foraging is a general behavior that can be used for search and rescue (or destroy), mining, food gathering, organizing, etc. Patrolling has several security applications such as detecting intruders, guarding borders, etc. None of these domains have been reported as being solved by robot swarms. Advancements in technology and algorithms needed to reach this point are discussed in the future work section of this paper.

In this paper we first cover some background that gives an overview of two robot swarm systems. Then, we survey a handful of example swarm robot algorithms that have been reported in the literature to show what these swarms can do. In our conclusion, we summarize the content herein as well as suggest what swarm robotics research should focus on next.

2 Background

Traditional Swarm Intelligence algorithms, such as Ant Colony Optimization (ACO) [1], do not transfer well to swarm robotics domains for several reasons. For example, ACO is hard to implement in a swarm robot because robots would have to drop pheromones and alter the environment, which is an unfavorable feature of robots systems and should be avoided. Meanwhile, dropping pheromones is not invasive at all in a computational world. This is a common theme when trying to transfer search algorithms and artificial intelligence techniques to robots. For example, breadth-first search is extremely inefficient when robots use it. A robot would have to backtrack and travel to that node instead of being able to move a pointer from node to node in a graph. For these reasons and others, devising algorithms for robot swarms has become a research field of its own. Swarms of robots must be built or simulated to test these algorithms, demonstrate they work and improve on previous work.

There have been a handful of robot swarms that have been built to serve as platforms for robot swarm algorithms. These projects have been very useful to the research community since simulations are not always accurate (and not as interesting). To understand the algorithms surveyed in this paper it helps to understand the capabilities of these systems. In the proceeding subsections we give overviews of two of the largest swarm robotics projects: The iRobot Swarm and the Swarmbots project.

2.1 The iRobot Swarm

The iRobot Swarm, a robot swarm of over 100 units is housed at the Massachusetts Institute of Technology and has been used as a platform for a multitude of projects and experiments. The individual modules, one of which is pictured in Figure 2, are roughly five inch cubes and have a wide array of sensors, communications hardware and human interface devices. The swarm also

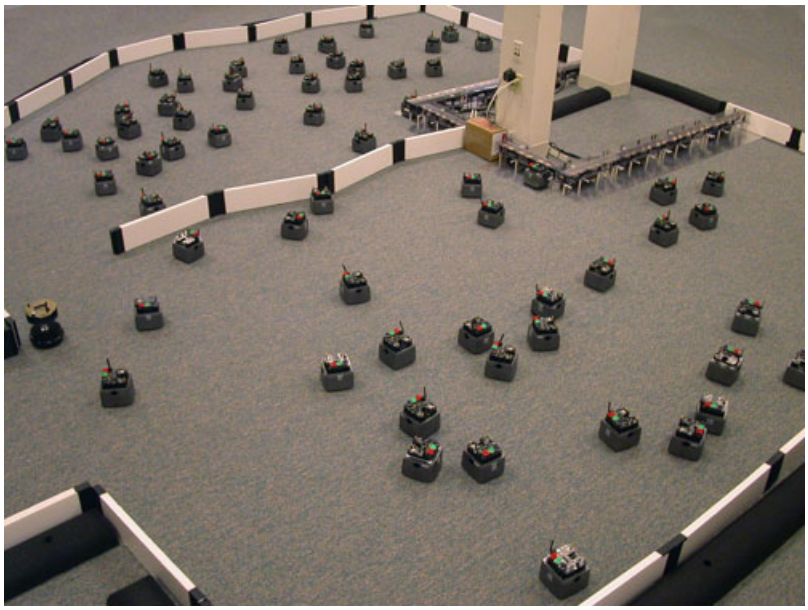

Figure 1: *The iRobot swarm.*

includes charging stations that the modules can autonomously dock to. Most research involving this project has been performed by McLurkin and associates at MIT.

Figure 2: *An individual iRobot module.*

The primary software tool used by the iRobot Swarm modules is an infrared communications system called ISIS that handles communication, localization and obstacle avoidance [4]. Robots in close proximity are able to communicate and are able to determine the locations and bearings of each other. Messages are passed with a gradient-based multi-hop messaging protocol that disperses messages throughout the swarm. Messages follow a gradient and "flow" through the network topology, which is constantly changing.

The features of the iRobot swarm already described provide a platform where researchers can implement swarm behaviors at a high level of abstraction. Most communication protocols and basic obstacle avoidance are handled by the underlying system so researchers can focus solely on emergent behavior.

In this paper we give an overview of two algorithms that have been implemented on the iRobot Swarm: a dispersion algorithm [4] and a distributed mapping and localization algorithm [8].

2.2 Swarm-bots Project

Figure 3: *A group S-bots joined together so they can traverse the hole.*

The Swarm-bots project is a large scale project based out of the Université Libre de Bruxelles, managed by Marco Dorigo and the IRIDIA Lab [2, 7]. The project has built a robot swarm platform that consists of modules called *S-bots*, one of which is pictured in Figure 4. The robots are slightly smaller than the iRobot modules and use less cutting-edge technology. Something that makes the S-bot stand out among other projects is the utilization of strong grippers to hold others to form complex structures.

Figure 4: *An individual S-bot module.*

Unlike the iRobot Swarm, the S-bots use a very basic form of communication: colors. The

S-bots do not directly communicate but their swarm behaviors can be based off of the color of and the distance to other S-bots. These readings of color and distance are all gathered from an omnidirectional camera placed at the top of the S-bot. A picture taken with a S-bot's camera is shown in Figure 5.

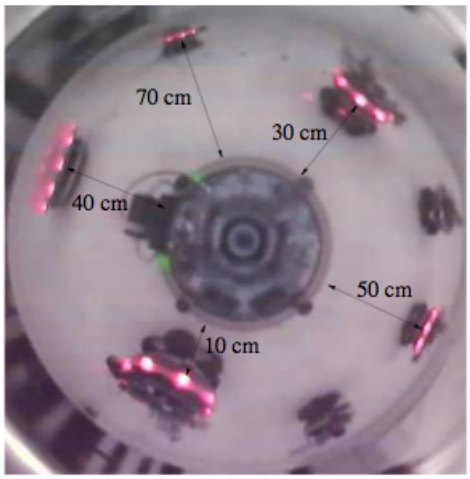

Figure 5: *The view from a S-bot's omnidirectional camera. The distance of the object is estimated based on its size.*

The Swarm-bot stays true to the simple design goal of swarm robotics and has researchers focus more on emergent behavior than complicated ad-hoc communication schemes. All projects done with the Swarm-bot platform are implementations of simple unit-level behaviors that result in interesting and useful emergent behavior. In this paper we give an overview of three individual swarm robotics algorithms that have been produced from the Swarm-bot project[1]: a hole avoidance algorithm [9], a cooperative dragging algorithm [6], and a chain formation algorithm [5].

3 Algorithms for Swarm Robotics

A variety of algorithms have been implemented to be run on swarms of robots. Some provide basic functionality, such as dispersion, while others demonstrate seemingly complex teamwork, such as chain formation. Although the algorithms all produce different emergent behavior, they all have many features in common. These features derive from the basic goals of swarm robotics discussed earlier and include:

- Simple and elegant – The robot controller that dictates the behavior of the individual robots is very simple. The behaviors of the individual robots can usually be represented as a state machine with few states and edges.
- Scalable – Swarm robotics algorithms are designed so that they work for any number of robots. Also, they are expected to scale well as new robots are added.
- Decentralized – The robots in a swarm are autonomous and do not follow any exterior commands. Although a member of a swarm can be directly and predictably influenced by the

[1]The comprehensive list of publications from the Swarm-bots project is available at the Swarm-bots website: http://www.swarm-bots.org/

behavior of another, the choice is under its own accord. Being decentralized is often coupled with being scalable.

- Usage of local interactions – Local interactions are used over broadcasting messages in the majority of these algorithms. Even broadcasts are implemented as message-hopping protocols. This ideal is a major factor in the scalability of the system

In the following subsections we give an overview of a selected number of swarm robotics algorithms. Each have been selected because the emergent behavior they generate is interesting, novel or elegant. For each algorithm we discuss how well they conformed to the ideals of swarm robotics.

3.1 Dispersion in Indoor Environments

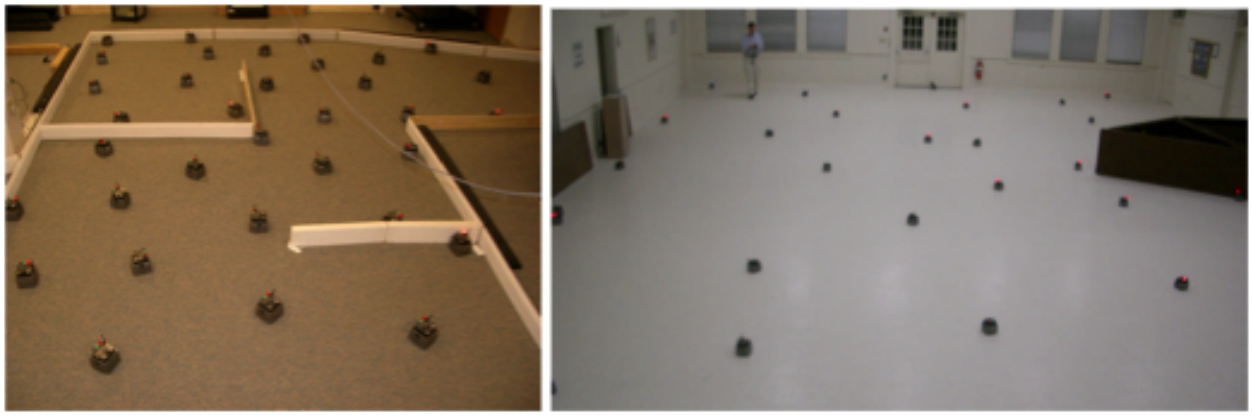

Figure 6: *The iRobot swarm uniformly dispersed over a somewhat complex environment (left) and an open space (right).*

One of the first algorithms deployed on a swarm robot is uniform dispersion. McLurkin and Smith describe their uniform dispersion algorithm for use on the iRobot Swarm [4]. In the spirit of swarm robotics, this algorithm is very simplistic and is broken into two algorithms: one that disperses robots uniformly and one that explores boundaries. These two algorithms provide different emergent behavior and are run in an alternating fashion.

The dispersion part of the algorithm has each individual robot locate its C closest neighbors. The distances to these neighbors are calculated and are used to generate vectors away from the particular neighbors. The closer the neighbor is, the stronger the repelling force. These vectors are summed and then dictates the direction the robot will move in next. The resulting emergent behavior is the robots move away from one another. It was discovered through empirical tests that $c = 2$ provided the best dispersion in practical situations.

The other part of the algorithm, frontier exploration, is designed to draw robots to the fringe robots. This way robots "flow" to new regions to fill out the space. In this phase, each robot figures out if it is a wall node (up against a wall), a frontier node (is adjacent to open space) or is an interior node (neither a wall or a frontier). Both the wall nodes and frontier nodes do not move. Next, the frontier nodes send out a gradient broadcast message. This message labels each node in terms of number of hops from a frontier in a breadth-first manner. Once this message is sent out, the interior robots move towards the lowest numbered neighbor, which represents the fastest path to the frontier. The effect of this method is the interior nodes move towards the part of the

space that will be soon explored. After this phase is done, the algorithm is switched back to the dispersion component. Now, the fringe nodes are repelled and continue to explore the frontier since the interior nodes have moved closer to them.

Tests were done in both restricted and open environments, shown in Figure 6. Also, in another test, over 100 robots dispersed in a 3000 square foot schoolhouse in approximately 25 minutes. The results show that the algorithm scales well, mostly due to its decentralized control algorithm. This method presented by McLurkin and Smith is a perfect example of a simple control algorithm that results in seemingly complex behavior. It is decentralized, scalable and efficient according to their empirical results. Whether this algorithm is fault tolerant was not directly addressed. However, this is not seen as a problem given that the algorithm is completely decentralized. This algorithm could be regarded as complex in comparison to the other algorithms we survey in further subsections of this paper. However, it stays true to the goals of swarm robotics and is a good introductory example algorithm to this field.

3.2 Distributed Localization and Mapping

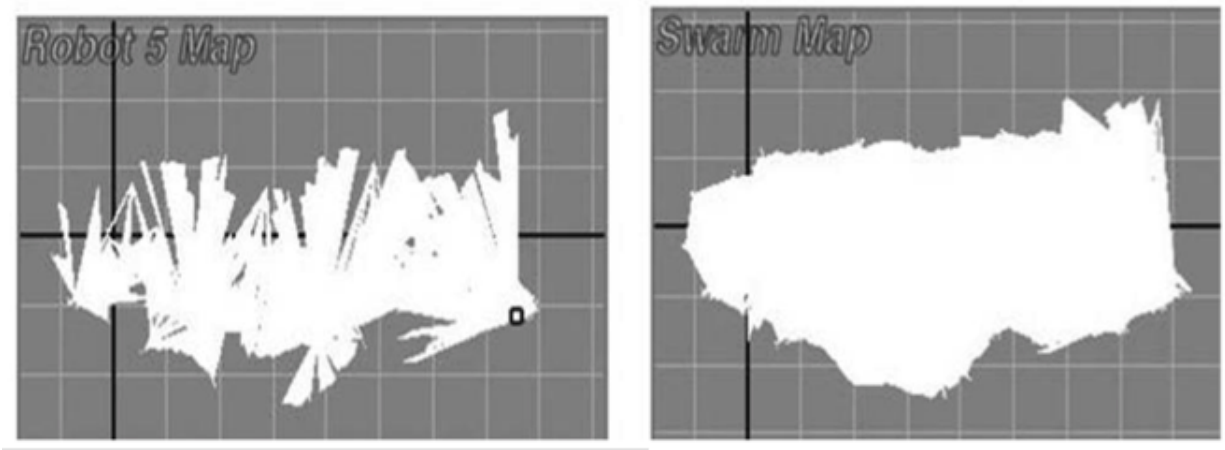

Figure 7: *A visualization of the distributed mapping algorithm. The left image shows the map contribution of a single robot. The right image shows all the robots' personal maps superimposed on one another.*

A very common robot task is to explore a building, generate a map for human use and perhaps find items of interest. Research by Rothermich et al. [8] sets out to answer the question: are robot swarms a good choice for this problem? To find out, they devise a clever algorithm that moves the iRobot Swarm through an indoor environment and generates a map. The only constraint posed is the swarm must stay close together so that no communication links are lost. Therefore, the only way to map a large area is to move through it as a group.

There is one major problem with robotic mapping, in general: the robot must know where it is and its orientation to be able to map current sensor readings to a map. Robots are left to depend on their odometry without some sort of localization scheme, which is often inaccurate and loses credibility over time. Rothermich et al. solve this problem by using beacons to denote locations. In singular robot situations this approach is unfavorable because the environment must be modified. However, Rothermich et al. propose that individual swarm robot modules can be used as mobile beacons. As the iRobot Swarm moves over an environment, anchor robots are formed at the front of the swarm and anchor robots are freed in the back. This allows the swarm robot to traverse the

terrain with decent accuracy.

The complex behavior exhibited by this algorithm is governed by simple autonomous individual decisions made at the module level. The controller on each individual robot in this algorithm follow the following steps:

1. Move in a general direction while maintaining a X, Y coordinate based on local beacons.
2. If the number of beacons goes below a certain threshold, become a beacon and start broadcasting position information to others.
3. If the number of dependent modules goes below a certain threshold, stop being a beacon and return to step 1.

The effect is the swarm moves across a terrain with front modules becoming beacons and back modules continuing on (i.e. leapfrog). While these robots are moving about the environment, they will be generating maps individually. Since all robots are using the same frame of reference in the global environment, all maps are identical in scale and orientation. This makes it a trivial process to superimpose the maps gathered by robots to generate a complete and accurate map. An example of this is illustrated in Figure 7.

This approach has two major advantages over a single robot creating a map:

- The map generated is taken from several different points of view so corners and edges are accurately captured.
- Localization in an unknown world is difficult for an individual robot. In this multi-robot approach the global frame of reference is maintained as the swarm moves onward.

This algorithm is one of the more clever swarm robotics algorithms that have been devised. Multi-robot mapping has always been a difficult problem and this algorithm appears to perform well. This algorithm is an great example of a simple, decentralized algorithm that develops complex and useful emergent behavior.

3.3 Mobile Formations

Moving a large number of robots through an environment with a large number of obstacles while maintaining connectivity is an important problem in swarm robotics. Hettiarachchi and Spears propose an algorithm that performs just that [3]. The authors present a very interesting algorithm although this domain may seem simplistic because their algorithm is based off of real physics dealing with attractions and repulsions. These artificial physics dictate the direction and speed each individual robot must move in as well as inherently avoid obstacles. In this paper, the authors discuss two physics models that can be used to dictate the forces: Newtonian physics and Lennard-Jones (LJ) forces. Newtonian physics are the rules that revolve around the famous equation $F = ma$, where F is the force, m is the mass and a is the acceleration. LJ forces on the other hand model forces between molecules and atoms and are often used to model crystalline formations, liquids and gases. The authors chose LJ because they pictured their robots "flowing" through the obstacles opposed to simply being repelled by them with Newtonian forces. Liquids, which are modeled by LJ accurately, tend to stay well connected, which is a favorable feature in swarm robotics.

Tests where a group of robots were tasked with migrating to a goal were ran in simulation. Configurations of 20 to 100 robots were used with both Newtonian and LJ physics and, as expected, LJ performed better than Newtonian physics. Surprisingly however, LJ's performance dwarfed that

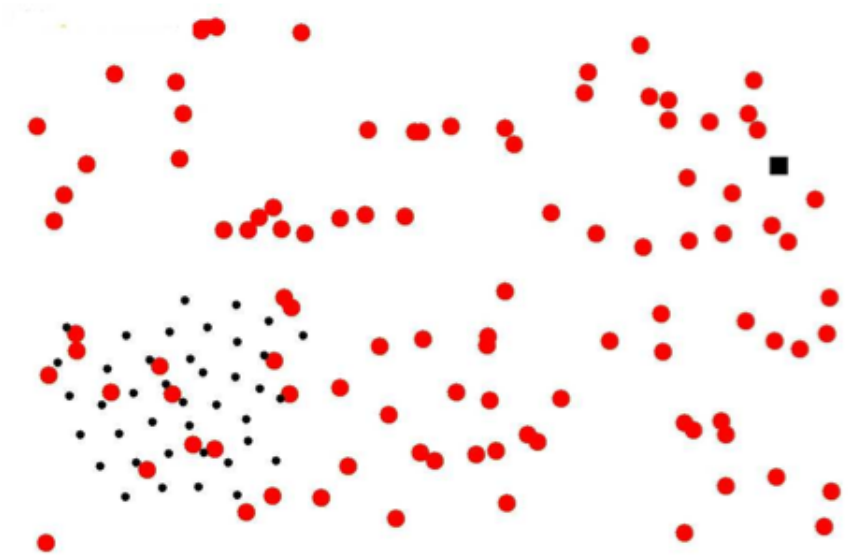

Figure 8: *A visualization of forty robots (small black dots) going towards a goal (black square). They must avoid obstacles (large red dots) while still maintaining a tight formation.*

of Newtonian physics. With 100 obstacles and 100 robots, LJ was able to consistently deliver all 100 robots to the goal and the number of obstacles only had a slight effect on the speed of the simulation. Meanwhile, in many test cases the Newtonian based algorithm was not able to deliver a single robot to the goal. It appears that Newtonian performed so poorly because the robots remained too tightly packed and did not cope well with being disconnected. At many points in the LJ simulations, only 60 robots were in communication range with one another but the swarm was still able to manage. Newtonian simulations were able to maintain a 100% connectivity rate, which is in vain because it cannot reach the goal. In the end, it appears that LJ is a great way to move swarms of robots through a group of obstacles.

This research is fundamentally different from other approaches because it used physics as its inspiration. The usage of the Lennard-Jones forces to have the robots flow through the obstacles was an ingenious idea and appears to perform well in experiments. The solution is very simple and elegant and it is obvious that the algorithm is both scalable and fault tolerant. The algorithm is completely decentralized: each module manages its own individual forces and moves accordingly. If a robot were to malfunction, the other robots would probably just regard it as an obstacle and continue on. It is unfortunate that this algorithm was only applied in simulation. Seeing this algorithm work with real robots would yield interesting results.

3.4 Cooperative Hole Avoidance

Moving in an environment efficiently is important for any robot and is often one of the first behaviors implemented. The Swarm-bot is no different and a group of joined S-bots should be able to traverse a complex landscape efficiently. However, as with other algorithms summarized in our paper, swarm robotics always complicates the problem. To solve this, Trianni et al. devise a method so that S-bots are able to move about an environment while joined, each controlling its own orientation, speed and direction autonomously [9]. The problem is harder by making joined S-bot configurations avoid holes. In robotics, holes are particularly difficult to avoid because they are hard to detect and a

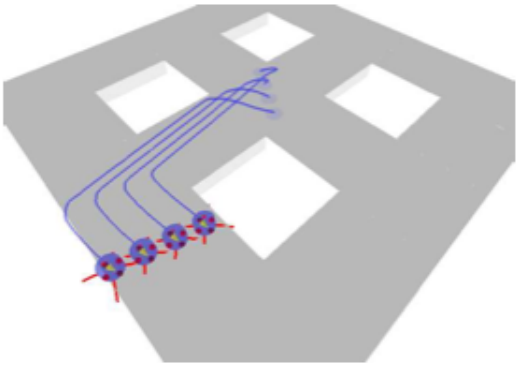

Figure 9: *Four connected S-bots moving through a hole riddled environment.*

slight mistake could be fatal. Most often a hole will only be detected when the robot is right on the edge so a quick response is favorable.

The two sensors the S-bot needs in this algorithm are its ground clearance sensors (to detect holes) and its traction sensors (to detect movement of other S-bots). No other sensors or communication mechanisms are used.

The method devised by Trianni et al. is a very common approach to programming a movement controller: use an evolutionary algorithm. A dynamic algorithm is required for this problem since the number of configurations S-bots could be in are vast and various and an evolutionary algorithm provides this. The fitness of the evolutionary algorithm favors multi-robot controllers that result in fast movement that does not fall into holes. The S-bots can use its traction sensors to detect where the other robots wish to go and cooperate accordingly. In a way, the S-bots communicate by moving in different directions and sensing their traction. The effect is an effective multi-robot movement scheme that avoids holes.

Trianni et al. performed experiments on their algorithm in simulated world with four holes, which is depicted in Figure 9. The results from these experiments were generally good. The algorithm performed well in other environments, different configurations of S-bots and different shapes of holes, as well.

This algorithm demonstrates a method for dealing with an important task that a joined group of S-bots may have to perform. The interactions between the S-bots remain simple in the spirit of swarm robotics and their test results show that the algorithm performs well with a larger number of robots. The authors do skip over a major pitfall of this work: it uses evolutionary algorithms in a simulated environment. This makes this research inapplicable for a few reasons:

- Evolutionary algorithms are very slow since they are iterative and maintain a large population of solutions.
- Learning is done in simulation and when transferred over to real S-bots, the controllers may not replicate the same behaviors seen before.
- If instead the evolutionary algorithm was applied to real robots, the S-bots would fall in holes and be irreparably damaged.
- New control algorithms cannot be learned on the fly with real S-bots. Even if a perfect controller was generated in simulation a priori, this algorithm could not be applied in an ad-hoc fashion because of the previous problem.

For these reasons, we believe that this research is not important when it comes to practical applications of swarm robotics. However, the approach does show that the traction sensors of the S-bots can be used to generate emergent behavior. At this point it is not certain how this fact can be used since the algorithm presented in this paper is not useful. In the following subsections, we cover more interesting and applicable swarm robotics algorithms applied to the Swarm-bots project.

3.5 Chain Based Path Formation

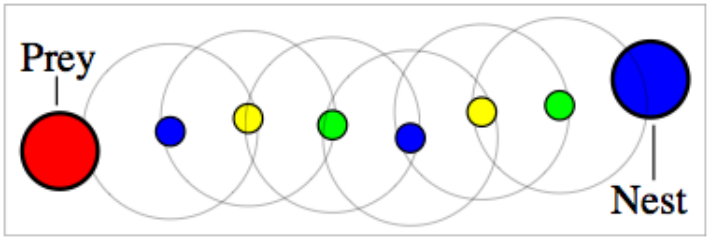

Figure 10: *A visualization of S-bots (small colored circles) forming a chain from a "nest" to a "prey." Each S-bot must be able to see (range given by thin grey circle) other s-bots in the chain.*

Nouyan and Dorigo proposed a chain based path formation algorithm for the Swarm-bots project that forms chains between a source (nest) and a destination (prey) with S-bots [5]. A visualization of a chain is depicted in Figure 10. This task is quite interesting given the limited array of sensors the S-bots are equipped with. In this algorithm the S-bots only use one sensor, their omnidirectional camera, and one communication mechanism, their LEDs. Different colors can be emitted from these LEDs and facilitate the behavior of other S-bots. Given that the communications mechanism is rather limited in range and the robots have virtually no memory, the S-bots must stay in range with one another. The only way to get a "prey" object to a "nest" object with these constraints is to form a chain.

The S-bot controller for this algorithm is very simple and elegant. The controller is designed as a simple state-machine that consists of four states: *search*, *explore*, *chain* and *finished*. Each state is associated with a repetitive task that the S-bot performs until sensory data causes the robot to transition to another state. For a visualization of this state diagram, consult the "Exploration Module" in Figure 12.

The following is an outline of states and transitions given by Nouyan and Dorigo:

- Search – randomly walk around the arena and avoid obstacles with infrared sensors. No LEDs are illuminated in this state since the S-bot is not contributing to the location of the prey.
- Explore – move along a chain by following colors. A chain always follows the following pattern: blue then green then yellow, repeated. From this information a robot knows which direction down a chain it is going (i.e. the S-bot will see BGYBGYBG... if moving away from the nest and will see BYGBYGBYG... if moving towards the nest).
- Search→Explore – if a chain member has been detected to be close.
- Chain – be a member of the chain with the color based on the previous S-bot in the chain. Actively try to maintain a specific distance from both neighbors in the chain. Also, actively maintain a 180° angle between both neighbors, give or take α (i.e. approximately opposite sides).

- Explore→Chain – if the prey is close, join the chain *or* if the tail of a chain is reached, join with a probability of $P_{e \to c}$.
- Chain→Explore – If at the tail of a chain, leave the chain with a probability of $P_{c \to e}$ per time step.
- Finished – the task has been completed. Now it can move on to other tasks, such as object transport, which is the algorithm presented in the next section.
- Explore→Finished – if the destination is really close.
- Explore→Search – no chain member is in sight. This will only be caused by an error or fault.
- Chain→Search – the previous neighbor is no longer detectable. This will only be caused by an error or fault.

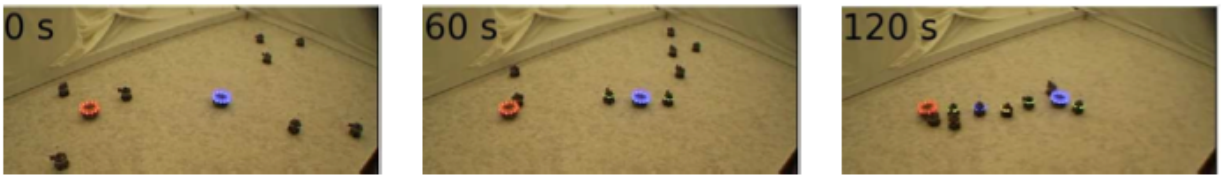

Figure 11: *A picture of S-bots forming a chain from the nest to the prey.*

Together, this simple state machine causes the behavior of chain based path formation to emerge. Intuitively, we can see how these states could develop such behavior. The robots start out moving about randomly and once they detect a chain they start moving down it. Once they reach the end of it, they either join the chain or go back down it. Every now and then, the leaf node of a chain decides to move back down the chain. This results in dynamic arms extending from a central nest that explore the domain.

To test the effectiveness of this emergent behavior, Nouyan and Dorigo ran experiments with groups of five to twenty S-bots and varying values for $P_{e \to c}$ and $P_{c \to e}$. Their results demonstrated that for simple tasks a high chance for both probabilities gave favorable results because the S-bots formed more chains and explored the space around the nest faster. On the contrary, harder tasks benefited from a low chance for both probabilities. This is because for a prey that is far away, it is beneficial to form longer chains. Their algorithm scaled well in terms of the number of robots, as expected. However, as with most search algorithms, the algorithm did not scale as well in terms of depth. The length of time required to form a path from to the prey scaled approximately quadratically.

This algorithm is perhaps the most advanced devised for robot swarms yet. Even more impressive is the Swarm-bot was able to perform this behavior even with limited sensors and communication. The four states and the carefully selected transitions performed well in developing a seemingly complex behavior. The algorithm is elegant, scalable (to a certain extent) and appears to be fault tolerant. This algorithm is expanded on by the work presented in the next subsection.

3.6 Group Transport

This work by Nouyan et al. is a continuation of his previous work that was summarized in the previous subsection [6]. This work concerns itself with what the Swarm-bot is suppose to do once

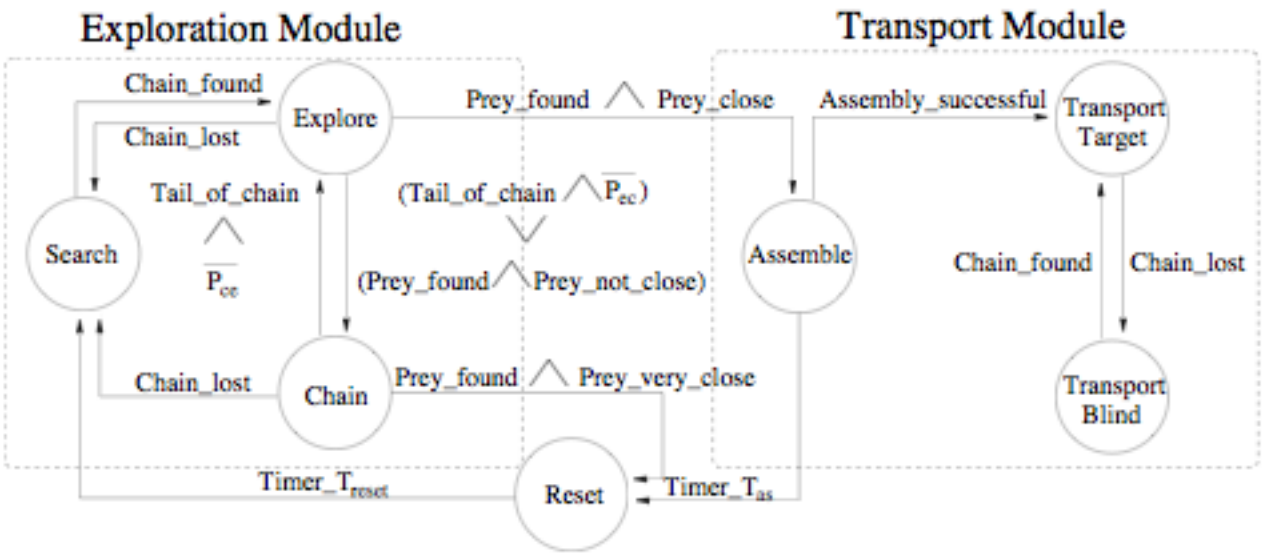

Figure 12: *The state diagram of the control program of each S-bot for this algorithm.*

it reaches the prey. Once in this situation, the Swarm-bot is tasked to pull the prey to the nest in a coordinated fashion. The basic state diagram presented in his previous work is slightly modified and is augmented with a new module, the "Transport Module." This updated state diagram is depicted in Figure 12.

The same challenges that were present in the previous work remain in this domain. The sensors are still limited, but now a torque sensor in the gripper of the S-bot will help with coordinated motion. Coordinated motion must be used because the prey is too heavy for a single S-bot to drag.

Four new states are added, which together form the Transport Module. They are outlined by Nouyan et al. as follows:

- Assemble – attach to the target object (prey or other S-bot).
- Explore→Assemble – A red object is seen.
- Transport-Target – orient self with the closest chain member in sight.
- Transport-Blind – sense the torque on the grippers and calculate the direction to push.
- Assemble→Transport-Target – the s-bot successfully grasped onto its red target.
- Transport-Target→Transport-Blind – the S-bot no longer detects a chain member.
- Transport-Blind→Transport-Target – the S-bot now detects a chain member in proximity.
- Reset – something bad happened so do nothing for a specified amount of time.
- Assemble→Reset – the S-bot does not succeed in connecting to the target object.

Although not as impressive as their previous work, the emergent behaviors contained in this work are still interesting. Once the path is formed from the prey to the nest, all S-bots traveling down the chain will join with the mass of red objects and start to help pull towards the chain. As tail robots of the chain continue to join the red mass, the prey is brought node by node closer to the next.

Figure 13: *A picture of S-bots dragging the prey to the nest.*

To test the effectiveness of their work, Nouyan et al. performed experiments of 2, 4 and 8 S-bots. They found that the time required to retrieve prey grew linearly with the distance between the nest and prey. This was expected because the distance that has to be travelled is linear in respect to the number of robots. They did find that using more S-bots than necessary to pull they prey was detrimental to performance. The prey only required two S-bots to be dragged and if instead it had more, the mass of S-bots would get stuck a lot. We believe this is a weakness in the coordinated moving algorithm presented in this work and feel that the authors skimmed over this fact.

Despites its minor shortcomings, Nouyan et al. have presented an amazing algorithm in these two papers. This algorithm successfully completes its goal of bringing a prey to the nest while maintaining the idealistic view of swarm robotics. The S-bots are simple and their sensors are not complex. The controller is just a small set of states and transitions that translates to a fully functional teamwork algorithm. This paper was published in 2006 and Nouyan et al. state "we believe that to date this study is the most complex example of self-organization in the robotics field."

4 Conclusion

In this survey paper we presented six interesting swarm robotics algorithms. All of the algorithms were extremely simplistic; perhaps the most complex being the chain formation and group transport algorithm with a grand total of seven states. The algorithms presented in this paper were all decentralized and relied completely on local interactions between modules. This property results in robust, fault tolerant and scalable systems that can be used in a variety of domains.

As shown by these examples, many basic behaviors have been defined. Similar to any other new field, it is important to get the low-hanging fruit out of the way first to provide a foundation for future research. We believe this time of inventing "neat robot tricks" for robot swarms is over. It is time for researchers to take swarm robotics to the next level and develop robust frameworks for developers to take full advantage of previous work. At this point, most emergent behavior must be hard-coded into an algorithm similar to the ones presented in this paper. A framework would allow developers to skip this step and go straight to writing applications in terms of macro-behaviors.

Since the completion of the Swarm-bots project in 2005, no work has been done to actually apply swarm robotics algorithms in practical scenarios. The results of the research are excellent and the advantages of swarm robots over traditional individual robots is clear. It is peculiar that this research has not been picked up more as a tool in practical applications. More research should be done to discover which real world scenarios swarm robots are actually effective in. In conclusion, we believe that swarm robotics is leaving its infancy and new research should focus more on applications of previous work.

References

[1] Dorigo, M., and Di Caro, G. The ant colony optimization meta-heuristic. *Mcgraw-Hill'S Advanced Topics In Computer Science Series* (1999), 11–32.

[2] Gross, R., Bonani, M., Mondada, F., and Dorigo, M. Autonomous self-assembly in mobile robotics. *IEEE Transactions on Robotics* (2005).

[3] Hettiarachchi, S., and Spears, W. Moving swarm formations through obstacle fields. *International Conference on Artificial Intelligence* (2005).

[4] McLurkin, J., and Smith, J. Distributed Algorithms for Dispersion in Indoor Environments using a Swarm of Autonomous Mobile Robots. *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)* (2004).

[5] Nouyan, S., and Dorigo, M. Chain based path formation in swarms of robots. In *ants2006*, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. St"utzle, Eds., vol. 4150 of *lncs*. springer-lncs, 2006, pp. 120–131.

[6] Nouyan, S., Gross, R., Bonani, M., Mondada, F., and Dorigo, M. Group transport along a robot chain in a self-organised robot colony. In *Proc. of the* 9^{th} *Int. Conf. on Intelligent Autonomous Systems* (2006), IOS Press, Amsterdam, The Netherlands, pp. 433–442.

[7] Pettinaro, G., Kwee, I., Gambardella, L., Mondada, F., Floreano, D., Nolfi, S., Deneubourg, J., and Dorigo, M. Swarm Robotics: A Different Approach to Service Robotics. *Proceedings of the 33rd ISR (International Symposium on Robotics) October 7* (2002), 11.

[8] Rothermich, J., Ecemis, M., and Gaudiano, P. Distributed localization and mapping with a robotic swarm. *Swarm Robotics, Springer-Verlag* (2004), 59–71.

[9] Trianni, V., Nolfi, S., and Dorigo, M. Cooperative hole avoidance in a *swarm-bot*. *Robotics and Autonomous Systems 54*, 2 (2006), 97–103.