A Distributed and Self-organized Network of Mobile Underwater Robots for Collective Search and Sampling in Dynamic Environments

Florian Berlinger* and Fritz Lekschas*

***Abstract*— In this paper we present a simulation of an underwater robot swarm. We modeled a swarm as a distributed network of mobile robots. Each robot can change its position within the network and communicate to local neighbors within its communication radius. With this model we aimed to find algorithms that self-organize the robots into a cohesive collective and allow them to coordinate their actions. Our studies include strategies for maintaining a desired number of neighbors, an analysis of information propagation and hop counts depending on the network topology, and a search mission in which the robots use self-organized and coordinated behaviors to find a signal source. Our results show that manipulating the network topology actively by keeping a minimum number of neighbors is important for successful and rapid communication as well as to avoid network partition. Our simulation can be used as a platform to rapidly investigate a variety of local rules that emerge in global collective behaviors. Given those rules, more extensive experiments can be run with real robots. The distributed nature of our simulation in which each robot runs on a separate thread facilitates the transition to real robot operations. A swarm of underwater robots could ultimately be used to find crashed airplanes more efficiently, or to sample environemntal gradients in coral reefs.**

I. Introduction

The natural world abounds with self-organizing systems, where large numbers of relatively simple agents use local interactions to produce impressive global behaviors, such that the system as a whole is greater than the sum of its parts. Well-known examples include insect colonies, flocks of birds, and schools of fish [1].

These biological collectives exhibit several properties that are highly desirable from an engineering perspective. Most prominently, they are decentralized, so that the failure of one or a few agents does not significantly affect the systems' performance. They also primarily rely on local sensing and nearest neighbor interactions, and as a result, exhibit high degrees of scalability and adaptability.

The research field of collective robotics draws inspiration from these natural systems and aims to engineer their attractive properties into de-novo, synthetic systems. So far, the majority of the success stories in collective robotics have been in 2D—mostly groundbased—systems, such as the Kilobot [2]. Going beyond 2D models, the research of the Self-organizing Systems (SSR) Group at Harvard University extends collective robotics into 3D domains.

*The authors contributed equally to the work, which was last updated on 05/11/18. Their names are listed in alphabetical order.

This paper has supplementary downloadable material available at `https://code.harvard.edu/frl487/cs262-final`.

The authors are with the John A. Paulson School of Engineering and Applied Science at Harvard University, Cambridge, Massachusetts. E-mails: {`fberlinger, lekschas`}`@seas.harvard.edu`

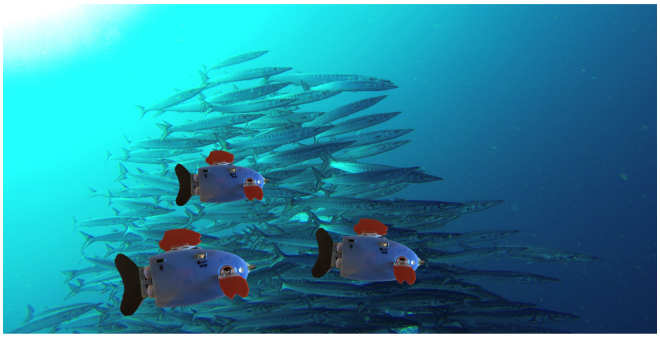

Fig. 1. On looking closely, one can spot three underwater robots swimming with a school of barracudas. We are simulating a distributed network of underwater robots to test rules for collective behaviors. Hopefully, the robots will soon not only look like fish but also mimic the behaviors of schools of fish.

While some promising results have been obtained in air, a significantly less developed but important upcoming application area is the underwater domain. Applications include environmental monitoring in sites of high ecological sensitivity such as coral reefs, gathering of data about ocean acidification and climate change, inspections of submerged wreck sites, and search-and-rescue operations over large areas.

The underwater domain however comes with a unique set of challenges. There are inherent sensory and communication deficits since some reliable modalities like GPS are not available underwater. Another characteristic underwater challenge is to monitor systems in real-time or to manually intervene for malfunctions. The systems must thus have especially high degrees of autonomy and robustness. Collective bio-inspired robotics has the potential to address these challenges by designing systems that rely on local instead of global information, thus reducing the risk of failure by distributing the task smartly across a large number of units.

The main difference to distributed systems in classic computer networks is the ability of agents to actively alter the network topology, i.e. to change their location such that the number of neighbors in the communication radius increases or decreases. Such active topology change can markedly influence the performance of the distributed systems in terms of the number of messages that are needed or the physical time it takes to accomplish some action.

In this final project, we simulate a network of sparsely-connected robotic fish. Each robot can communicate with other robots in a limited range through broadcasting undirected messages. Global consensus is eventually reached

through a cascade of messages. To study the impact of the network topology on our fish we implement a simple but powerful self-organization procedure based on the attraction-repulsion model [3].

We contribute a simplified but realistic simulation of a distributed collective of underwater robots and demonstrate self-organized behavior such as idling, aggregation, dispersion, or homing. Our experiments provide evidence that the simulated behavior is robust to various distortion of the fish' position, which acts as a proxy for underwater current, as well as dynamically-randomized white noise. We demonstrate how active topology change can lower the amount of messages being sent and reduce the physical time it takes to achieve eventual consistency in information propagation and hop count. Finally, we present how self-organized and coordinated behaviors can be combined to accomplish higher level objectives such as a search mission.

Our simulation will allow to test simple rules for local robot interactions and combine them into algorithms that can coordinate the collective. Various trade-offs, like for instance the speed of information propagation versus the density of the collective, can be rapidly studied before running more extensive experiments with robots in the wild.

We give an overview of related research at the interface of biology, robotics, and distributed systems design in Section II. In Section III, we describe the architecture of the model we created to simulate an underwater robot collective. The simulation results include studies on cohesion, information propagation, a search mission, leader following, and network topology estimation. We discuss the results in Section IV.

II. Related Work

Fish are a popular choice to study collective behaviors in animal groups. A model for fish schooling assumes two basic behavioral rules: (i) repulsion to maintain a minimum distance with neighbors [3] and (ii) attraction to avoid isolation [3]. Additionally, fish show a tendency to copy directional changes made by those ahead [4]. Applying those rules locally to change their position within the school allows fish to align and maintain a cohesive collective on a global level. Group consensus decisions can be achieved even if informed individuals are unaware of whether they are in the majority or minority [5]. Uninformed individuals in fact promote majority decisions [6]. Group behaviors can also emerge without the need for consent, based on distributed sensing of environmental gradients [7].

Bio-inspired robotics has aimed to replicate collective behaviors as seen in animal groups on a large scale. With 1024 robots, the Kilobot has come closest so far [2]. A small number of platforms exist to systematically investigate collective behaviors in 3D space. An underwater robot swarm named CoCoRo was presented in 2014 but few results with more than five robots have been reported since [8].

Systems design for mobile computing, including mobile agents, often shows a heterogeneous and hierarchical architecture [9], [10], [11], i.e. a diverse set of agents communicate with other agents at different hierarchical levels in a strict manner. For example, mobile agents move to stationary hosts, which store application specific data, and execute users' requests in local interactions. Underwater networks are also mostly heterogeneous [12], [13]. Communication protocols are designed to connect mobile and stationary agents with various degrees of information, including seafloor sensors, underwater and surface vehicles, surface buoys, onshore stations, and satellites. A Distributed Self-Spreading Algorithm (DSSA) was developed to deploy homogeneous mobile wireless sensor networks [14]. The sensor network converges efficiently from a random initial distribution to a uniform coverage of a target area. The DSSA assumes that every node knows its global absolute position in the network, e.g. by using GPS. In either case, while the agents can move freely their behavior is defined by the hierarchical communication structure.

Our approach is tailored to the characteristics of an underwater and mostly self-organized robot collective. In our simulation, all robots are homogeneous in their perception, actuation, and cognition compared to [9], [10], [11], [12], [13]. How the homogeneous behavior models are implemented is irrelevant but for this project we assume that all fish are identical instances. The robots act locally only and do not require global knowledge of any kind as opposed to [14]. Together with a hierarchy-free organization this gives us the scalability and robustness to run large-scale collectives in 3D space, extending on [15], [8]. With such a collective, we can mimic behaviors that are observed in schools of fish [3], [4], [7].

III. System Design

We model a swarm of robotic fish in a sparsely connected and dynamic network. Each robot fish is a node in a complete graph where edges represent the possible connectedness between a pair of fish and the edge weight models the physical distances between fish. The nodes are associated to a position in a 2D space that allows for the calculation of distances between them. Communication via intact edges follows a probabilistic model that depends on the edge weight. Hence, whether two fish are connected is determined dynamically based on the message transmission probability. Probabilistic communication is modeled through a `Channel` class, which is responsible for transmitting all broadcasted `Event` instances. Fish instances can interact with the environment through the `Interaction` class, i.e. relative movement or perception of other fish. Finally to be able to study various properties of our simulation we implement an `Observer` class that has god-like access to any part of our system. The entire system architecture is illustrated in in Figure 2. Our architecture aims to full decoupling of every component of the distributed system to enable gradual adjustments as needed.

A. Classes

a) Environment: The `Environment` class simulates an underwater 2D environment. It is responsible for storing

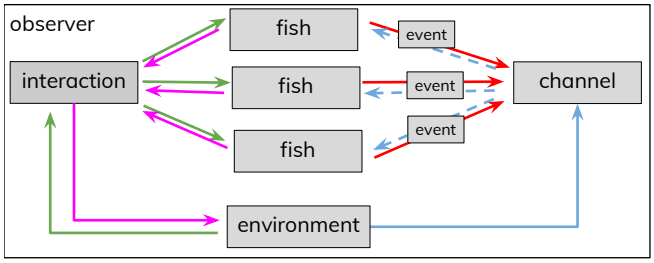

Fig. 2. Our system architecture includes multiple `fish` running in parallel on individual threads. The `environment` keeps track of their positions. `fish` can perceive each other if close enough, which is modelled by the interaction (green). Based on their perception, `fish` can change their position in the `environment` (magenta). Communication between `fish` is modelled by the `channel` (red). If their relative distance is within the communication range, `fish` can exchange messages with neighbors (blue). The `observer` simulates a tracking system with God's view in order to analyze experiments.

the position of `Fish` instances and stores position-related distortion fields. `Environment` is needed to determine the relative distance between the fish, which is used to determine the probability of successful communication a pair of fish. A uniform vector field can be provided to the environment to model time-invariant distortion of the fish's positions. To that end, the 2D environment is equally divided into a matrix grid. Upon moving to a certain grid cell, the environment adds the corresponding vector as distortion to the final position of a move. Additionally, time-varying uniformly-distributed random noise can be added to every move to model the dynamics of an underwater environment.

b) Fish: Instances of the `Fish` class represent moving agents in our distributed system and allow us to models the behavior of a fish swarm. Each instance is self-contained and cannot directly communicate with other instances. Communication is undirected and mediated through the `Channel` class. `Fish` instances can communicate with neighbors via broadcasting events and are able to actively change their position in the network by initiating a move via the `Interaction` class. The fish are implemented as synchronized thread processes, that are currently running at the same clock speed. Each clock cycle is split into two parts. In the first half of the cycle the fish are evaluating their event queue and act upon the received events. Fish current do not have memory, i.e. only the current events influence the fish behavior. Depending on the actions taken, events to be broadcasted are prepared and added to an *outbox*-like queue. During this first part of the clock cycle, the fish can also initiate one of two interactions with their environment: move and perceive as explained in the next section. In the second half of the clock cycle all outgoing events are broadcasted by the channel, which evaluates the probability of transmission and adds successfully transmitted events onto the corresponding fish's *inbox* queue. The clock cycle updates of the `Fish` instances are summarized in Algorithm 1.

c) Interaction: The `Interaction` class defines all possible interactions between `Fish` in the `Environment`.

Algorithm 1 Clock cycle updates implemented on `Fish`.

Given the simulated `Interaction` and `Channel`, a `Fish` executes four steps in each round, which determines its behavior.

1) **Evaluate received event messages**: Evaluate the event queue. For example, for every incoming `Ping` try to perceive the relative position to the source `Fish`.
2) **Prepare new events to be broadcasted**: Depending on incoming `Event`, update state and forward the new state by generating new `Event` instances.
3) **Determine next move**: (i) Derive a vector towards or away from the centroid of all neighbors, based on the number of neighbors. Add a vector that represents an overarching mission target, such as swimming from A to B. (ii) Request that move from the `Interaction`, which constrains it by the `Fish`'s maximum speed and adds ambient distortion plus random noise.
4) **Talk to neighbors**: Send all events to be broadcasted to the `Channel`.

Currently `Fish` can actively change their position by moving and perceive the relative position to other fish or objects in the environment. This models a `fish`'s sensory system and resulting perception. Given the number and relative positions of all neighbors from its own perspective, a `fish` can perform actions from the `Interaction` to maintain connectedness.

d) Channel: The `Channel` class is an intermediary between the `Environment` and the `Fish` classes. It models a wireless communication channel and handles transmissions of events within the `Environment`. Upon a newly `Event` to be broadcasted, the `Channel` retrieves the probability for a successful transmission between the source `Fish` instances and the potential target `Fish` instance and draws a random number, which is compared against the probability, to simulate the distance-based transmission probability.

e) Events: We defined specific event classes for every type of communication. `Ping` is the most ubiquitous event as it is being broadcasted by every `Fish` instances in every clock cycle. Its purpose is to determine the number of neighbors per clock cycle, which is subsequently used to perceive neighboring other fish in close proximity. The `Homing` event is used to adjust the fish behavior such that it trying to increase its neighbors. To initialize a hop count some fish broadcasts a `HopCount` event, which contains the local clock time and the number of hops since the event was broadcasted. Finally, we implement a generic `Info` event for information propagation, which lets the fish broadcast an atomic value, e.g. a string, integer, boolean, etc.

f) Observer: The god-like `Observer` class is an auxiliary class for linking and studying various properties of the distributed systems that are otherwise hidden from each other. For example, to analyze and visualize the fish position and their state we take a snapshop of both in every clock

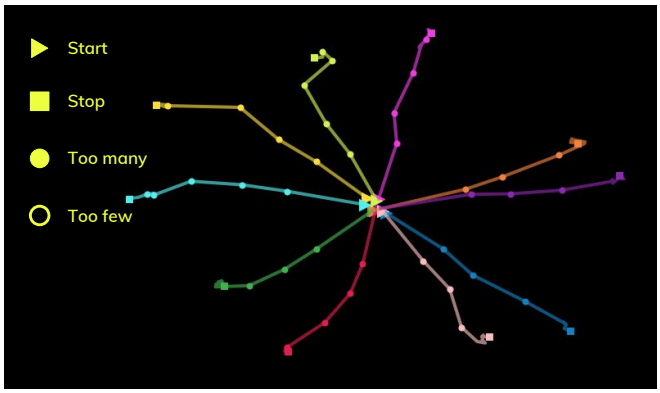

Fig. 3. Dispersion: The robots are starting close to each other and trying to reduce the number of neighbors. There is noise but no ambient distortion.

Fig. 4. Aggregation in a counterclockwise curl: The robots are starting dispersed and trying to increase their number of neighbors. Simultaneously, they are affected by a counterclockwise curl and noise.

cycle. The rationale for adding an extra class is to avoid adding code to either of the other classes that is not directly needed for the simulation to run.

B. Implementation

Our models and simulations are implemented in Python 3. The `Fish` and `Observer` class start a new threat for every instance. The fixed and synchronous clock cycle is implemented using forced sleep time. While synchronous clocks are not necessary for our system to run they make experimental assessment of the simulation much more accessible. Our experiments are implemented and documented within a Jupyter [16] notebooks that imports the classes and functions from our python files. A user can choose a setting and simulate a robot collective. Settings include the number and initial positions of `Fish`, the position distortion, communication type, and the collective behavior. The code is Open-source and freely available on `https://code.harvard.edu/frl487/cs262-final`.

IV. Simulations

We assumed holonomic movements of fishes, uniform perception and communication as well as environments with no obstacles in all simulations. During bootstrapping we ensure that all `Fish` instances are within a certain distance from each other that ensure high probability that the network of fish is not partitioned. The failure modes we allowed and studied are temporal and permanent loss of communication and perception. We did not consider message corruption or adversarial failure. For visual clarity in our figures, we show results with limited numbers of robots and rounds in a 2D space.

The three overarching questions we aimed to answer experimentally are:

1) How to maintain a connected mobile network in the presence of distortions?
2) How to switch between passive self-organization based on environmental gradients and active coordination based on robot interactions?
3) How to facilitate active network topology changes for 1) and 2)?

We were studying the questions with five main experiments: (A) low-level self-organization strategies for cohesive behavior, (B) eventual consistency in information propagation, (C) efficiency of event message hop counts, (D) leader following, and finally (E) an integrative collective search mission. With this set of experiments out goal is to (A) assess the effectiveness and robustness of our self-organizing behavior, (B) understand the how the network topology of fish influences consistency or (C) drives the performance in terms of the number of messages send, (D) study the change of self-organization strategies, and (E) evaluate if the system as a whole is able to accomplish a higher-level objective.

A. Cohesion

We investigated basic perception based behaviors such as attraction/repulsion that allow our robots to maintain a cohesive collective or connected network respectively. Connections to a fixed number of neighbors, i.e. having a network that is neither too sparse nor too dense, avoids loosing nodes in an overly sparse network, or causing communication congestion and collisions in an overly dense network.

Starting with dispersion, we initialized the robots within an area of smaller radius than the communication radius. This led to a fully connected and dense network in which all 10 robots had 9 neighbors. Subsequently, the robots aimed to decrease their number of neighbors to 3 (Figure 3).

In the opposite scenario, we initialized the robots within an area of greater radius than the communication radius. This led to a partially connected and sparse network in which robots had varying numbers of neighbors. Subsequently, the robots aimed to increase their number of neighbors to 7. We added a counterclockwise curl to show the robustness of aggregation in a dynamic environment with distortions (Figure 4).

Finally, we tested aggregation again but added the high level directive to swim from left to right to each robot's behavior (Figure 5). The robots managed to maintain a cohesive collective while starting from different positions and swimming from left to right.

Those and further basic behaviors work in the presence

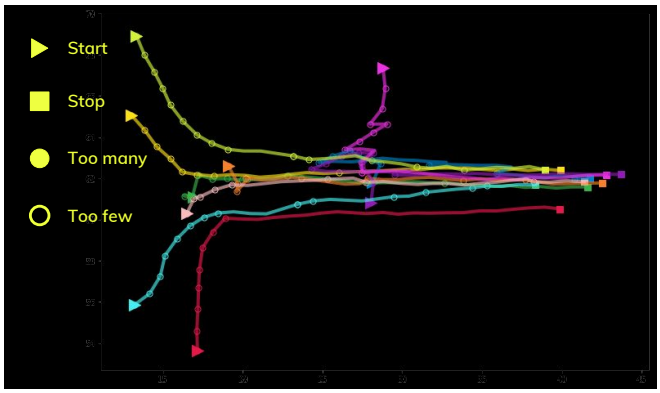

Fig. 5. Aggregation and migration: The robots are starting dispersed and trying to increase their number of neighbors. Simultaneously, their high level objective is to swim from left to right. There is noise but no ambient distortion.

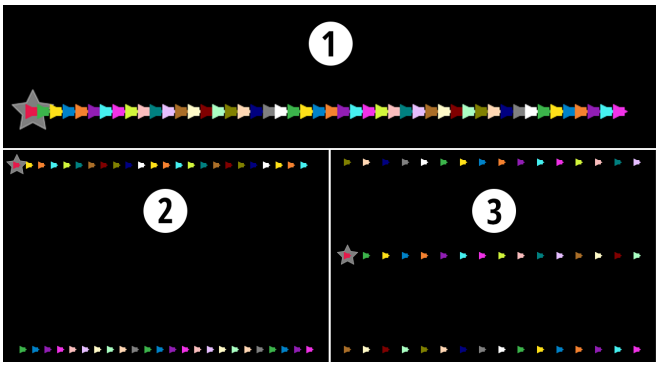

Fig. 6. Information propagation setup for (1) linear 1D arrangement, (2) zigzag 2D arrangement where each fish has at least 2 neighbors, and (3) quad-zag 2D arrangement where each fish has at least 4 neighbors. Note that the axis range is not equal: in (2) each fish is exactly 2 underwater space units apart and in 3 each fish is exactly 1 underwater space unit apart from its four direct neighbors.

of ambient distortions as long as the robot maximum speed is greater than the magnitude of the distortion. Network partitions can occur in rare cases where robots disperse or aggregate into sub-clusters.

B. Information propagation

Although the fish are mostly self-organized they need to be able to share information if needed. Under the assumption that no fish instance terminates or halts during the experiment, we are interested to find out how the network topology of fish influences consistency. Under the assumption that the network of fish is not partitioned we guarantee eventual consistency as the fish do not have a history and the last value, where last is referring to the highest clock time, is overwriting older values. Given that under probabilistic communication methods we cannot assume that every single message is successfully transmitted it is possible that consistency fails in cases where the node degree of every fish is at most 2. Such potential consistency failures could either be overcome by periodically broadcasting the same information again, in the hopes that the message arrives on a subsequent call, by requesting a confirmation, which could fail as well, or by relying on an information cascade. Since we are interested in how the topology changes impact performance, we study how altering the information cascade by increasing the number of neighbors improves consistency.

To that end, we ran 3 experiments with different static topologies for a 48 fish network and 50 repetitions each. In the first experiment the fish are aligned in a 1D line, where each neighbor can only talk to the next neighbor with approximately 62% and the second next neighbor with approximately 18%. The ensures that at least 2 neighbors can be reached with approximately 62%. And the third setup ensures that at least 4 neighbors can be reached with approximately 62%. The setup is illustrated in Figure 6.

As expected the likelihood of consistency increases as the number of neighbors increase. For the linear 1D setup the fish never reached information consistency as the likelihood of a failed transmissions increases. In the second setup, in only 2 out of 50 runs did the fish reach consistency. But for the third topology where each fish had at least 4 neighbors, the fish reach consistency in 50 out of 50 runs. This is expected as `Fish` instances broadcast new information uniformly without knowing who will receive the information. In the case of 4 neighbors each fish will likely receive 4 times the same information, hence, if one fish misses an event, it is highly likely that it gets one of the other 4 `Info` events.

C. Hop counts

A hop count combined with the communication radius gives a robot an upper bound for the distance to the far end of the collective. This allows for a rough estimation of the topology and size of a collective, as long as its distribution is approximately uniform. A uniformly distributed collective can be achieved by having each robot maintain a minimum and maximum number of neighbors. Robots with a higher number of neighbors are located closer to the center of the collective.

The minimal number of hops to send a message across a fully connected network of n robots is 1. The maximal number of hops in a sparse network with single edges between robots, i.e. a line of robots, is $n-1$.

Assuming a uniform network and given its number of local neighbors as an estimate for its own position within the network, repeated hop counts inform a robot about network density. A sudden jump in the number of hops between two hop counts would hint towards the addition of robots at the periphery of the network.

Our hop count starts and ends at a seed robot and propagates information in forward and backward direction simultaneously. Each robot updates its own count iff it hears a higher count and broadcasts this new count. The algorithm terminates when every robot heard the highest number, i.e the final count, and therefore stops broadcasting. Our hop count is robust and works for any network topology. The minimal number of messages required is n in a fully connected network. The seed sends 1 message that reaches all other $n-1$ robots and they all send their updated hop count back.

Settings [seed, network]	$\mu(msgs)$	$\sigma(msgs)$	$\mu(hops)$	$\sigma(hops)$
in center, static	87.5	11.4	2.9	0.3
in center, aggregate	66.3	15.5	2.2	0.4
at periphery, static	110	7.9	5.1	0.3
at periphery, aggregate	85.5	16.9	3.6	0.5

TABLE I

HOP COUNTS WITH 40 ROBOTS, STARTING FROM THE CENTER AND PERIPHERY OF STATIC AND DYNAMIC NETWORKS. THE AVERAGES AND STANDARD DEVIATIONS OF THE NUMBER OF MESSAGES AND HOPS REQUIRED TO COMPLETE A HOP COUNT ARE REPORTED OVER 10 RUNS.

The maximal number in an extremely sparse network, i.e. a line of robots, is $1+n\cdot(n-1)/2$. The seed sends 1 message. All other robots send 1 message when they currently have the highest count plus as many messages as there are robots with higher counts further down the line from them.

We ran four different experiments with 40 robots to analyze the number of messages and hops required to complete a hop count (Table I). We placed the seed at either the center or the periphery of the network and we made the robots keep their positions or aggregate during the count.

Aggregation helps the robots to reduce the number of messages and hops required to complete a count. A hop count started from a central location requires fewer hops than one started from a peripheral location.

D. Leader election

Leader election is similar to the hop count. Instead of counting up the maximum number of hops and broadcasting that hop count, each robot sends out the maximum ID of its own ID and the ID of the received event message and keeps track of the highest ID number. In contrast to hop count, this algorithm (sometimes called Bully Algorithm) terminates when communication ceases after all robots received the highest ID number corresponding to the leader. The robot whose own ID number is equal to the highest ID number is the leader. This guarantees a conflict-free replication of the leader ID on all robots as we implemented leader election as a conflict-free replicated data type [17]. The probabilities of some robots not receiving the ID number of the leader are depending on the network topology (see Figure 6).

E. Collective sampling and search

Collective sampling and search combines cohesion and information propagation and includes self-organized as well as coordinated behaviors. A collective of smartly coordinated robots could reduce search time for missing aircraft in the ocean.

We simulated 15 robots that sample a search area for a signal coming from a star-shaped source (Figure 7). Upon deployment, the robots disperse until the first one captures the signal and broadcasts this information. This first robot can then use perception to home in on the star. The information propagates through the network of robots and every robot that heard it switches to aggregation. Once an aggregating robot is close enough to capture the signal as well, it switches from aggregation to swimming towards the star. Steadily, the centroid of all robots shifts towards the star.

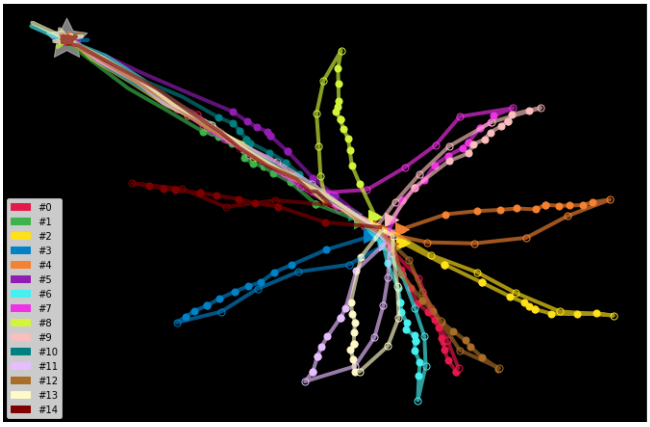

Fig. 7. Collective sampling and search: 15 underwater robots are deployed to search and rescue a star in distress.

V. CONCLUSIONS AND FUTURE WORK

We built a distributed system to simulate a network of mobile underwater robots. Our experiments showed that simple local interactions among neighboring robots can lead to coordinated global behaviors such as running a search mission in a dynamic environment. The analysis of fundamental behaviors including aggregation, dispersion, or migration revealed their implications on network cohesion, topology, and information propagation. Cohesion is achieved by imposing a minimum number of connections to neighbors that each robot has to maintain. We further found that rapid propagation of information is accomplished by changing the network topology, i.e. by having the robots aggregate and therefore increase the density of the network. We implemented a version of hop count such that the robots can estimate the density of the network.

In the next steps, we plan to formalize our results and develop a theoretical understanding that allows us to optimize our algorithms for scalability and speed. We also aim to increase the robustness of our algorithms and completely avoid network partitions.

The distributed nature of our system architecture will facilitate the transfer from simulation to experiments with underwater robots. We plan to modify and run the `Fish` on the BlueBots [18]. To this end, we will add a third dimension to the vectors for positions and velocities. The other components of our system architecture become no longer necessary, since the simulated , `Interaction`, `Event`, and `Channel` occur naturally in the physical world.

With such a collective of underwater robots, we could then replicate behaviors observed in schools of fish and reinforce biological findings about animal groups. Ultimately, we hope that our underwater robots leave the laboratory testbed behind. We envision a larger version of BlueBot searching for missing aircraft in the ocean, or making observations in coral reefs, the very place its inspiration came from (see Figure 1).

REFERENCES

[1] Camazine *et al.*, "Selforganization in biological systems," *Princeton University Press*, 2003.

[2] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[3] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, "Collective memory and spatial sorting in animal groups," *Journal of theoretical biology*, vol. 218, no. 1, pp. 1–11, 2002.

[4] Y. Katz, K. Tunstrøm, C. C. Ioannou, C. Huepe, and I. D. Couzin, "Inferring the structure and dynamics of interactions in schooling fish," *Proceedings of the National Academy of Sciences*, vol. 108, no. 46, pp. 18 720–18 725, 2011.

[5] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin, "Effective leadership and decision-making in animal groups on the move," *Nature*, vol. 433, no. 7025, p. 513, 2005.

[6] I. D. Couzin, C. C. Ioannou, G. Demirel, T. Gross, C. J. Torney, A. Hartnett, L. Conradt, S. A. Levin, and N. E. Leonard, "Uninformed individuals promote democratic consensus in animal groups," *Science*, vol. 334, no. 6062, pp. 1578–1580, 2011.

[7] A. Berdahl, C. J. Torney, C. C. Ioannou, J. J. Faria, and I. D. Couzin, "Emergent sensing of complex environments by mobile animal groups," *Science*, vol. 339, no. 6119, pp. 574–576, 2013.

[8] S. Mintchev, E. Donati, S. Marrazza, and C. Stefanini, "Mechatronic design of a miniature underwater robot for swarm operations," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2938–2943, 2014.

[9] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, and D. Rus, "D'agents: Applications and performance of a mobile-agent system," *Software: Practice and Experience*, vol. 32, no. 6, pp. 543–573, 2002.

[10] C. Raibulet and C. Demartini, "Mobile agent technology for the management of distributed systems–a case study," *Computer networks*, vol. 34, no. 6, pp. 823–830, 2000.

[11] B. Badrinath, A. Acharya, and T. Imielinski, "Designing distributed algorithms for mobile computing networks," *Computer Communications*, vol. 19, no. 4, pp. 309–320, 1996.

[12] D. Pompili and I. F. Akyildiz, "Overview of networking protocols for underwater wireless communications," *IEEE Communications Magazine*, vol. 47, no. 1, pp. 97–102, 2009.

[13] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: applications, advances and challenges," *Phil. Trans. R. Soc. A*, vol. 370, no. 1958, pp. 158–175, 2012.

[14] N. Heo and P. K. Varshney, "A distributed self spreading algorithm for mobile wireless sensor networks," vol. 3, pp. 1597–1602, 2003.

[15] M. Gauci, R. Nagpal, and M. Rubenstein, "Programmable self-disassembly for shape formation in large-scale robot collectives," *Distributed Autonomous Robotic Systems (DARS)*, 2016.

[16] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[17] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[18] F. Berlinger, J. Dusek, M. Gauci, and R. Nagpal, "Robust maneuverability of a miniature low-cost underwater robot using multiple fin actuation," *IEEE Robotics and Automation Letters (RA-L)*, 2017.