FISH: The Learned or Engineered *F*ormation of *I*ntelligent *S*warming *H*abits in Simulation

Florian Berlinger*, Katherine Binney*, Magaly Gutierrez*

***Abstract*— We present a comprehensive approach to learn from the behavior of living fish and engineer the behavior of robot collectives. To this end, we built a three dimensional simulator that realistically captures the physics of BlueBots, an underwater robot collective designed to study swarm intelligence. With that simulator, we extended existing methods of learning from observation to create replica fish that could mimic aggregation and dispersion behavior without being explicitly programmed to do so. Finally, we investigated algorithms that lead robotic fishes towards efficient swimming formations, based on the assumption that a reduction of total energy expenditure is possible when swimming in a school. Our results include algorithms that successfully lead to schooling and orbiting behaviors, and learn aggregation and dispersion from observation. In the future, BlueBots will be used to test our algorithms and transfer them into the real world, where swarms of robotic fishes could search for crashed aircraft or sample environmental data more efficiently.**

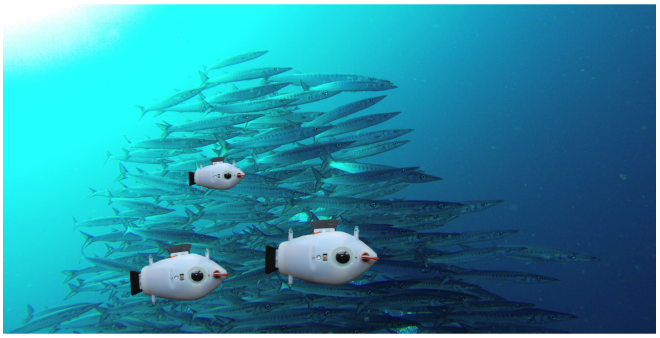

Fig. 1. On looking closely, one can spot three underwater robots swimming with a school of barracudas. Each one of them has two cameras to observe its environment and four independent fins to move in 3D space accordingly. We are working on intelligent fish-like robots capable of imitating their living counterparts. Hopefully, the robots will soon not only look like fish but also mimic the behaviors of schools of fish.

I. Introduction

In the ocean, we can see schools of fish migrating across far distances, turning on a dime to avoid sharks and other predators, or foraging in coral reefs. The ubiquity of schooling behaviors suggests there are great advantages in schooling, yet the mechanisms of schooling are still not well understood. We aim to build a robotic school of fish that allows us to empirically investigate plausible methods for schooling.

Experiments in physical robots are costly and time consuming, and the robotic platforms needed to study schooling are still under development. Within this project, we thus build a realistic simulator allowing us to test algorithms for behavior quickly and efficiently. Our efforts aim at minimizing the reality gap between simulation and robot experiments since we are planning to demonstrate schooling behaviors with a swarm of `BlueBots` [1] in the future (see Figure 1).

Current observational methods and technology allow the tracking of actions of fish in great detail. However, translating these observations to algorithmic models is difficult. One evolutionary learning method, Turing Learning, is promising in its ability to learn the behavior of replicas — the robotic fish — from observation of real agents — the biological fish. We investigate the feasibility of using Turing Learning in developing fish schooling behavior.

One hypothesis for schooling is that the overall energy expenditure of a group of fish is reduced when they are part of a school. The hydrodynamic resistance caused by the proximity and relative position of fish in a school seems to affect their overall endurance [2]. In this part of the project, we will explore local rules that enable BlueBots to school in different formations. The hope is that in the future, we will be able to measure the energy expenditure of each `BlueBot` given the formation in which it is swimming. Since BlueBots do not have any perception of hydromechanic resistance or flow speed, such rules will be vision-based. The question of what an energy efficient formation looks like will be addressed separately.

The simulator, the observational learning of behaviors, and the rule-based design of swimming formations will aid us in reverse engineering the behavior of schooling fish. We hope our insights translate into other fields as well — the simulator can simulate BlueBots and other swarming robots, Turing Learning could be used by researchers attempting other forms of biomimicry, and energy minimization may be useful for submarines and boats cruising in formation.

*The names of the authors are listed in alphabetical order. Their contributions are explained at the end of this paper.

This paper has supplementary downloadable material including our code and videos available at `https://www.dropbox.com/sh/it5uixormfvaeyt/AAAChn4eQTXEa3dEvTlHbZKVa?dl=0`. The live code lives on `https://code.harvard.edu/flb979/FISH`. Access could be granted to the course instructor if she wishes to activate her account on `https://code.harvard.edu`. Finally, we avoided `youtube`-videos to prevent publishable materials from going public. Please find all videos in the supplementary downloadable material.

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

II. Related Work

A. Simulation of Multi-Agent Systems

A variety of robotics simulators exist, including ones with multi-agent systems capabilities such as `NetLogo`, `Jade`, `v-rep`, `ARGoS`, or `Gazebo`. v-rep and Gazebo excel at graphical visualizations of simulated robots but require significant computational power to simulate large numbers of robots. ARGoS was specifically designed for simulating large-scale swarms and its visualization is more light-weight. v-rep, Gazebo, and ARGoS come with a library of robots — ARGoS for instance has a Kilobot in its library — and integrated physics engines. Robots in Jade are implemented as one thread per robot, which allows for parallelism. NetLogo on the other hand is completely synchronous, i.e., robots take actions sequentially. Furthermore, NetLogo is missing a messaging system.

The robots simulated in this paper, the `BlueBots`, operate in a simple and static environment, a 1.78 m x 1.78 m x 1.17 m tank with blank white walls. Visualizations and animations of `BlueBots` are of secondary importance, since simulations will be followed by robot experiments, which we can film. We will mainly analyze trajectories from simulation, and eventually compare them to experimental trajectories.

`BlueBot` is a novel robot, which does not exist in any library yet. We therefore have to create our own model of `BlueBot`. Since its sensors and actuators are very specific, and the physics of the static environment rather simple, we prefered to model `BlueBot` ourselves over using a physics engine. Futhermore, creating our own model will give us full control and insight into `BlueBot's` behavior.

Our first contribution is `BlueSim`, a realistic 3D simulator for `BlueBots` that allows us to pre-run collective behaviors in simulation, tune control parameters, and reduce experimental time in the water to a minimum. `BlueSim` is based on an earlier class project by Florian Berlinger and Fritz Lekschas [3]. It supports parallelism, running robots on individual threads, and simulates their interactions, communication, and the environment they are operating in. An observer keeps track of all events and robots, which allows for simulation analysis (see Figure 2 and supplementary material). Special attention was paid to facilitate the transfer from simulation to robot experiments. When running robot experiments, the simulation of interactions, communication, and the environment are obsolete; the observer is replaced by a tracking system consisting of two cameras. The `BlueBots` will ideally run the exact same `python` code like their simulated counterparts. To achieve this, we will address three current limitations of the existing simulator. First, we will move from 2D- to 3D simulations. Second, we will improve the perception model from omni-vision to a visual system with a blind spot in the back and occlusions in the field of view caused by neighboring robots. Third, we will add realistic dynamics of `BlueBot` and discard the current holonomic model.

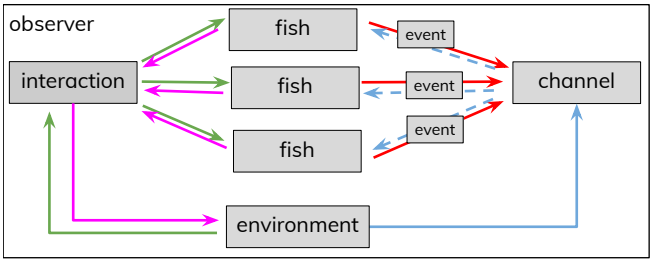

Fig. 2. The existing 2D simulator runs multiple `fish` in parallel on individual threads. The `environment` keeps track of their positions. `fish` can perceive each other if close enough, which is modelled by the `interaction` (green). Based on their perception, `fish` can change their position in the `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 in order to analyze experiments.

B. Learning Collective Behaviors

Turing learning has been proposed as a way to automatically create mimics of biological agents [4]. Its essential principle is the co-evolution of a replica and a classifier able to discriminate between the replica and the existing agent. In previous works, it has been shown that Turing Learning can be effectively used to learn models for aggregation and object clustering in e-puck robots [5]. Though Turing Learning has been proposed as a way to learn behavior directly from animals, in practice, replicas have learned the behavior of existing computer models executing a goal behavior [5].

In 1987, Reynolds proposed three rules for understanding flocking behavior that have served as a foundation for many models of schools of fish [6]. Delight et. al. formalized these rules to create an algorithmic model for flocking behavior [7]. We will treat this model as the "biological" constituent we wish to mimic. Our contribution will thus be to demonstrate Turing Learning is capable of learning this previously validated flocking behavior.

In [5] sensing consisted of a single sensor that could take two or three different values. The authors learned a reactive architecture mapping these inputs directly to velocities for two wheels. Turing Learning has not yet been applied to more complex functions. Our function will take 2 inputs with continuous values. In the scope of this project, we will learn models of dispersion and aggregation behavior for fish replicas.

Though Turing Learning has not yet been applied to fish schooling, evolutionary learning approaches with preset fitness functions have been successfully applied to learn collective behaviors. Search, which generally incorporates elements of aggregation and dispersion, was learned for 2D movement of boats in [8] and for 3D movement of drones in [9].

We expand previous work in Turing Learning, which does not use a preset fitness function, to the learning of more complex functions for more complex behaviors. Our second contribution is a robot replica that learns the mapping from two continuous inputs—the relative x- and y-coordinates of

a neighboring robot—to an output—the contribution of that robot to the replica's velocity.

C. Energy-efficient Schooling Formations

Studies on indicators of energy expenditure of fish such as tail-beat frequency and tail-beat amplitude show decreased rates when a fish is swimming as part of a school as opposed to swimming alone [2], [10]. Having BlueBots able to swim in formation will allow us to compare tail-beat frequencies among individual BlueBots in different positions within the formation. BlueBots in positions with reduced hydromechanic resistance will supposedly be able to flap their fins at lower frequencies and still maintain their relative position within the formation. The sum of all tail-beat frequencies will provide an estimate of the total energy expenditure of a chosen formation.

Inspired by this idea, we focused this part of the project on achieving different target formations and being able to switch between those formations. Local rules which bring a school of BlueBots into a *predefined* schooling formation will also allow the robots to move into an *energy-efficient* formation. In a first step, BlueBots will have to aggregate, align their velocity vectors, as well as avoid collisions. Rules for those three basic behaviors have been suggested by Reynolds [6].

The BlueBots will then have to move into a *specific* formation. For this second step, we will implement two algorithms. First, we will use a derivation of Rubenstein's DASH model [11]. DASH is an algorithm capable of generating a static formation from a swarm of robots given each robot is supplied with information of the target formation. DASH used a predefined gradient map to motivate the movement of the robots and achieve the final formation. We choose a similar approach because of its feasibility and possible adaptability to 3D modeling and physical experiments.

For the second algorithm we chose to focus on local geometric shapes. The idea behind this choice lies on the high possibility that in a deployed school of robots, a single BlueBot will not be able to see all members of the school at all times and make decisions based on that. Thus the BlueBot will choose its next position to align with its k closest neighbors in a regular geometric shape. The BlueBots are capable of estimating the relative distance to their neighbors and communicating among each other [1]. We plan to exploit these capabilities to allow dynamic movements of the simulated robotic fishes within the school formation.

Although all fishes are known to decrease their usage of energy when they are part of a school, the magnitude of this energy reduction is dependent on the position a fish takes within the school[10]. Movements within a formation can help to spread the energy expenditure equally among all BlueBots and increase the total endurance of the school. A robot in an energetically draining position could, for instance, send a signal to its neighbors when it wants to move to a more conservative position. Moreover, BlueBots know their tail-beat frequencies, which can provide a measure of energy consumption similar like in real fish (see [2], [10]).

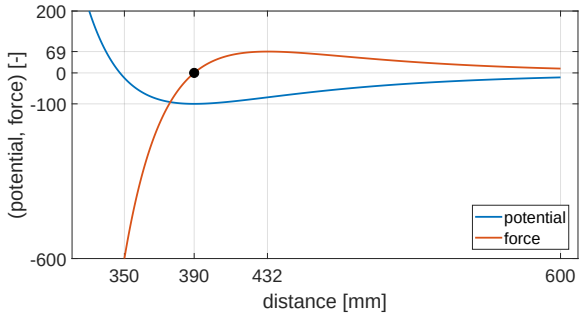

Fig. 3. The Lennard-Jones potential and force shown from the perspective of a fish. The target distance to neighbors in this example is 390 mm, which corresponds to 3 body lengths. Closer neighbors exert a repulsive force, while farther away neighbors exert an attractive force. The force magnitude of very close neighbors is much higher than far away neighbors. This results in "fixing the local neighborhood first", i.e., prioritizing local collision avoidance over global cohesion. The averaged force exerted by all neighbors can be scaled to meaningful values within BlueBot's control space.

Our second contribution is a more dynamic version of DASH engineered such that BlueBots can swim in formation and switch positions within a formation. This will allow BlueBots to exploit energetically beneficial formations found by experts in fluid mechanics or seen in nature.

III. Realistic Simulations of Collective Behaviors with BlueSim

The BlueBots were developed without ever modelling their perception and motion in simulation. Until recently, experiments with BlueBots were exclusively carried out in water. Several test runs with modifications of control parameters between them were usually required to demonstrate an intended behavior. Iterations between test runs were slow because video data had to be analyzed and code updated accordingly.

BlueSim speeds up the process of demonstrating behaviors with BlueBots manifold, since ideas, rules, and algorithms can be pre-tested in simulations, and control parameters can be tuned rapidly. This Section explains the architecture and capabilities of BlueSim including a discussion on how it captures real-world physics realistically. It presents experimental results obtained with BlueSim and compares them to previous experiments with a BlueBot.

A. Architecture of BlueSim

BlueSim is written in the language of BlueBots, python, for ease of transfer between code written for simulation and BlueBots. BlueSim's architecture is closely related to an earlier 2D version of the simulator (c.f. Figure 2). Multiple BlueBots called fish run in parallel on individual threads. The environment represents the test bed in the SSR Waterlab and keeps track of their positions. fish can perceive each other if close enough and not occluded, which is modelled by the interaction. Based on their perception and respecting their dynamics, fish can change their position in the environment. Communication between fish is modelled by the channel. If their

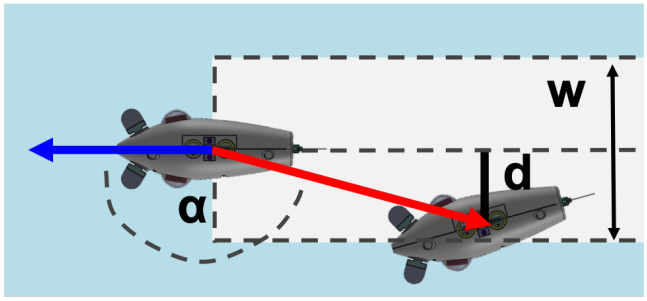

Fig. 4. The blind spot in the back of a BlueBot swimming along the blue vector is modelled as a cuboid of width w (light gray). A neighbor connected through a red vector and within the blind spot, i.e. with $\alpha > \pi/2$ and $d < w/2$, is not visible.

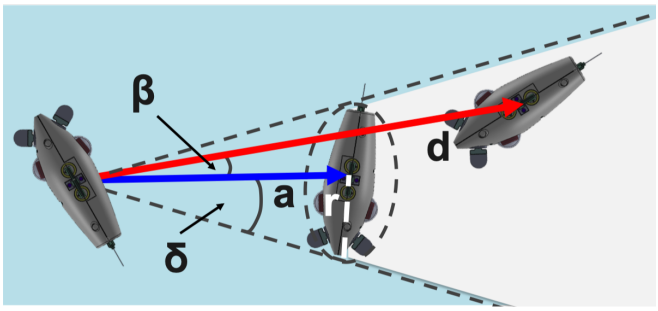

Fig. 5. Neighbor occlusions are modelled with blocking spheres of radius r surrounding each BlueBot. A neighbor connected through a red vector behind the blocking sphere of a closer neighbor connected through a blue vector, i.e. with $\beta < \delta$ and $d > a$, is not visible (light gray area).

relative distance is within the communication range, `fish` can exchange messages with neighbors. The `observer` simulates a tracking system in order to analyze experiments. Communication is not used in this paper.

B. 3D Motion and Animation of Simulated Behaviors

The position of a `fish` is represented by a three dimensional vector. The derivation of positional updates and relative positions between `fish` is based on simple vector algebra. `fish` can move within the boundaries of the environment. Any move leading to a position outside the environment is clipped.

The first collective behaviors run with multiple `fish` were inspired by Reynolds [6]. We chose aggregation and dispersion and modelled them with a single function based on the Lennard-Jones potential [12]. The Lennard-Jones potential models the interaction between `fish` based on their distances. A single parameter, a target distance at which the potential reaches its minimum, has to be set. Neighbors closer than the target distance exert a repulsive force on a `fish`, neighbors farther away an attractive one. The force contributions of neighbors are obtained by taking the first derivative of the potential with respect to their distances (see Figure 3). The force averaged over all neighbors determines the next motion of a `fish`.

Initializing multiple `fish` in a dispersed state and setting a small target distance leads to aggregation. Conversely, initializing multiple `fish` in an aggregated state and setting a large target distance leads to dispersion. Results are shown in the next subsection.

In order to visualize simulated results, we started `BlueAnimat`. `BlueAnimat` is based on `ipyvolume` by Maarten A. Breddels. It takes trajectories from simulations with `BlueSim` and creates three dimensional animated quiver plots. The observer's perspective can be changed during an animation, allowing for different view points, e.g., looking at `fish` from above, below, or the side. Moreover, we are working on a `fish`-view, allowing an observer to experience a collective behavior from within, i.e., from the perspective of one of the `fish`. The 3D experience is best when wearing virtual reality glasses, a feature that is already working.

Trajectories for `BlueAnimat` could not only originate from simulations with `BlueSim` but also from tracked data of `BlueBots` or real fish. This would allow for a direct comparison between experiments with `BlueSim` and `BlueBots`, and therefore, an assessment of `BlueSim's` authenticity. More importantly, we hope that the different viewpoints will increase our understanding of collective behaviors. With `BlueAnimat`, we will be able to go from the global observer's perspective to the local `fish'` perspective and back.

C. Robot Perception Including a Blind Spot and Occlusions

This subsection discusses a model and its implications for the local `fish's` perspective that includes a blind spot behind its own body and occlusions of neighbors caused by neighbors.

`BlueBot` has two wide-angle lens cameras to check its environment. The mounting of the cameras results in a cuboid slice behind its own body that is omitted from its almost spherical field of view. The width w of the cuboid is equal to the width of `BlueBot's` body. The vertical and longitudinal dimensions of the cuboid expand to infinity. In simulation, neighbors within this cuboid have to be removed from the list of visible neighbors. The problem is illustrated in Figure 4 and can be solved in a 2D plane, since the vertical dimension is uniform across all planar slices of the cuboid.

A neighbor within the cuboid has a directional angle $\alpha > \pi/2$ and a lateral distance d smaller than half the width of the cuboid $w/2$. The angle α is found with the dot product between the heading vector of a `BlueBot` and the vector representing the relative position to its neighbor. The distance d follows from α and the magnitude of the relative position, i.e., the planar distance to the neighbor.

A neighbor can further be positioned between a `BlueBot` and other neighbors down its line of sight (see Figure 5). Neighbors hidden behind a closer neighbor have to be removed from the list of visible neighbors of a `BlueBot` as well. Neighbor occlusion is modelled with a blocking sphere of radius r surrounding each `BlueBot`, through which no ray can pass. The spherical occlusion model is a conservative one, since it artificially inflates the width of a `BlueBot`. Importantly, it allows us to project the occlusion problem

Algorithm 1 Visible neighbors: Omit neighbors within the blind spot, then omit occluded neighbors sequentially from close to far range.

Given a `fish` with a set of neighbors and a dictionary with their relative positions:

1) **Check blind spot**
   Remove all neighbors within the blind spot from the set of neighbors.
2) **Prepare occlusions**
   Transfer the magnitudes of the relative positions from the dictionary into a list and sort that list. The neighbors are now listed by their relative distance from low to high.
   Initialize an empty list of valid neighbors and add the closest neighbor to that list.
3) **Check occlusions**
   Iterate through the sorted list and check whether a candidate neighbor is occluded by any neighbor in the valid list. If occluded, remove it from the set of neighbors. If not occluded, add it to the list of valid neighbors.
   The valid list of neighbors will grow while the set of neighbors will shrink over iterations. The final set of neighbors are the visible neighbors.

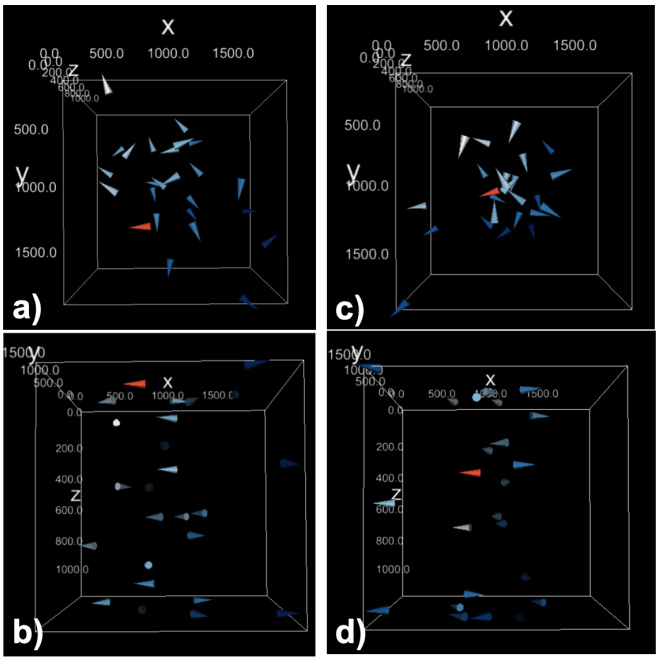

Fig. 6. Four snapshots of dispersion after 20 seconds. The overhead and side views (a and b) of dispersing robots with a `BlueBot's` blind spot show minimally better spreading than dispersing robots with a hemispherical forward looking field of view (c and d). The spreading in the *xy*- and *xz*-planes looks slightly better while it is very similar along the *z*-axis.

into 2D space because a sphere with radius r looks like a circle with radius r from *all* viewing directions. A `BlueBot` viewing at a neighbor sees that circle, which is the base of a virtual right cone. The vertex of the cone is at the position of the `BlueBot`. Its altitude a is equal to the distance to the neighbor. Its opening angle δ is defined by a and r. A further away neighbor occluded by a closer one has a distance d greater a and a relative angle β smaller δ. The relative angle β between two neighbors is found with the dot product between the vectors representing their relative positions.

Local decisions in all of the subsequently presented experiments can only be made based on visible neighbors. Active communication, such as radio transmission, is not enabled. The procedure of omitting neighbors within the blind spot (Figure 4) or blocked by others (Figure 5) is summarized in Algorithm 1. The remainder of this subsection compares different field of views in aggregation and dispersion behaviors. The effects of local decision making on a robots motion will be discussed in the next subsection.

In a **first experiment**, I investigated the influence of the **blind spot** width w on **dispersion** of 25 robots. In dispersion, robots move *away* from each other. A blind spot *in the back* presumably impairs dispersion.

I initialized the robots at random positions no farther than two body lengths (2 BL = 260 mm) from the center of the testing environment. In order to make them disperse, I set the target distance between neighbors used in the Lennard-Jones potential to the diagonal of the environment (≈10 BL).

As a performance metric for dispersion, I measured the percentage increase in the mean distance between robots inc_d over the course of a 15 s long simulation. I tested four different widths $w = \{0, 50^*, 250, 2517^{**}\}[mm]$ and found no drastic impact of the blind spot on dispersion (*actual width of the blind spot behind `BlueBot`; **diagonal of the environment, resulting in a hemispherical 180° forward looking field of view). The respective increases in mean distance averaged over five test runs for each width were $inc_d = \{151.6, 143.4, 141.4, 122.4\}[\%]$.

While dispersion performance decreases with increasing widths, even robots with an unrealistic blind spot of 50 body widths — not even one such robot of corresponding actual size would physically fit inside the testing environment — dispersed. Those robots saw as few as zero neighbors in some iterations. An explanation made visible in animations is that the robots extensively use the third vertical z-dimension to disperse (see Figure 6). They disperse only slightly slower with a hemispherical forward looking field of view than the real field of view of the `BlueBots`.

In a **second experiment**, I investigated the influence of the **blocking sphere** radius r on **aggregation** of 25 robots. In aggregation, robots move *toward* each other. Occlusions in the *forward* facing field of view presumably impair aggregation.

I initialized the robots at random positions of up to ten body lengths (=1300 mm) from the center of the environment and set the target distance to two body lengths (=260 mm) in order to make them aggregate.

As a performance metric for aggregation, I measured the percentage reduction of the mean distance between robots

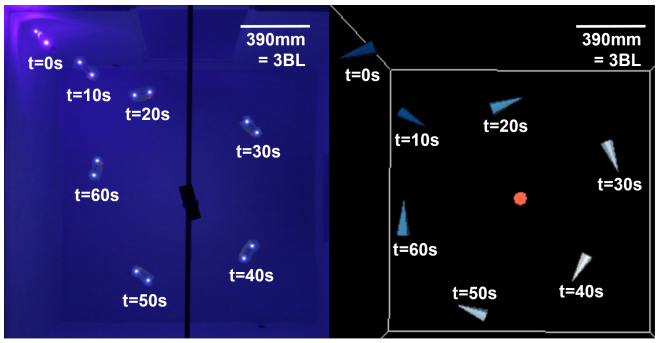

Fig. 7. Seven superimposed overhead snapshots of three dimensional orbiting in the SSR Waterlab (left) and simulation (right). The `fish` in `BlueSim` is running the same control code like `BlueBot` in the water. The reality gap for this simulated behavior is small.

red_d over the course of a 15 s long simulation. I tested seven different radii $r = \{0, 65^*, 130, 195, 390, 520, 1040\}[mm] = \{0, 0.5^*, 1, 1.5, 3, 4, 8\}[BL]$ and found no drastic impact of the blocking sphere on aggregation (*actual radius of sphere surrounding `BlueBot`). The respective reductions in mean distance averaged over five test runs for each radius were $red_d = \{32.4, 32.0, 27.8, 25.2, 23.8, 24.0, 16.8\}[\%]$.

While aggregation performance decreases with increasing radii, even robots with an unrealistic blocking sphere of radius r = 8 BL — not even one such robot of corresponding actual size would physically fit inside the testing environment — aggregated. Those robots saw as few as two neighbors in each iteration. A possible explanation is that aggregation *toward the center* of all robots proceeded like a cascade of following behaviors propagating *from the center* in outward direction.

The new perception model will allow us to anticipate the effects of the blind spot and the blocking sphere on collective behaviors. The experiments in this subsection were already run with a new dynamics model that will be discussed next.

D. Robot Dynamics based on Actuations of Fins

The local coordinate frame attached to a `BlueBot` follows the ISO axes convention for aircraft: positive x is forward, y is rightward, and z is downward. Yaw, pitch, and roll describe rotations around the x-, y-, and z-axis respectively. `BlueBot` is by design passively stable and neutral in pitch and roll, as well as positively buoyant with a buoyancy force F_{buoy} of 0.010 N permanently acting in negative z-direction, i.e., upward.

`BlueBot` has four independent fins to control its three dimensional motion. The caudal and dorsal fins generate thrust forces $||F_{caud}|| = ||F_{dors}||$ of up to 0.020 N in forward x- and downward z-direction respectively. The pectoral right and left fins generate thrust forces $||F_{PR}|| = ||F_{PL}||$ of up to 0.006 N in the xy-plane. Their thrust forces contribute to motions in the x- and y-directions since they are rotated backward at yaw angles γ_{pect} of $\pm\pi/6$ against the y-axis. Furthermore, the pectoral fins are mounted offset from `BlueBot's` center of rotation at a distance d_{pect}, therefore introducing a rotational motion around the z-axis.

All fins can be actuated independently. For instance, running the caudal, dorsal, and pectoral right fin together could result in a right turn at constant depth; running one of the pectorals only results in near-on-the-spot turning; running both pectorals accelerates `BlueBot` in backward direction.

Besides the actively controllable thrust forces generated by `BlueBot's` four fins, passive and velocity dependent drag forces act to oppose `BlueBot's` motion. Considering all forces acting on `BlueBot`, a non-linear time-invariant dynamics model can be formulated. The corresponding equations of motion and all relevant modelling parameters are shown and used in Algorithm 2. The parameters were found empirically by measuring `BlueBot's` accelerations and velocites, as well as matching its dynamics to video data. The control of the fins and corresponding generation of thrust forces were assumed to be instantaneous.

Since `BlueBot` is passively stable in roll and pitch, its motion in z-direction can be decoupled from its xy-planar motion, effectively reducing the control and dynamics problem from 3D to 2.5D. At any iteration of its control loop, `BlueBot` considers its perceptual information and makes a decision on where to move next. It then sets its fins to effect the desired motion, using three independent P(I)D-controllers. The first one controls depth, while the other two control planar heading and velocities respectively.

Given the current fin controls, velocities, and the orientation of `BlueBot` in any one iteration of its control loop, the dynamics model simulates its movement until the next iteration by solving the equations of motion with Euler integration. Control happens from a `BlueBot's` perspective in its local frame, and velocities along its axes of motion are local properties. However, `BlueBot's` position in the `environment` has to be expressed in global coordinates. The local velocities are therefore transformed into global space by rotation around the z-axis, and integrated there to derive positional updates. The full procedure of simulating `BlueBot's` motion between two control iterations is shown in Algorithm 2.

Previously, `BlueBot's` motion was holonomic and only constrained by the magnitude of its velocity vector, which led to unrealistic non-smooth and discontinuous movements. This new dynamics model enables authentic motion as shown in Figure 7 of the next subsection.

E. Authenticity and Potential of BlueSim

The ultimate goal for `BlueSim` is to close the reality gap between simulation and experiments with `BlueBots`. We discussed essential methods to do so previously in this Section. The ultimate tests for `BlueSim` are simulated collective behaviors that are recognizable from previously seen behaviors of `BlueBots` or living fish, and control parameters that work just as well in real robot experiments as they do in simulation.

Using `BlueSim`, we replicated *orbiting*, an experiment previously run with a `BlueBot`. The code for controls and the control parameters were copied from the orbiting

Algorithm 2 Forward dynamics: At each control iteration, take a `fish`'s current dynamics and control states and derive its next dynamics state.

Given water density $\rho = 998$ $[kg/m^3]$, Euler integration step width $\Delta t = 0.01$ $[s]$, and the following static `fish` parameters:

- $f_{ctrl} = 1$ $[Hz]$, control update frequency
- $l = 0.150$ $[m]$, length including fin
- $w = 0.050$ $[m]$, width; $h = 0.080$ $[m]$, height
- $A_x = \pi/4 \cdot h \cdot w$ $[m^2]$, reference area in x-direction
- $A_y = \pi/4 \cdot l \cdot h + 2 \cdot 0.00075$ $[m^2]$, including fins
- $A_z = \pi/4 \cdot l \cdot w$ $[m^2]$
- $A_\phi = A_y$, reference area for turning around z-axis
- $m = 2 \cdot 0.25$ $[kg]$, mass including added mass
- $I = \frac{m}{5} \cdot \frac{1}{4} \cdot (l^2 + h^2)$ $[kg \cdot m^2]$, inertia
- $c_{dx} = 0.5$ [-], drag in x-direction (c.f. cone)
- $c_{dy} = 2.1$ [-], c.f. flat plate; $c_{dz} = 0.7$ [-]
- $c_{d\phi} = 1.0$ [-], drag for turning around z-axis
- $d_{pect} = 0.055$ [m], pectoral distance from center
- $\gamma_{pect} = \pi/6$ [rad], pectoral mounting angle
- $F_{buoy} = 0.010$ [N], buoyant force

Given a `fish` with current velocity vector v, orientation ϕ, and angular velocity v_ϕ in its local frame, and fin control F_{caud}, F_{PR}, F_{PL}, F_{dors} (caudal, pectoral right/left, dorsal)

Repeat the following 3 steps $(f_{ctrl} \cdot \Delta t)^{-1}$ times:

1) **Equations of motion:** Derive current accelerations from current velocities and fin controls.
$\dot{x} = v_x$, $\dot{y} = v_y$, $\dot{z} = v_z$; $\dot{\phi} = v_\phi$

$$\ddot{x} = 1/m \cdot \big(F_{caud} - \sin(\gamma_{pect}) \cdot F_{PL} - \sin(\gamma_{pect}) \cdot F_{PR} - \tfrac{1}{2} \cdot \rho \cdot c_{dx} \cdot A_x \cdot sgn(\dot{x}) \cdot \dot{x}^2\big)$$

$$\ddot{y} = 1/m \cdot \big(\cos(\gamma_{pect}) \cdot F_{PL} - \cos(\gamma_{pect}) \cdot F_{PR} - \tfrac{1}{2} \cdot \rho \cdot c_{dy} \cdot A_y \cdot sgn(\dot{y}) \cdot \dot{y}^2\big)$$

$$\ddot{z} = 1/m \cdot \big(F_{dors} - F_{buoy} - \tfrac{1}{2} \cdot \rho \cdot c_{dz} \cdot A_z \cdot sgn(\dot{z}) \cdot \dot{z}^2\big)$$

$$\ddot{\phi} = 1/I \cdot \big(d_{pect} \cdot \cos(\gamma_{pect}) \cdot F_{PL} - d_{pect} \cdot \cos(\gamma_{pect}) \cdot F_{PR} - \tfrac{1}{2} \cdot \rho \cdot c_{d\phi} \cdot A_p hi \cdot sgn(\dot{\phi}) \cdot (\tfrac{l}{6} \cdot \dot{\phi})^2\big)$$

2) **Euler integration:** Integrate current accelerations to derive next step velocities.
$v_x = \dot{x} + \Delta t \cdot \ddot{x}$, $v_y = \dot{y} + \Delta t \cdot \ddot{y}$, $v_z = \dot{z} + \Delta t \cdot \ddot{z}$

$v_\phi = \dot{\phi} + \Delta t \cdot \ddot{\phi}$ and $\phi = \phi + \Delta t \cdot \dot{\phi}$

3) **`fish` to global transformation:** Given the current orientation, transform the current velocities into the global coordinate frame. Integrate the global velocities to update the global positions.
${}^g\dot{P}_r = R_z(\phi) \cdot \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$

${}^gP_r = {}^gP_r + \Delta t \cdot {}^g\dot{P}_r$

Finally, update the global position and velocity of the `fish` with the final ${}^gP_r$, ${}^g\dot{P}_r$, ϕ, and v_ϕ.

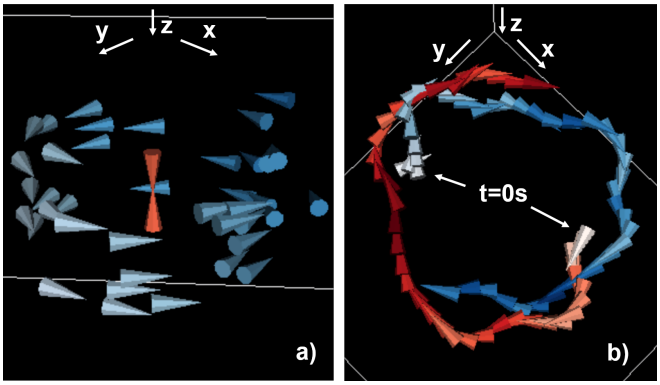

Fig. 8. A snapshot of 50 simulated `BlueBots` orbiting a common red center (left), and the trace of a red and a blue simulated `BlueBot` orbiting each other (right). The multi-robot orbiting looks much like fish milling about. The two-robot orbiting works without any common center.

experiment with the `BlueBot` and used identically in simulation. The animation of the simulated trajectory, done with `BlueAnimat`, strongly resembles the video of orbiting with the `BlueBot`. Superimposed snapshots from experiment and simulation are shown in Figure 7.

We went on and added more simulated `BlueBots` orbiting around a fixed and visually detectable center (see Figure 8 a). Their behavior looked much like fish milling about. They even collided occasionally, which schooling fish do all the time. For the sake of robot health, we added a local Lennard-Jones potential field for collision avoidance (Figure 3). We set the target distance greater than the neighbor detection radius, resulting in purely repelling behavior.

Next, we simulated two `BlueBots` orbiting around each other, i.e., around a common dynamic and virtual barycenter (see Figure 8 b). In physics, this is famously known as the two-body problem. Examples include a satellite orbiting earth, two stars orbiting each other (a binary star), or an electron orbiting an atomic nucleus. The two-body problem could be extended to an n-body problem, for which we have not found a simple solution in time.

People orbit each other as well, for instance when dancing the waltz. We envision the *robo-waltz*: a multi-agent experiment, in which each `BlueBot` has to find a dancing partner, synchronize and form a pair with this partner, desynchronize as a pair from all other pairs, and waltz in two-body fashion.

`BlueSim` has demonstrated its potential to test novel ideas for collective behaviors in simulation, and transfer control strategies to real robot experiments with `BlueBots`. We expect an optimized version of `BlueSim` to facilitate greatly the investigation of feasible algorithms for collective behaviors with `BlueBots`.

F. Discussion of BlueSim

`BlueSim` reflects the current state of work with the `BlueBots`. Its perception model, for instance, does not consider the three LEDs mounted on each `BlueBot`, which are actually perceived. The parsing of observed LEDs into triplets of LEDs belonging to individual `BlueBots` has to

be solved first, to then integrate the solution into BlueSim. Another open problem are reflections on the water surface, through which a neighbor can appear twice in the field of view of a BlueBot.

Furthermore, the experiments reported in this paper assumed that the distances to neighbors could be inferred at all times. Aggregation and dispersion might work without this assumption, relying solely on the direction of neighbors. A comparison between behaviors using direction only, using direction and distance, or even using direction, distance, and relative angle could be interesting in the future. The relative angle between two BlueBots has not been used so far, neither has inter-robot communication. Communication is enabled in BlueSim and could allow for more sophisticated behaviors in the future.

BlueSim could become an interesting simulator to test collective behaviors with robots other than the BlueBot in environments other than water. The architecture of the simulator would remain the same while the perceptual and dynamics models would be adjusted to suit another robot.

To this end, BlueSim was carefully designed to be usable by other people than its creators. My co-authors provide a first example. They used an earlier version of BlueSim for their investigations presented in this paper.

The current version of BlueSim is available on github and includes instructions for installation operation. All results presented in this Section can be reproduced by running the respective jupyter notebooks. A "BlueSim-Docs"-folder with "index.html" as a main file has the complete documentation of BlueSim and is in the supplementary materials of this paper.

IV. Imposter Fish

Turing Learning presents a promising approach for the creation of biomimics. In this approach, observations of an ideal model are used to train black box replicas. Previous work with Turing Learning has been in simple reactive models with 3 possible inputs. In this section, we present an application of the Turing Learning method that allows us to learn dispersion and aggregation behavior in simulated schools of fish. The learned swarm-level behaviors are significantly more complex than prior behaviors learned via this method: actions of fish depend on relative positions of their neighbors, rather than only on the value of a binary or trinary input.

A. Learning Procedure

Turing Learning is a method to learn the behavior of artificial agents from observation of natural or artificial systems. In this method, as specified in [5], we generate *genuine* and *counterfeit* data samples. The *genuine* data samples come from the ideal system we wish to mimic. The *counterfeit* data comes from the replica systems. In each epoch of learning, the classifiers are asked to discriminate between the *genuine* and *counterfeit* data samples. The procedure for Turing Learning, elaborated in [5], is specified in Algorithm 3.

Algorithm 3 Turing Learning Procedure

```
initialize population of M models and population of N classifiers
while termination criterion not met do
  for all classifiers i ∈ {1,2,...,N} do
    obtain genuine data samples (system, classifier i)
    for each sample, obtain and store output of classifier i
    for all models j ∈ {1,2,...,M} do
      obtain counterfeit data samples (model j, classifier i)
      for each sample, obtain and store output of classifier i
    end for
  end for
  reward models (r_m) for misleading classifiers (classifier outputs)
  reward classifiers (r_c) for making correct judgments (classifier outputs)
  improve model and classifier populations based on r_m and r_c
end while
```

We worked with a population of $N = 100$ candidate classifiers and $M = 100$ candidate replica models. In each epoch, we ran a 15 second simulation of a school of 25 fish. All fish in a given simulation followed the same model. We thus had 25 genuine data samples in total, and 25 counterfeit data samples for each candidate model. The quality of a candidate replica model j was given by:

$$r_m(j) = \frac{1}{N}\frac{1}{25}\sum_{i=1}^{N}\sum_{k=1}^{25} m_{ikj} \tag{1}$$

where $m_{ikj} = 1$ if classifier i wrongly categorized data sample k of model j (i.e. classified model j as an ideal agent), and $m_{ikj} = 0$ otherwise.

The quality of a classifier is measured by its ability to distinguish between genuine and counterfeit data samples. The quality of a candidate classifier i is given by:

$$r_c(i) = \frac{1}{2}(\text{specificity}_i + \text{sensitivity}_i) \tag{2}$$

The *specificity* of a classifier refers to the fraction of genuine data samples it correctly categorizes, while the *sensitivity* of the a classifier refers to the fraction of counterfeit data samples it correctly categorizes. Formally, the *specificity* is given by:

$$\text{specificity}_i = \frac{1}{25}\sum_{k=1}^{25} a_{ik} \tag{3}$$

where $a_{ik} = 1$ if classifier i correctly categorizes genuine data sample k, and $a_{ik} = 0$ otherwise. The *sensitivity* is formally given by:

$$\text{sensitivity}_i = \frac{1}{25}\frac{1}{M}\sum_{j=1}^{M}\sum_{k=1}^{25}(1 - m_{ijk}) \tag{4}$$

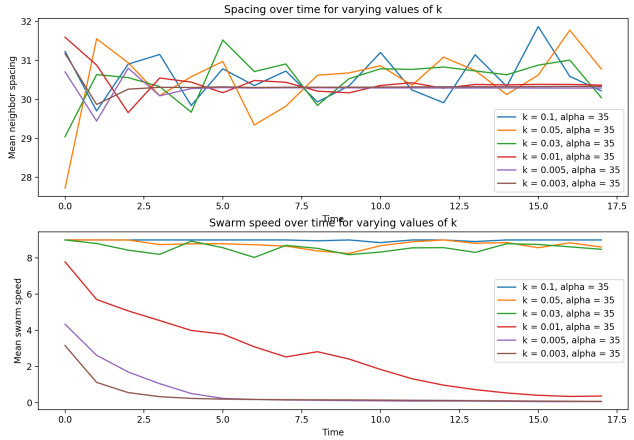

Fig. 9. k_{ar} should be chosen such that a stable equilibrium is quickly reached from an initial distribution. Here, the initial spread of the swarm was 60 units, and σ was set to 35.

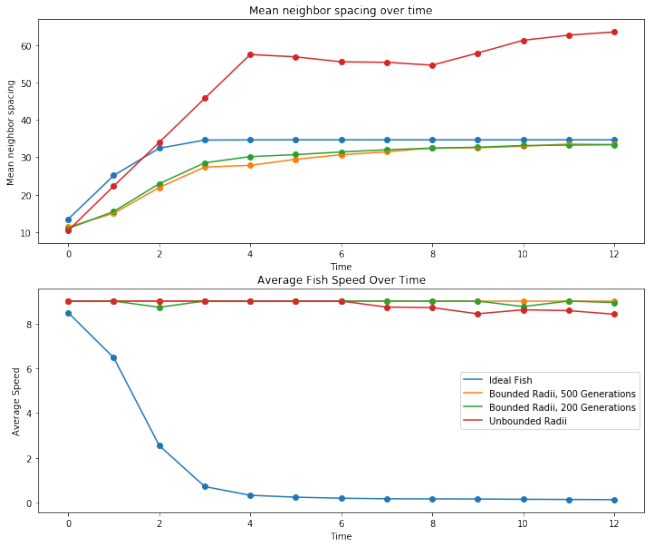

Fig. 10. Comparison of various models for learning dispersion. Allowing the classifier to learn radii of neighbor detection led to fish learning the same average spacing as the ideal fish when those radii were bounded. However, neither model for learning showed an ability to hit equilibrium and stop moving.

We used the covariance matrix adaption evolutionary strategy (CMA-ES) to improve models [13].

B. Ideal Fish Model

Turing Learning requires imitation of some ideal model. We chose the model in [7] which can be used for aggregation, dispersion, cohesion, and more complex behaviors. We looked only at the simplest behaviors: aggregation and dispersion in 2 dimensions. In our use of this model, the

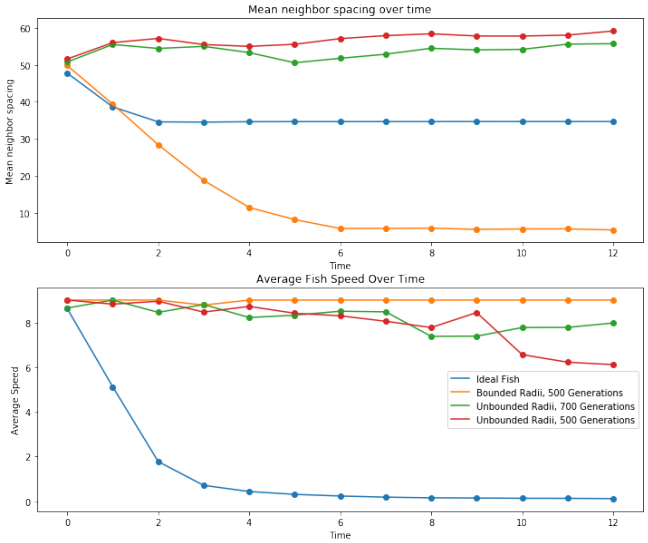

Fig. 11. Comparison of various models for learning aggregation. When the classifier learned radii of neighbor distances that were bounded, the fish excessively aggregated. When said radii were unbounded, the fish did not sufficiently aggregate. Neither model for learning showed an ability to hit equilibrium and stop moving.

time evolution of a fish i is given by

$$\vec{v}_i(t) = \sum_{j \in \Omega_i(t)} k_{ar} sgn(\|\vec{r}_{ij}\| - \sigma)(\|\vec{r}_{ij}\| - \sigma)^2 \frac{\vec{r}_{ij}(t)}{\|\vec{r}_{ij}(t)\|} \tag{5}$$

$$\vec{r}_{ij}(t) = \vec{x}_j(t) - \vec{x}_i(t) \tag{6}$$

$$\Omega_i(t) = \{j \mid \|\vec{r}_{ij}(t)\| < r_{comm}\} \tag{7}$$

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \vec{v}_i(t)\Delta t \tag{8}$$

Essentially, this means fish will move towards neighbors further than σ away, and away from neighbors closer than σ. k_{ar} was chosen to be 0.003 via parameter tuning (see Figure 9). The communication radius r_{comm} was set to 100 to mimic the expected eventual application in the `BlueBot`.

C. Replica Fish Model Design

The replica fish contained the same capabilities as the ideal fish, including the same communication radius and speed limitations. As in the ideal fish, the replica fish observed the relative position of all neighbors — other fish within the communication radius. At each time step, the replica fish's new velocity was set equal to a summation of a function of each neighbor's relative position. This function was the learned function. It was modeled as a simple multi-layer perceptron, with 2 inputs, a single hidden layer with 3 nodes, and 3 outputs. All nodes also contained a bias. The activation function was the logistic sigmoid. The input was the relative x- and y-position of a single neighbor. The outputs are the direction and speed the fish should move given this neighbor's position, appropriately scaled. The weights of this perceptron were learned during the evolutionary process.

D. Classifier Design

The classifier was implemented as a simple recurrent neural network (RNN), with a single hidden layer consisting

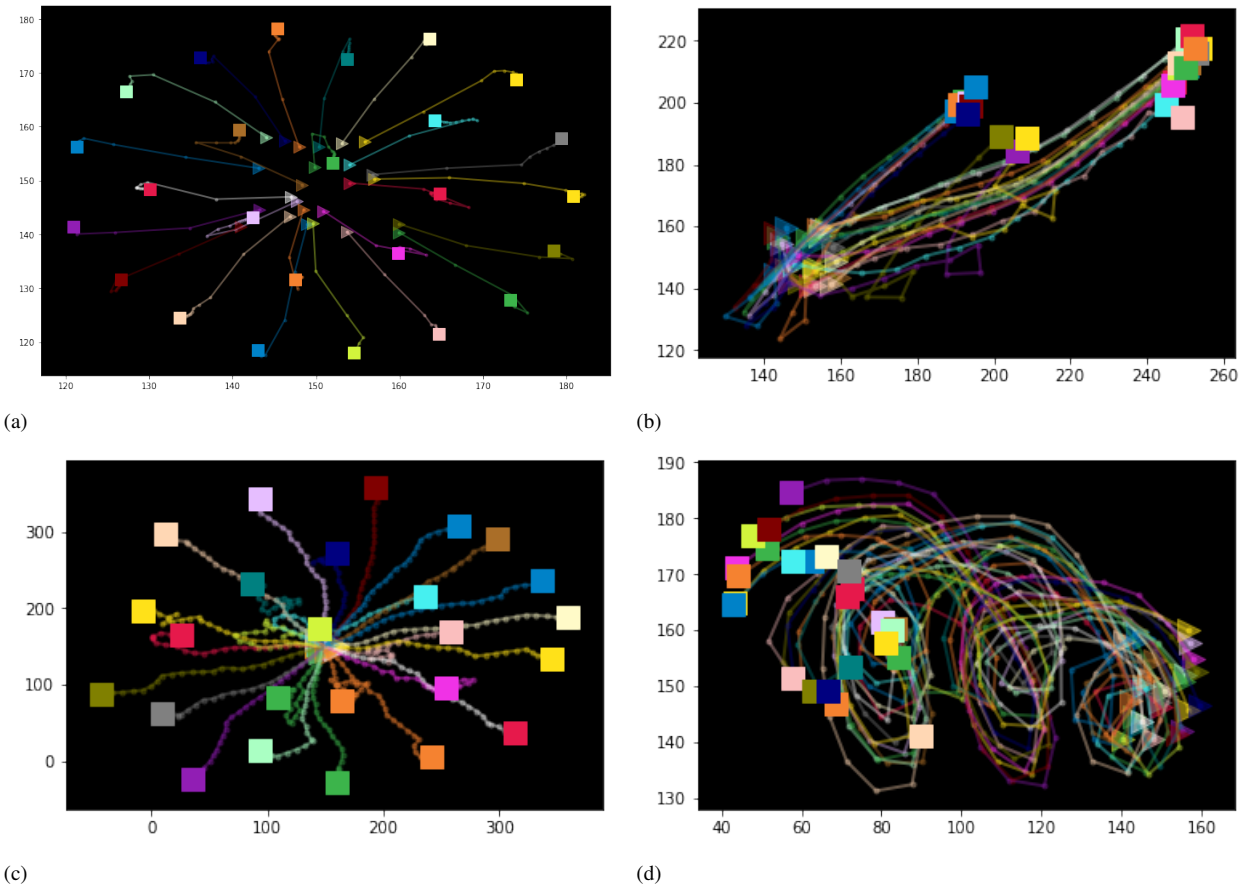

Fig. 12. a) Dispersion with ideal model, b) Dispersion with bounded radii at 490 generations, c) Dispersion with unbounded radii at 200 generations, d) Dispersion with bounded radii at 200 generations. The triangles are where the fish begin. Their trajectories are then traced and marked at each time step, and their position after 15 seconds marked with a square.

of 5 nodes, a bias for each node and a single memory layer. The net had a single output: the classification of an agent as ideal or replica. The inputs to the classifier were varied across trials. The possibilities consisted of a fish's speed over time, a fish's angular velocity over time, and a fish's number of neighbors within 1 or 2 radii, with the magnitude of the radii learned alongside the network weights. In this final case, we recorded the distance of all neighbors of a fish. We then took the radius learned by the classifier, and asked how many neighbors were within this radius at each time step. The number of neighbors was ultimately what was given to the classifier as input.

E. Conditions in Learning Experiments

We attempted to learn aggregation and dispersion behavior. The ideal agent includes aggregation and dispersion in a single algorithm: changing the initial spread will determine if the swarm aggregates or disperses. We chose in learning to separate these behaviors. In aggregation, fish move closer together. In dispersion, they spread out. Thus, we believed our classifier would need to be given some way to determine if a swarm was expanding or contracting over time for learning to occur. Our results supported this – we found that using number of neighbors in a learned radius as an input to the classifier led to the most successful learning. We ran trials under 2 conditions: one in which the classifier learned 2 radii, and where these radii could take any value from -40 to 300 (the same values that could be learned for weights in the RNN). We termed this condition the unbounded condition. In the second condition, we limited the radii to take only values between 0 and 300, termed the bounded-radii condition. We saw the most success in learning dispersion with a bounded-radii input to the classifier.

F. Results in Dispersion and Aggregation Trials

We ran two types of successful trials in learning dispersion. In dispersion, the fish are initialized with a swarm spread of 20. In the ideal fish, σ is 40, leading to an equilibrium average neighbor distance of approximately 35. We ran the bounded radii learning condition to 500 generations, and the unbounded radii condition to 200 generations. In the unbounded radii experiment, the fish model with the highest fitness dispersed well, but continued dispersing over time, slowing down but not stopping. The bounded radii fish also continued to move over time, but kept an average distance between neighbors consistent with the ideal condition. See comparisons of swarm speed and neighbors distance for learning and ideal conditions in Figure 10. To keep neighbors

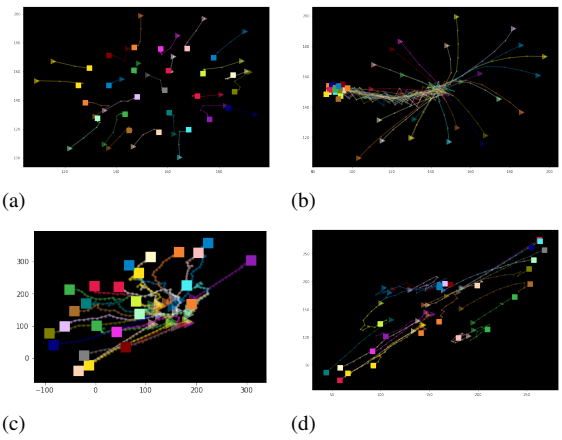

Fig. 13. a) Aggregation with ideal model, b) Aggregation with bounded radii at 500 generations, c) Aggregation with unbounded radii at 500 generations, d) Aggregation with unbounded radii at 700 generations. The triangles are where the fish begin: their trajectories are then traced and marked at each time step, and their position after 15 seconds marked with a square.

at a consistent distance, the dispersing fish learned in the bounded radii condition engaged in interesting migration behavior, visible in Figure 12.

In aggregation, the fish were initialized with a spread of 100. In the ideal fish, σ was once again set to 40, causing aggregation. We again learned aggregation behavior with bounded and unbounded radii. Here, we found that the fish with unbounded radii did not aggregate sufficiently, and the fish with the bounded radii aggregated too much. See Figure 11 for more information on speed and spacing comparisons and Figure 13 for visualizations of fish trajectories in simulation.

In all our learning conditions, the classifiers learned various radii they then used to determine the number of neighbors in a given time step. neighbor. When bounded to positive values, these radii seem intuitively useful. However, in the unbounded cases, the classifiers learn a negative radius. This result is puzzling, as there are no neighbors inside a negative radius, so this radius will not provide information to the classifier. See the radii over all aggregation and dispersion trials in Table I.

G. Imposter Fish Reach Equilibrium

In both aggregation and dispersion learning, learned models were moderately effective at learning correct spacing, but did not reach an equilibrium and halt movement. We also attempted learning by providing trajectory information to the classifier. We found no evidence of trajectory information (linear and angular speed over the simulation) leading to efficient learning of dispersion. Trajectory information in addition to 2 learned radii was equally ineffective: the fish learned to move in a single straight line. We did, however, find that providing the classifier with the number of neighbors in learned radii and the linear speed over time led to replica fish that learned quickly to stop. Unfortunately, these fish did not disperse or aggregate sufficiently before stopping.

	μ_1 Rad. One	σ_1 Rad. One	μ_2 Rad. Two	σ_2 Rad. Two
Dispersion, Unbounded, 200 Gen	-5.9	3.2	2.8	1.6
Dispersion, Bounded, 500 Gen	47.7	4.4	66.4	3.7
Dispersion, Bounded, 200 Gen	35.1	5.7	69.4	8.2
Aggregation, Unbounded, 700 Gen	-26.1	10.9	9.5	0.8
Aggregation, Unbounded, 500 Gen	-17.6	7.0	8.5	0.8
Aggregation, Bounded, 500 Gen	54.8	39.4	161.6	39.4

TABLE I

IN THE UNBOUNDED CASES, ONE OF THE TWO RADII LEARNED IS NEGATIVE. THIS RADIUS WILL NOT PROVIDE USEFUL INFORMATION TO THE CLASSIFIER, AS NO FISH ARE WITHIN A NEGATIVE RADIUS. INTERESTINGLY, THE AGGREGATION CONDITIONS LEARN WIDER RADII, SUGGESTING THE CLASSIFIER MAY BE ESSENTIALLY CHECKING FOR FISH WITHIN THOSE RADII. IN CONTRAST, IN DISPERSION, WITH SMALLER RADII, THE CLASSIFIER MAY BE LOOKING FOR THE ABSENCE OF FISH FROM ONE OF THE RADII OVER TIME.

See suplementary material for visualizations of simulations in these trials. Further work in tweaking the inputs to the classifier seems promising to have replicas learn both the dispersion and the equilibrium behavior.

H. Limitations of Current Learning Framework

Learning was moderately successful with some notable limitations. We found that inputting to the classifier the number of neighbors in a learned radius was most successful in learning aggregation and dispersion behavior, where success was measured by an average neighbor distance graph most similar to the ideal model. Including the fish's linear speed over time led the model fish to learn to stop swimming very fast, but dispersion and aggregation prior to halting was poor. Providing only linear and angular speed to the classifier led to poor classification fitness, even after many rounds of trials and poor imposter fish. Providing linear speed, angular speed and 2 radii also did not improve learning.

Future work will focus on tweaking additional inputs to the classifier so that it considers both fish-level and swarm-level characteristics. In this work, we found that the learned radii was a promising but not sufficient measure of swarm level behaviors. Consider aggregation: we found that in the unbounded radii aggregation trial, the classifier learned a single positive radius of approximately 9. We hypothesized that the ideal fish did not have neighbors closer than this: thus, it was a good discriminator between the real and replica. However, the replica agents instead moved away from each other, as they merely needed to be at least 9 units apart to trick the classifier. In future work, may also attempt to merge

Algorithm 4 Global Formation Algorithm

Require: Number of fish, Intended formation, Relative position of neighbors
  $m \leftarrow map\_generator(No.Fish)$, generate global map
  while m not empty **do**
    $position \leftarrow$ closest position in map.
    $distance\_closest\_position \leftarrow distance(position)$ distance to closest map position
    $nearest\_neighbor \leftarrow nearest(rel\_pos, closest\_position)$
    $distance\_nearest \leftarrow distance(nearest\_neighbor, position)$

    if $distance\_closest\_position \leq distance\_nearest$ **then**
      $move \leftarrow position$
      return move
    else
      $m = m - position$
      $neighbors = neighbors - neighbor\_at(position(n))$
      End if m or neighbors is empty and return [0,0]
    end if
  end while

Algorithm 5 Local Shape Formation Algorithm

Require: Number of fish, Intended formation, Relative position of neighbors
  $possible\_moves \leftarrow possible\_moves(reference\_neighbors)$
  while possible_moves not empty **do**
    $position \leftarrow$ closest possible move.
    $distance\_closest\_position \leftarrow distance(position)$
    $nearest\_neighbor \leftarrow nearest(rel\_pos, closest\_position)$
    $distance\_nearest \leftarrow distance(nearest\_neighbor, position)$

    if $distance\_closest\_position \leq distance\_nearest$ **then**
      $move \leftarrow position$
      return move
    else
      $m = m - position$
      $neighbors = neighbors - neighbor\_at(position(n))$
      End if m or neighbors is empty and return [0,0]
    end if
  end while

learned fish-level behavior that was significantly dispersive or aggregative with behavior that including halting after initial movement.

V. Schooling Formations

This section of the project focused on achieving different formations through fish's decentralized movement decisions. To achieve this, we decided to implement two types of algorithms. One in which the fish had a map of the intended overall target formation and another in which the fish made movement decisions to form regular geometric patterns with its closest neighbors. We called these algorithms *Global Map Formation* and *Local Shape Formation* respectively. The details of both of these algorithms is given in the following sections.

A. Fish Formation Algorithms

Global Map Formation

In the first two attempted formations, the fish have a global map of the intended target figure. The neighbor furthest from the fish given the intended orientation is selected as a leader (e.g. if intended orientation is left to right, fish with largest x-value will be selected). The overall figure will be formed around this leader. The global map held in memory by the fish is transposed to start the formation at the leader's position. Then the fish ranks the neighbors it can visibly see and adopts the designated spot given such rank. If the spot is taken by another fish then it selects another available spot in the map. If there is no spots available the fish will remain in place. When another fish is closer to the target position the fish also changes a to a different target position. This way we avoid collision between neighbors attempting to reach the same space.

We selected two formations from regularly occurring sequences: *Fibonacci Sequence* and *Triangle Pyramid Sequence* to test this algorithm. We generate the global map of the formation given the number of fish in the school and mapped them in 2D space . For the initial trials of this algorithms we gave the fish a global view of the tank without occlusion. Algorithm 4 details the process for achieving these formations.

Local Shape Formation

The *Local Shape Formation* algorithm focuses on achieving local shapes within the `BlueBot's` k closest neighbors. The fish can only make movement desisions based on their closest k neighbors. For this instance, Algorithm 4 was modified to define the possible moves of the fish as those moves that would allow the fish to form the geometric shape with its closest neighbors.

We decided to test this algorithm using regular geometric shapes as well as shapes inspired by naturally occurring molecular forms. In the *Triangular Formation* attempt, we determine the possible target positions of the current fish so that it is able to generate an equilateral triangle of *side length* = l. For the *Trigonal Planar Formation* trial, we decided a list of possible target positions and the fish decides based on that. There is one fish that takes the middle position and the other fish form 120° angles around it. The decision of which fish will take what position is made depending on the number of fish within the fish's radius. This shape is very susceptible to the initial position of the fish and how close the fish are of each other. One possible solution to improve the implementation of this shape is to have a flag that signals whether a fish is already part of a shape. This process can be recreated on the physical world trough the communication capabilities of the `BlueBots`.

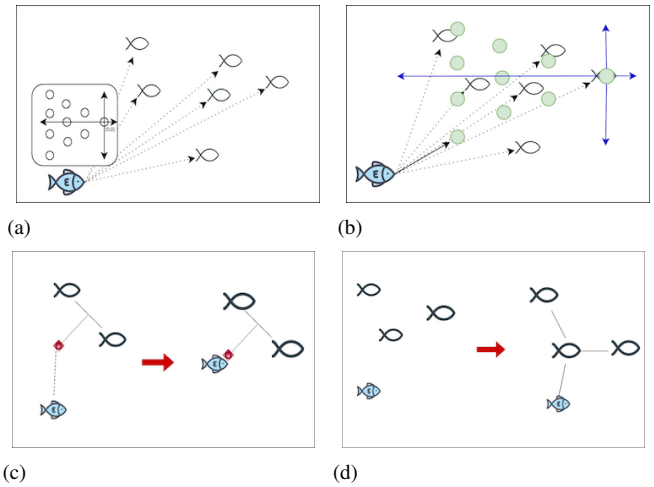

Fig. 14. (a) The fish has global view of the other fish in the tank and knowledge of the global intended formation. (b) Fish transposed global intended formation to be centered with the leader fish. (c) Fish makes a local movement decision based on its two closest neighbors.(d) Fish makes a movement decision based on its three closest neighbors.

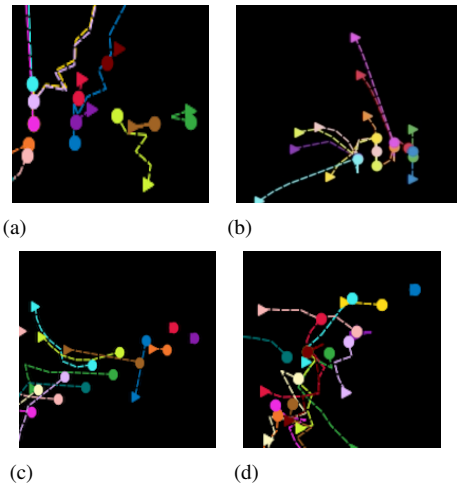

Fig. 15. (a) Fibonacci Map Global Formation. (b) Global Pyramid Formation. (c) Local Triangle Formation, no orientation.(d) Trigonal Planar Formation.

B. Fitness Function

To determine whether a fish has reached a comfortable position or needs to move to a different position, we defined a *fitness-function* to asses the fitness level of individual fish. The fitness level of a fish is a function of distance to the closest target position and the distances of neighbors that are within a radius of the fish. It is defined as follows:

$$F(f_i) = C(f_i, neighbors) + \min(D(f_i, t_i))$$

$$C(f_i, neighbors) = \sum_{i=0}^{NR} tolerance$$

$$D(f_i, t_i) = \begin{cases} tolerance & : T = \{\} \\ min(distance(n_i, t_i)) : \forall t_i \in T \end{cases}$$

,

where $F(f_i)$ if the fitness measurement of fish f_i, $C(f_i, neighbors)$ is a measure of how clustered the area is given the neighbors within a specified radius. NR is the set of fish that are within a radius `r` of the f_i's current position. $D(f_i, t_i)$ is either the distance to the closest available target position or a penalty factor if there are not available target positions. T is the subset of available target positions.

The value of C was chosen to penalize the fish if there is more than one fish on the same position. This prompts the robot to move and cover the entire map if this specific spot is already occupied.

C. Results

Both, Algorithm 4 and Algorithm 5 were used with two different formations. For Algorithm Algorithm 4 we decided to try the *Fibonacci Sequence Formation* and the *Triangular Pyramid Formation*. As mentioned in the previous section, the fitness level of these algorithms was given by the sum of the distances between each fish and its intended position in the final formation. The overall fitness of the formation over time was averaged over *t=5* trials for each value in *tolerance = [0.25,0.50,0.75,1.0, 1.5,2]* for a duration of 30 seconds and 15 fish. The results of this experiment are shown in Figure 16. It is visible that although increasing the tolerance level initially improves the *fitness-level* after a certain level it worsens the formation. In Figure 16 we can see how *tolerance=2* has a higher error than almost all other tolerance levels used for both algorithms.

For Algorithm 5, a *Triangle Based Shape Formation* and a *Trigonal Planar Based Shape Formation* were implemented. For both of these cases, the overall error of the formation over time was averaged over 3 trials with a duration of 30 seconds and 15 fish. The result for each *tolerance =[0.1,0.3,0.5,0.7, 0.9]* values used is synthesized in Figure 17. The general trend is that the *fitness_level* decreases over time which indicates that the fish are reaching more stable formations. However, this could be an effect of the fitness function definition. Although this function provides a measure of the closeness to achieving the desired formation, there are other parameters that could be incorporated into this equation so that it provides a better measure for fitness. For the global approach, the measurement of the fitness function was less susceptible to changes on the tolerance level.

D. Discussion

Both algorithms were able to come to the predetermined formations the majority of the time. In the global algorithms, it is easier to see the target formation whereas in the local algorithm it is harder to asses whether a final formation is reached. There are several aspects of the implementation of these algorithms that have room for improvement. For example, the collision avoidance mechanism of the algorithms can be improved. At the moment, it is handled by a target conflict avoidance function that checks whether another neighbor is closest to the `BlueBot's` intended target. If this is the case then the `BlueBot's` changes targets to the next closest possible target. The issue with this method is that there is

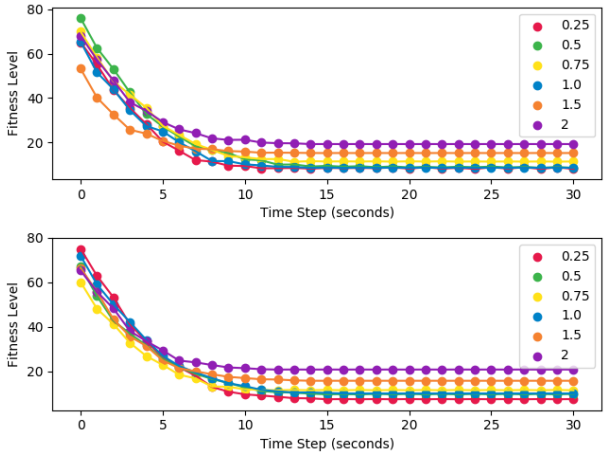

Fig. 16. (Top) Fitness Level of School for Fibonacci Formation over time for different `tolerance` levels. (Bottom) Fitness level of School for Triangle Pyramid Formation over time. Each color represents the tolerance level used in the trial.

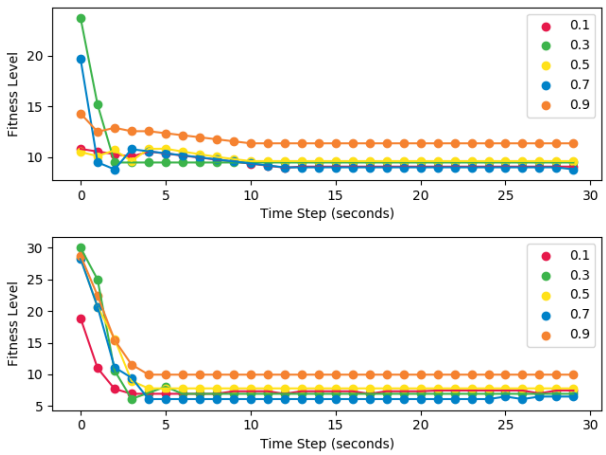

Fig. 17. (Top) Fitness Level of School for Triangular Formation over time for different `tolerance` levels. (Bottom) Fitness level of School for Trigonal Planar Formation over time. Each color represents the tolerance level used in the trial.

a possibility that it does not take into account the velocity of the fish and that the fish is strictly altruistic (if another fish is closest to the target than I am I will choose another target). Other parameters can be incorporated in this decision to make it more realistic.

Furthermore, the fitness function of the fish can be improved greatly. At the moment it only takes into account the distance to the target position but does not offer an overall measure of whether all spots are filled in the target map or how regular it is. It would be interesting to integrate other measures to this function in order to improve formation fitness. Some algorithm specific improvements are detailed in the sections below.

Global Algorithm

The main limitation of the global algorithm is that it relies on the `BlueBot` having a global view of all of the fish in the school. This is not a feasible option for physical experiments due to the hardware limitations of the robots. Nevertheless, this algorithm offers the possibility of generating a global map of any size which means that it is possible to generate block formations by restricting the field of view of the `BlueBots`. Formations that are independent of school spread radius would be able to form (e.g. square grids etc.)

Local Algorithm

The forms achieved through this algorithm are more organic and there is not a predetermined global shape. The triangular algorithm shows a decrease in the error function as time passes meaning that the formation is becoming more regular. For the trigonal planar shape it is harder to see whether there is a formation being reached or not. One way to improve the overall results for this shape is to have better guidelines for the fish to choose which position within the shape it will take. It is possible to implement other types of figures with this algorithm, however, due to time constraints this option was not explored.

VI. Conclusions and Future Work

We modelled the perceptual system and dynamics of `BlueBot` and created a realistic three dimensional simulator. Preliminary experiments on variations in the field of view during aggregation and dispersion suggest that neither the blind spot in the back of `BlueBot` nor occlusions by neighbors impair `BlueBot's` performance significantly. We modelled aggregation and dispersion with attractive and repulsive forces based on the Lennard-Jones potential. We further reproduced an orbiting experiment carried out with a real `BlueBot` in simulation, using the same control code. The simulation and the real robot experiment looked very similar, validating our dynamics model. Using this model, we designed new behaviors such as multiple robots milling about a common center, or two robots orbiting each other. Our simulator has demonstrated great potential to pre-test novel ideas for collective behaviors and thus shorten experimental time with `BlueBots` in the water.

We adapted `Turing Learning` to evolve behaviors for imposter fish. This adaption demonstrated promise in creating mimics that could execute desired swarm-level aggregation and dispersion. We found that counting number of neighbors in a learned radius provided useful information to classifiers trying to distinguish between replicas and ideal agents. We also found that providing a classifier linear speed over time led the classifier to use this information to distinguish between the ideal fish, which generally halted over time, and replica fish, which eventually also showed this behavior. In future work, we will investigate additional characteristics that provide useful information to the classifier: a better classifier leads to imposters that must more fully mimic the ideal fish to succeed. Our work with radii and speed suggests that future trials could explore various weightings of these two inputs. Basic trials with just these inputs were insufficient in teaching imposter fish to be effective mimics, suggesting the classifier may still have insufficient information to be successful.

Through the use of the local and global algorithms, we were able to achieve different predetermined formations. The results confirmed that parameters such as tolerance and visual field radii had a large impact on whether the school can come into formation. This is true specially for the local algorithms. Whether a desired central figure is achieved depends on how large the visual field of the fish is and whether the fitness function prevents a fish from attaching exclusively to a set of neighbors. As mentioned in the previous section there is a lot of room for improvement in various areas of this part of the project. First changing the fitness function so that it takes into consideration other parameters and gives a more accurate, and standardized measure of the completion of a formation is an immediate project. This will help our in understanding to what extent the algorithm has reached formation without visual inspection. Then, other collision avoidance algorithms must also be considered. Although there is an implementation that accounts for collision avoidance, the simulations show that the fish are still likely to collide with each other. Furthermore, in order for this part of the project to become useful, we have to implement similar algorithms in the new 3D simulator. One idea was to base the formations in 3D space on molecular shapes. Once different formations are achieved in 3D space, we can use the `BlueBot's` tail beat frequency raste in order to estimate the energy consumption for each different formation.

`BlueSim`, `Imposter Fish`, and `Schooling Formations` offer a comprehensive approach to learn from the behavior of living *FISH* and engineer the behavior of robot collectives. We foresee that the *F*ormation of *I*ntelligent *S*warming *H*abits with `BlueBots` will soon be possible.

Project Contributions

We all helped each other conceptually in asking meaningful questions, framing our problems, and debugging errors.

Florian developed `BlueSim`, including a three dimensional perception and motion model of the `BlueBot` and all related experiments. Moreover, he started `BlueAnimat`, an animation tool for realistic visualizations of simulated results. He did not implement milestones 6-9 from the project proposal. 8 and 9 are still open problems that have to be solved conceptually before they can be incorporated into `BlueSim`. 6 and 7 are solved and trivial given 9, but meaningless without 9. Florian's contributions are found in Section III.

Katherine developed the Imposter Fish. This work included developing a framework for learning and all learning experiments. She ultimately found that existing frameworks for Turing Learning were not effective in learning fish school behavior and spent most of the project time exploring various modifications to classifier inputs: as such, she did not reach her final milestone, which was to learn behavior for a replica fish with capabilities different from real fish. Katherine's contributions are found in Section IV.

Magaly implemented two different algorithms that resulted in fixed formations of the fish. In the first algorithm, the fish had a global view of the tank and moved according to a predetermined mental map transposed to start at a determined leader fish. This algorithm is able to generate a `Triangle Formation` and a `Fibonacci Formation`. On the second algorithm, the fish had only a local view of the tank and made decisions only based on their nearest neighbors. In this algorithm the overall formation is more organic but the shapes between the fish remain accordingly. Due to the time constraints of the project, she was not able to implement an energy function or translate her work into the 3D simulator generated by Florian. Magaly's contributions can be found in Section V.

We agreed not to disclose any of the materials presented in this paper publicly since they might be used in future publications.

References

[1] 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.

[2] D. Weihs, "Hydromechanics of fish schooling," *Nature*, vol. 241, no. 5387, p. 290, 1973.

[3] F. Berlinger and F. Lekschas, "A distributed and self-organized network of mobile underwaterrobots for collective search and sampling in dynamic environments," *CS262 - Distributed Systems*, 2018.

[4] W. Li, M. Gauci, and R. Groß, "A coevolutionary approach to learn animal behavior through controlled interaction," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 223–230.

[5] ——, "Turing learning: a metric-free approach to inferring behavior and its application to swarms," *Swarm Intelligence*, vol. 10, no. 3, pp. 211–243, 2016.

[6] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH computer graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.

[7] M. Delight, S. Ramakrishnan, T. Zambrano, and T. MacCready, "Developing robotic swarms for ocean surface mapping," pp. 5309–5315, 2016.

[8] M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen, "Evolution of collective behaviors for a real swarm of aquatic surface robots," *PloS one*, vol. 11, no. 3, p. e0151834, 2016.

[9] S. Hauert, J.-C. Zufferey, and D. Floreano, "Reverse-engineering of artificially evolved controllers for swarms of robots," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 55–61.

[10] S. Marras, S. S. Killen, J. Lindström, D. J. McKenzie, J. F. Steffensen, and P. Domenici, "Fish swimming in schools save energy regardless of their spatial position," *Behavioral ecology and sociobiology*, vol. 69, no. 2, pp. 219–226, 2015.

[11] M. Rubenstein and W.-M. Shen, "Scalable self-assembly and self-repair in a collective of robots," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 1484–1489.

[12] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents part i: dynamic topology," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2. IEEE, 2003, pp. 2016–2021.

[13] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.