2023 American Control Conference (ACC)
San Diego, CA, USA. May 31 - June 2, 2023

Imitating Swarm Behaviors by Learning Agent-Level Controllers

Ibrahim Musaddequr Rahman[1*] Stanford White IV[2*] Katelyn Crockett[3]
Yu Gu[4] Dimas A. A. Dutra[4] Guilherme A. S. Pereira[4]

***Abstract*— A main challenge in swarm robotics is the unknown mapping between simple agent-level behavior rules and emergent global behaviors. Currently, there is no known swarm control algorithm that maps global behaviors to local control policies. This paper proposes a novel method to circumvent this problem by learning the agent-level controllers of an observed swarm to imitate its emergent behavior. Agent-level controllers are treated as a set of policies that are combined to dictate the agent's change in velocity. The trajectory data of known swarms is used with linear regression and nonlinear optimization methods to learn the relative weight of each policy. To show our approach's ability for imitating swarm behavior, we apply this methodology to both simulated and physical swarms (i.e., a school of fish) exhibiting a multitude of distinct emergent behaviors. We found that our pipeline was effective at imitating the simulated behaviors using both accurate and inaccurate assumptions, being able to closely identify not only the policy gains, but also the agent's radius of communication and their maximum velocity constraint.**

I. Introduction

Research on robotic swarms has gained considerable attention for applications where robustness, scalability, and flexibility are desired, as it allows simple robots with limited information to execute tasks far beyond their combined individual capabilities. A fundamental challenge that is faced by engineers when designing swarm controllers, however, is the nontrivial mapping between local agent-level policies and the swarm's global emergent behavior [1]. This makes the design of individual behaviors that result in an intended collective behavior a main challenge in swarm robotics, for which there is currently no general solution [2]–[4]. Consequently, controller development is usually done in an iterative, trial-and-error process, guided by designer intuition, experience, and skill [4]–[6]. In this paper, we propose a method to design the controllers of individual members of a robotic swarm by learning the observed modes of motion of another group of robots or animals.

Swarm robotics is inspired by the behavior of social animals like ants, termites, birds, and fish [1], [5], [6]. An influential early approach to reproduce the flocking motion

This work was supported in part by the National Science Foundation Research Experience for Undergraduates Program, Award #1851815, and West Virginia University Statler College.

[*]The first two authors contributed equally to this work. [1]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. [2]Department of Mechanical Engineering, University of Mississippi, University, MS, USA. [3]Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV, USA. [4]Department of Mechanical and Aerospace Engineering, West Virginia University, Morgantown, WV, USA. Emails: iamr@umich.edu, sewhite5@go.olemiss.edu, kec0035@mix.wvu.edu, yu.gu@mail.wvu.edu, dimas.dutra@mail.wvu.edu, and guilherme.pereira@mail.wvu.edu.

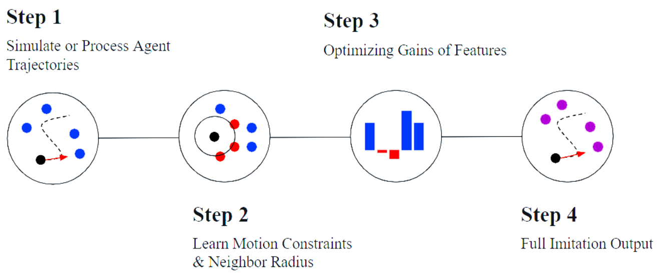

Step 1
Simulate or Process Agent Trajectories
Step 2
Learn Motion Constraints & Neighbor Radius
Step 3
Optimizing Gains of Features
Step 4
Full Imitation Output

Fig. 1: Learning Pipeline for Swarm Imitation.

of large numbers of bird-oid objects (*Boids*) for computer graphics was based on postulating the basic behaviors followed by each individual [7], such as cohesion, alignment, and separation. This idea, inspired by an understanding of natural swarms, has been applied to design the guidance of robotic swarms for various tasks [8].

More recently, several approaches that learn the underlying individual controller policies from motion data of existing swarms have been proposed. This allows imitation learning of decentralized robot control from centralized demonstrations [9], [10] and learning robot motion patterns from observations of animal motion [11], [12]. Most existing solutions for data-driven swarm imitation utilize Neural Networks. Graph Neural Networks (GNNs) have been utilized to parse the system of communicating networks, where each agent is a node with edges connecting to each agent in its communication radius [9], [10]. Such GNNs frequently also have layers of Recurrent Neural Networks (RNNs) to account for any action in the controllers that relates to previous states [11], [13]. A more recent approach utilizes Knowledge-based Neural Ordinary Differential Equations (KNODEs) to accomplish swarm imitation [12]. This approach allowed for the embedding of prior knowledge about the local agent controller and allows a simple neural network to approximate the local agent behavior in conjunction with it. However, similar to other Neural Network-based controllers, agent-level controller policies may not be well-defined on the entire input space, which may result in unexpected behavior of individual agents.

Swarm controllers have also been modeled as a linear combination of specific features of an agent's neighborhood, such as polynomials of the distance between agents or to a centroid, and Bayesian optimization has been used to train a parameter matrix to model the behaviors of a swarm with the same features [14]. However, this approach may be limited in that it requires a complete understanding of which nonlinear features are important to a given swarm before training.

A useful imitation paradigm for swarms would therefore be capable of quantitatively decomposing swarm behavior into distinct controller policies which have a clear interpretation. A decomposition of this nature would provide insight into the relationship between local and global behaviors. In this work, we introduce an imitation pipeline capable of decomposing swarm behavior by observing the trajectories of agents in a swarm. Figure 1 presents the steps of the method, which is characterized by treating agent controllers as a set of policies that can be combined to determine the agent's change in velocity over a given time step.

In our framework, the policies are linearly combined and then saturated, making the controller linear in the parameters over a range of inputs. Over a subset of the data linear regression can then be used as a first pass to identify the strength (gain) of each policy, and global optimization methods can be employed to refine the results. In parallel, an algorithm is employed to identify the radius of communication between swarm agents, and the maximum observed velocity is recorded directly after processing the trajectory data. Due to the relative simplicity of each step in the proposed pipeline, the results of the decomposition are understandable and insightful at each step. By applying our method to both simulated and natural swarms, we demonstrate that it is capable of learning maximum observed velocity and radial communication constraints between swarm agents as well as correctly identifying and assigning weight to individual controller policies.

This work provides several unique contributions: First, we provide a pipeline for swarm imitation that is explainable at every stage, allowing for a straightforward interpretation of the results. Additionally, our pipeline is the first that is capable of inferring the radius of communication between local agents. Our methodology has been proven effective by application to both simulated and vision-tracked natural swarms. The pipeline has a potential impact in many areas where decentralized swarms hold advantages over centralized swarms where communication links cannot be broken. This includes the use of large drone swarms in search and rescue and military applications.

The rest of this paper is organized as follows: the problem solved by our method is defined in the next section; Sect. III presents details of the proposed method, which is evaluated in sections IV and V using simulated and real-world data, respectively; Finally, conclusions are presented in Sect. VI.

II. Problem Definition

The objective of this work is to design a simple algorithm to learn the agent-level controllers of an observed swarm in order to imitate its emergent behavior. We consider 2D swarms that fulfill a certain set of constraints. First, they exhibit the defining characteristic of a swarm: the controller for each agent is decentralized. Additionally, this controller is dependent only on the actions of neighbors within a certain radius, r, and the local environment (e.g., boundaries). We assume that this radius is unknown. We also assume that each agent i is a single integrator controlled by it's velocity v_i. Furthermore, we assume that v_i depends only on the instantaneous positions and velocities of the agent's neighbors. Thus, there is no memory over time.

Finally, the agents motion is constrained to a bounded environment and is subjected to a maximum observed velocity.

III. Method

The fundamental idea behind our method involves modeling the local agent controller $\mathbf{v}_i$ as a linear combination of policies $\boldsymbol{\theta}$, where each policy is a nonlinear function of the velocities and positions of neighboring agents and environmental factors such as boundaries. The unsaturated controller of agent i is then written as:

$$\mathbf{v}_i(k, \boldsymbol{\theta}) = k \cdot \boldsymbol{\theta}\,, \tag{1}$$

where k is a vector of parameters that weigh each policy, which we assume is common to all the agents in the swarm. The policies are constrained to a set that contains social policies: cohesion (C), separation (S_{re}), alignment (A), rotation (R), and steer-to-avoid (S_t); and non-social policies: boundary separation (W_s), boundary steer (W_t), and inertia (I). These policies are illustrated in figures 2 and 3.

Among the social policies, cohesion, separation, and alignment are consistent with their original definitions in Boids [7]. To represent additional behaviors exhibited by several swarms, we also included rotation, which causes agents to orbit each other in either a clockwise or counter-clockwise direction depending on the sign of the weight, and steer-to-avoid, which ensures that agents dodge each other by changing their direction to avoid collisions. To compute each social policy, the agents consider the adjacent neighbors within a radius, such that the neighborhood set $\mathcal{N}_i$ of agent i is defined as

$$\mathcal{N}_i = \{ j | \, r \geq ||\mathbf{p}_j - \mathbf{p}_i|| \}\,,$$

where $\mathbf{p}_i$ and $\mathbf{p}_j$ represent the 2D positions of agents i and j, respectively. The social policies for agent i are then defined as:

$$\begin{aligned}
C &= \frac{1}{|\mathcal{N}_i|} \sum_{j=1}^{|\mathcal{N}_i|} \mathbf{p}_j - \mathbf{p}_i\,, \\
S_{re} &= \sum_{j=1}^{|\mathcal{N}_i|} \frac{\mathbf{p}_i - \mathbf{p}_j}{||\mathbf{p}_j - \mathbf{p}_i||^{e+1}}\,, \\
A &= \frac{1}{|\mathcal{N}_i|} \sum_{j=1}^{|\mathcal{N}_i|} \mathbf{v}_j\,, \\
R &= C^{\perp}\,, \\
S_t &= \frac{1}{||\mathbf{p}_c - \mathbf{p}_i||} \mathbf{v}_i^{\perp}\,,
\end{aligned} \tag{2}$$

where $\mathbf{p}_j$ and $\mathbf{v}_j$ are the respective 2D position and velocity of each agent in agent i's neighborhood, $\mathbf{v}_i$ is agent i's current velocity, $\mathbf{p}_c$ is a future colliding agent in the path of the current velocity, e is an integer number, and the $\perp$ symbol denotes the orthogonal vector on the counterclockwise side.

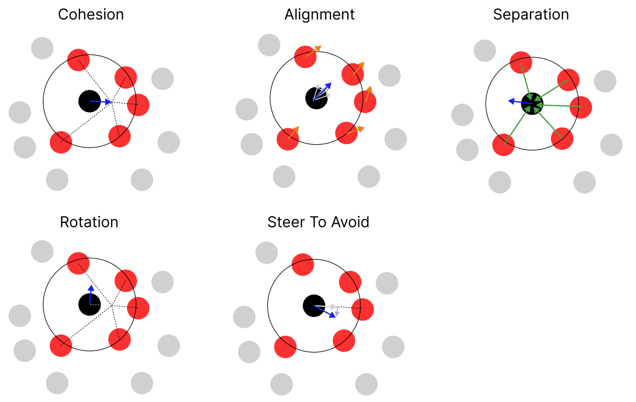


Fig. 2: Agent Social Control Policies

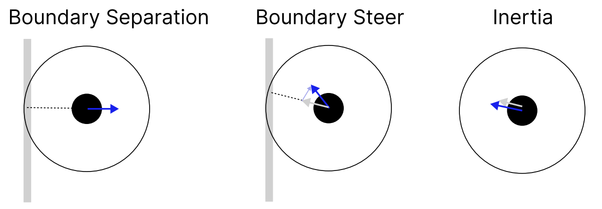


Fig. 3: Agent Non-Social Control Policies

The non-social policies include boundary separation and boundary steer, which cause agents to separate and steer away from the enclosure walls. Additionally, inertia causes agents to continue moving with their current velocity. Those policies are defined as:

$$\begin{aligned} W_s &= \frac{\mathbf{p}_i - \mathbf{d}}{||\mathbf{d} - \mathbf{p}_i||^3} , \\ W_t &= (\mathbf{d}_c - \mathbf{p}_i)^\perp , \\ I &= \mathbf{v}_i , \end{aligned} \tag{3}$$

where $\mathbf{d}$ is the displacement vector to the closest point on the boundary, and $\mathbf{d}_c$ is the displacement vector from the boundary along the current direction of motion.

In order to accomplish imitation of an observed swarm, a multi-step process is proposed in this paper (see Fig. 1). First, the trajectory data of the known swarm is collected and processed to extract and estimate positions and velocities of each individual agent. For example, in practical situations, this can be done by using a computer vision based tracking system [15]. With the positions and velocities of the known swarm, observational data such as the communication and motion constraints are inferred. The relative strengths (gains) of each control policy can then be determined with linear regression and refined with nonlinear optimization, which also enforces the maximum observed velocity constraint. By combining each of the learned parameters, a complete imitation can be created, and several metrics are used to measure its validity. The next subsections present details on this process.

A. Learning Communication and Motion Constraints

The agents' maximum observed velocity can be determined by direct analysis of the available data (i.e., taking the maximum of all velocities). When working with real data that may have anomalies due to sensor noise (e.g., vision), the 99th percentile of the velocity can be taken instead.

To determine the radius of communication, we minimize the 1 step prediction error of a controller trained with first pass linear regression at a given radius. Because the space is discontinuous and nondifferentiable, the following search algorithm is used: In the first iteration, a number (e.g., 15) of random radii are selected from the search space, and for each of them a cost value is computed. To create the next set, the best solutions (e.g., top 6) are selected, and random variations of these are used to replace the rejected radii. This process is repeated for some number of iterations (e.g., 10 times), and the best solution in the final set is selected.

To compute the cost for each radius, we begin by calculating the influence of each policy (equations (2) and (3)) for each agent at each timestep, based upon the estimated radius. The linear regression step of our imitation pipeline (shown in Sect. III-B) is then applied at this specific radius, outputting a set of weights for each policy. The cost of this radius and its respective weights are then evaluated as the sum of the error for a 1-step trajectory prediction:

$$C(r) = \sum_{i=0}^{n} \sum_{t=0}^{T} \|\mathbf{v}_i^*(t) - \mathbf{v}_i(\mathbf{k}, \boldsymbol{\theta}_i(t,r))\| \tag{4}$$

where n is the number of neighbors (agents inside the current radius r), $\mathbf{k}$ is the vector output gains from the linear regression, $\mathbf{v}_i^*(t)$ is the observed velocity of agent i at time t, and $\boldsymbol{\theta}_i(t,r)$ is the vector of policies for agent i computed at time t with the current radius r. Because agent motion has a velocity constraint on it, the behavior is not linear to the policies when at maximum velocity. As such, datapoints at maximum observed velocity are ignored in this step.

B. Learning Policy Weights

Our method requires a two step optimization. Due to the linearized form of (1), linear regression is initially performed to identify the initial weights (gains) of each policy in the controller. In this step, datapoints at maximum observed velocity are ignored. To consider the maximum velocity saturation, the initial gains are refined through a nonlinear optimization step that includes the maximum observed velocity constrained datapoints. This constraint is encoded as:

$$\boldsymbol{\sigma}(\mathbf{v}) = \begin{cases} \mathbf{v} & |\mathbf{v}| \le v_{\max} \\ \frac{\mathbf{v}}{|\mathbf{v}|} v_{\max} & |\mathbf{v}| > v_{\max} \end{cases} \tag{5}$$

where $\boldsymbol{\sigma}$ is the constrained version of $\mathbf{v}$.

For the optimization problem, our proposed cost function considers the difference between the change in velocities of agents in the observed swarm and the learning swarm:

$$L(k) = \frac{1}{n} \sum_{i=0}^{n} \sum_{t=0}^{T} \|\mathbf{v}_i^*(t) - \boldsymbol{\sigma}(\mathbf{v}_i(k, \boldsymbol{\theta}_i(t)))\| \ . \tag{6}$$

The optimization problem is then formally posed as:

$$k = \arg\min_{k} L(k)\,. \tag{7}$$

To minimize $L(k)$, vector k obtained with linear regression is utilized as the first-pass guess for Sequential Least Squares Programming (SLSQP) in the SciPy library [16], [17]. SLSQP is a nonlinear optimization method that minimizes an input function by iterating through various estimated inputs until it returns approximate zero values. A nonlinear optimization algorithm that does not use boundaries was desired because the scale of the policy weights should not be assumed. Unlike many other nonlinear optimization methods, SLSQP operates without any user-provided boundaries around the estimated input values, which automates the algorithm and removes any initial guess bias. For this reason, it was chosen as the nonlinear optimization method most fit for the complete pipeline's purpose.

C. *Imitation metrics*

Several numerical metrics are considered in analyzing the efficacy of an imitation. When the policy weights are known for both the original and imitated swarms (only possible in controlled simulations), a direct comparison of their values can be made. In order to compare the emergent behaviors entirely based on trajectory data, however, we consider three separate metrics. First, the Mean Absolute Error (MAE) between the predicted velocity change and the real velocity change across all timesteps is used to show how similarly individual agents acted compared to agents in the original swarm. This is the same loss function in (6), used in the nonlinear optimization step of our pipeline. So:

$$\text{MAE} = L(k)\,. \tag{8}$$

Notice that the difference of predicted and real velocity change is equivalent to 1-step trajectory prediction, as the problem is modeled as first order.

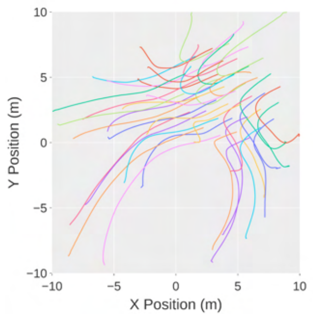
We also consider the metrics of Average Minimum Distance (AMD) and Average Velocity Difference (AVD), as proposed by [12]. The AMD to a neighbor measures how cohesive agents in a swarm are over time. For swarms exhibiting pure flocking behavior, for example, AMD should converge to a fixed value. AMD is given by:

$$\text{AMD}(t) = \frac{1}{n}\sum_{i=1}^{n} \min_{j} ||\mathbf{p}_i(t) - \mathbf{p}_j(t)||\,, \tag{9}$$

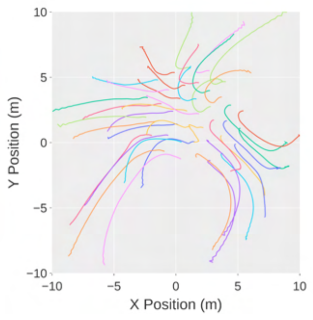
where t is the current timestep, n is the number of agents, and $\mathbf{p}_i$ and $\mathbf{p}_j$ are the positions of agents i and j, respectively. The AVD between swarm agents measures how similar the velocity vectors are across all agents. For swarms exhibiting pure flocking behavior, AVD is also expected to converge to a constant value. It is given by:

$$\text{AVD}(t) = \frac{2}{n(n-1)}\sum_{i\neq j} ||\mathbf{v}_i(t) - \mathbf{v}_j(t)||\,, \tag{10}$$

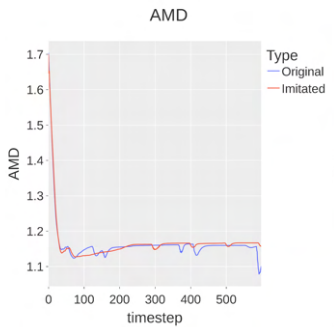
where $\mathbf{v}_i$ and $\mathbf{v}_j$ are the velocity vectors of agents i and j.



(a) Original Swarm



(b) Imitating Swarm



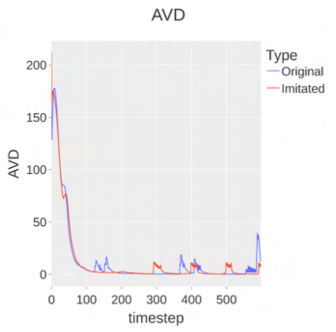


Fig. 4: Imitation of pure Boids exhibiting flocking behavior (MAE: 10625.575)

IV. Simulations

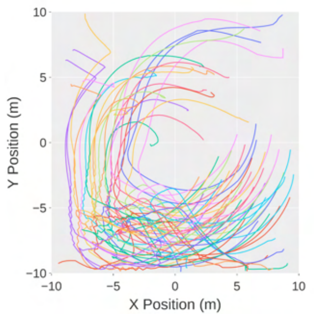
Our pipeline was applied to several different types of both simulated and physical swarms. Videos of such simulations and experiments can be watched at: https://tinyurl.com/acc2023.

In order to verify the efficacy of the pipeline, swarm simulations were created, which can exhibit any behavior derivative of the policy set. Training data was generated consisting of 40 agent, 5 seconds simulations in a 10 m by 10 m enclosure. Simulations were considered with varying maximum velocities and communication radii. Training simulations were generated with Gaussian noise on positions to emulate robotic sensor noise, with a magnitude of 5% of the maximum displacement per timestep. Many short simulations were favored over a single long one, since the transient behavior of the swarm is assumed to be more descriptive of the agent controller policies. In steady-state flocking behavior, agents move in unison, and certain policies are not clearly demonstrated/activated.

A. *Imitating with Same Feature Set*

To verify the pipeline's efficacy upon data that fits its assumed model, in this subsection the set of policies in the learning set consisted only of policies in the simulated swarm, which were assumed to be known.

In our first experiment, we consider a simple, fundamental case of a swarm of Boids with the policies of cohesion (C), separation (S_{r2}), and alignment (A). The weights of these policies were set to $k = [1.00, 1.00, 1.00]$, the maximum velocity $v_{\max}$ was 7 m/s, and the radius r was 5 m. As shown in Fig. 4b, our methodology was effective at imitating the



(a) Original Swarm (b) Imitating Swarm

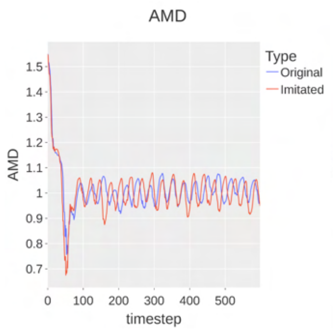
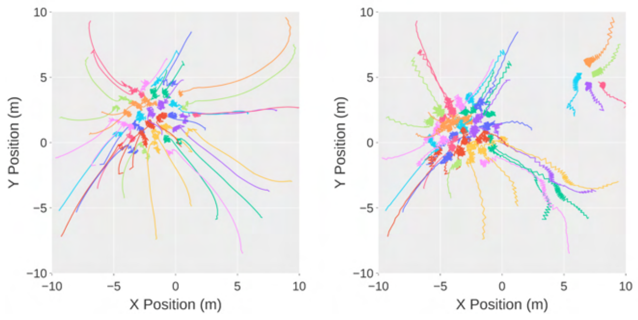


Fig. 5: Imitation of Boids exhibiting flocking behavior with boundary steer and separation (MAE: 7617.865)



(a) Original Swarm (b) Imitating Swarm

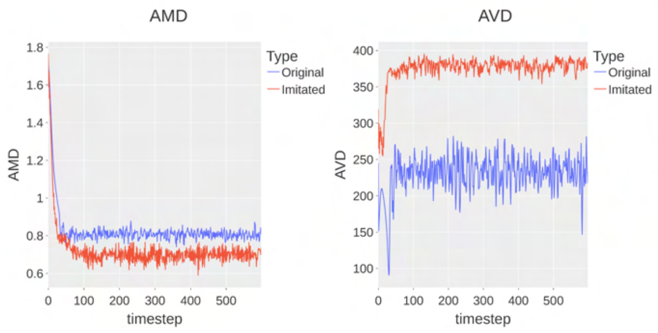


Fig. 6: Imitation of Boids exhibiting flocking behavior with differing kinds of separation and different policy set. (MAE: 28002.142)

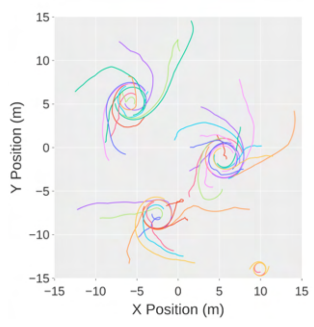
emergent behavior of the swarm. The parameters identified by our method were $k = [1.37,\ 1.36,\ 1.31]$, $v_{\max} = 6.19\,\text{m/s}$ and $r = 4.98\,\text{m}$. By analyzing the trajectories, it is evident that the same distinct flock and direction of motion was preserved in the imitating swarm. Moreover, the AMD and AVD against time plots demonstrated that agents across the two swarms converged to the same average distance and heading. It is notable that AMD and AVD are mostly smooth and overlapped with some minor deviations.

To show the behavior of our method when the swarm includes nonsocial policies, such as separation (W_s) and steering away (W_t) from boundaries and obstacles we created a Boids simulation that, besides C, S_{r2}, and A, includes W_s and W_t policies, in this order. We used $k = [1.00,\ 1.00,\ 1.00,\ 1.00,\ 1.00]$, $v_{\max} = 7.00\,\text{m/s}$ and $r = 5.00\,\text{m}$, the trajectories of which are seen in Fig. 5a. Similarly to the pure Boids case, the pipeline was effective at identifying the parameters with high precision as $k = [0.98,\ 0.99,\ 0.99,\ 1.10,\ 0.94]$, $v_{\max} = 8.02\,\text{m/s}$ and $r = 5.01\,\text{m}$, the trajectories of which are seen in Fig. 5b. The AMD and AVD plots, compared to the pure Boids case without nonsocial controller policies, exhibited a more oscillating behavior. The general trend of the behavior was preserved in the imitation, which in both the AMD and AVD cases oscillated with roughly the same frequency, magnitude, and average value.

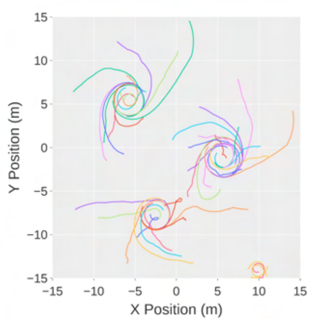
B. Imitating with an Incomplete Policy Set

It is expected that natural swarms exhibit behaviors that are similar to commonly used heuristics but with fundamentally different natures. For example, several different inverse power laws (different e values in (2)) can be used to calculate separation, which all act along the same line of action but are linearly independent. In order to evaluate our pipeline on data whose behavior is not perfectly represented in the feature set, in this section we show simulations generated with controller policies that were not in the set used for learning.

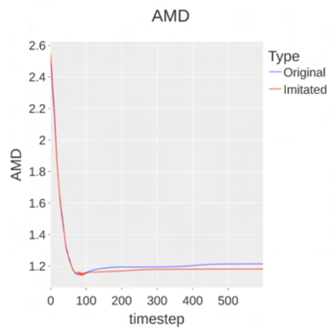
First, we considered a simple case where the imitation pipeline assumed the incorrect inverse power law and a nonidentical set of policies. The trajectories observed in Fig. 6a were generated using the policies C, S_{r6}, and A with $k = [1.00,\ 1.00,\ 1.00]$, $v_{\max} = 7.00\text{m/s}$, and $r = 5.00\text{m}$. On the other hand, the trajectories observed in Fig. 6b were generated using our pipeline with the policies C, S_{r2}, A, and R. The imitated policy set was $k = [2.10,\ 1.13,\ 1.37,\ -0.46]$, $v_{\max} = 7.83$, and $r = 4.96$. The values of the imitated policy set demonstrate the principle that the emergent behavior is not entirely dependent on the absolute values of the controller policies and that it can be obtained with an imperfect set of policies. The trajectories observed in Fig. 6 exhibit the same cohesive behavior across both the original and imitated. The policy set k of the imitated swarm accomplishes this behavior, however, by increasing the value of both C and S_{r2} in order to compensate for a different mode of separation introduced by S_{r6}. Moreover, the introduction of rotation R into the policy set resulted in its introduction into the policies of the imitated swarm, but it was compensated by the increased alignment A. The observed differences can be viewed in both the trajectories, AMD plots, and AVD plots. The trajectories in Fig. 6b show a mild oscillating behavior



(a) Original Swarm



(b) Imitating Swarm



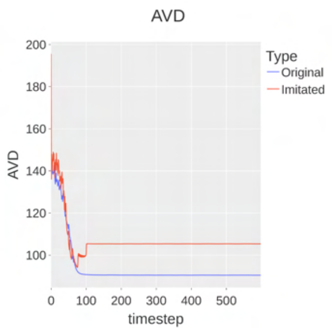
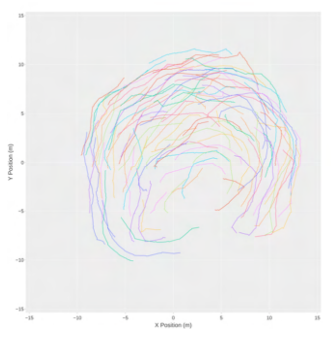
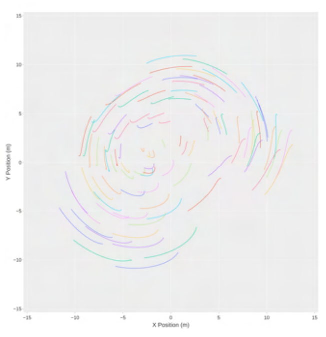


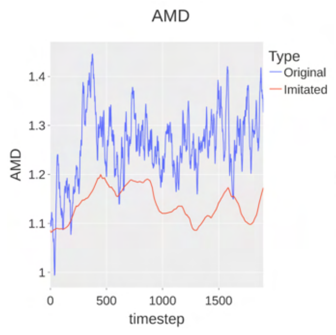
Fig. 7: Imitation with a partially overlapping policy set. The original swarm had C, R, and S_{r2}, and the learning algorithm with C, A, S_{r2}, S_t,R, I, W_s, and W_t. (MAE: 4760.562)



(a) Fish Initial Trajectories



(b) Fish Imitated Trajectories



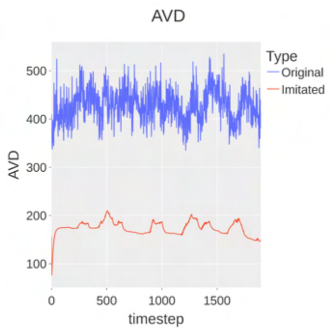


Fig. 8: Imitation of a fish swarm through the pipeline. (MAE: 73858.041)

compared to those in Fig. 6a. That effect exacerbated the differences observed in the AVD plot, especially, since the oscillation results in slightly different headings despite the agents traveling in the same general direction. The difference in the separation parameter resulted in a small, yet noticeable difference in the AMD plot, where the imitated swarm tended to flock slightly more closely than the original swarm.

In another simulation, we show that our method allows the imitation of swarm behaviors whose controller policies are a proper subset of the imitation policies considered in the learning algorithm. In this case, the estimated gains for the unused policies of the original swarm should be close to zero. We considered a swarm exhibiting rotational behavior, with the controller policies C, R, and S_{r2}, in that order, with $k = [1.00, 1.00, 1.00]$, $v_{\max} = 5.00\,\text{m/s}$, and $r = 6.00\,\text{m}$. The swarm behavior is shown in Fig. 7a. We then try to imitate the behavior with the complete set of controller policies, C, A, S_{r2}, S_t,R, I, W_s, and W_t. The parameters of the imitated swarm shown in Fig. 7b are $k = [1.37, 0.00, 1.38, 0.00, 1.38, -0.38, -0.02, 0.00]$, with $v_{\max} = 5.15\,\text{m/s}$, and $r = 6.05\,\text{m}$. As expected, the unutilized policy gains returned a value close to zero. The gains that were utilized were slightly overestimated, but as shown by the trajectories, the behaviors of the imitated swarm closely matched the original. Moreover, the AMD has a similar behavior to the original in terms of convergent flocking distance. The AVD converged to a larger value, but the effect on the trajectories was negligible. This result shows that the imitation pipeline is effective at identifying which features are unnecessary to imitate different swarms.

V. Fish Swarm Experiment

Finally, our pipeline was utilized on a real-life swarm. To imitate the swarm behavior, the proposed methodology used data from a 100 guppy fish swarm [18], collected with the TRex vision tool [15]. The used video is 60 seconds long with 1900 viable frames, providing 190,000 points of agent motion data. The natural fish motion consists of short jumps followed by a still period (see video at https://tinyurl.com/acc2023), which is different than the continuous motion model used in our existing controller policies. To filter these discrete jumps into a continuous stream of motion, a low-pass filter was applied to the fish velocities extracted from the video before they were used in training. Due to the unknown nature of an individual fish's motion, we used a two-pass approach to better approximate their set of control policies. The first pass is done with all the features we had available (equations (2), with $e = 2$, and (3)). Then, the policies with larger gains (≥ 0.1) were selected. A final pipeline run was done with only these specific features, which were found to be Alignment (A), Rotation (R), and Inertia (I).

The parameters of the imitated swarm were found to be $k = [0.59, 0.17, 0.37]$, $v_{\max} = 5.65\text{cm/s}$, and $r = 5.08\text{cm}$. The trajectories observed in figures 8a and 8b presented some visual differences. The primary difference was that the initial swarm tended to exhibit more erratic (noisier) behavior compared to the imitation, which was found to be smoother and without sudden changes in motion direction. These differences can be also observed in the AMD and AVD plots. The AMD of the imitation swarm tended to underestimate but was not significantly dissimilar to the original swarm. The AVD plot, however, resulted in a more jarring difference. AVD measures the alignment of the swarm, and the behavior of the original swarm was erratic, so the AVD tended to be high compared to the imitation. It is also notable that the trendlines of both AMD and AVD follow a similar pattern over time, implying similar dynamics in how the overall swarm moves.

A better imitation can likely be achieved through the inclusion of additional controller policies that take into account the erratic behavior of fish. It is worth noting, however, that the purpose of this experiment is not to create a perfect imitation of the neurological processes of the guppies, but rather to decompose their behavior into simple controller policies that are able to be implemented in real robots. The presented imitation accomplishes this using only three simple controller policies, which are enough to preserve the general trends of the swarm's motion.

VI. Conclusion and Future Work

We have introduced an effective, multi-step machine learning pipeline to decompose and imitate swarm behavior. Our algorithm was the first one that explicitly identified the radius of communication and the velocity constraints. Additionally, our algorithm was the first to accomplish swarm imitation without utilizing neural networks, offering greater interpretability and transferability to other problems. Comparatively, we have offered an imitation method that decomposes swarm behavior into understandable controller policies. We have demonstrated the efficacy of our algorithm by applying it to a variety of different swarm behaviors, which were generated both artificially and naturally. Specifically, we have shown near-perfect imitation from swarm trajectory data that fits its assumed model. Furthermore, we have shown the effectiveness of the pipeline on simulated swarm data that does not fit the assumed model as well as on physical swarms that exhibit comparatively complex behaviors. Global swarm behavior can now be decomposed and analyzed at an agent level, which is instrumental in informing agent controller design decisions and understanding the relationship between local and global swarm behaviors.

We did not conduct any stability analysis, however, our model has bounded output due to the velocity constraints. Additionally, the decomposition into separate policies would allow for simpler analysis in future work, as opposed to more complex approaches. Because we frame agent controllers as a combination of social and non-social policies, this method's applicability expands beyond the simple examples shown. To improve the modeling of 2D swarms, such as the school of fish in this paper, more self-interested non-social policies may be considered. More complex situations involving higher dimensional motion, or environmental interactions like pheromone deposition or structure building, could have equivalent policies used to model the motion of other existing swarms such as ants or bees.

This work can be expanded by implementing the learned controllers in robots, utilizing modern machine learning techniques to generate controller policies by isolating policy contributions that cannot be approximated with the existing feature set, and adapting the pipeline to accomplish online imitation learning. These are some of the lines that we consider for future research.

References

[1] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[2] D. Bozhinoski and M. Birattari, "Designing control software for robot swarms: Software engineering for the development of automatic design methods," in *Proceedings of the 1st International Workshop on Robotics Software Engineering*. Association for Computing Machinery, 2018, pp. 33–35.

[3] ——, "Towards an integrated automatic design process for robot swarms," *Open Research Europe*, vol. 1, p. 112, 2021.

[4] M. Birattari, A. Ligot, and K. Hasselmann, "Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 494–499, 2020.

[5] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[6] M. Dorigo, G. Theraulaz, and V. Trianni, "Reflections on the future of swarm robotics," *Science Robotics*, vol. 5, no. 49, p. eabe4385, 2020.

[7] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[8] N. K. Long, K. Sammut, D. Sgarioto, M. Garratt, and H. A. Abbass, "A comprehensive review of shepherding as a bio-inspired swarm-robotics guidance approach," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 4, pp. 523–537, 2020.

[9] S. Zhou, M. J. Phielipp, J. A. Sefair, S. I. Walker, and H. B. Amor, "Clone swarms: Learning to predict and control multi-robot systems by imitation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4092–4099.

[10] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conf. on robot learning*. PMLR, 2020, pp. 671–682.

[11] H. M. Maxeiner, "Imitation learning of fish and swarm behavior with recurrent neural networks," Ph.D. dissertation, Master's thesis, Freie Universität Berlin, 2019.

[12] T. Z. Jiahao, L. Pan, and M. A. Hsieh, "Learning to swarm with knowledge-based neural ordinary differential equations," 2021. [Online]. Available: https://arxiv.org/abs/2109.04927

[13] E. Zhan, S. Zheng, Y. Yue, L. Sha, and P. Lucey, "Generative multi-agent behavioral cloning," *arXiv*, 2018.

[14] S. Jacobs, R. M. Butts, Y. Gu, A. Baheri, and G. A. S. Pereira, "A framework for controlling multi-robot systems using bayesian optimization and linear combination of vectors," 2022. [Online]. Available: https://arxiv.org/abs/2203.12416

[15] T. Walter and I. D. Couzin, "Trex, a fast multi-animal tracking system with markerless identification, and 2d estimation of posture and visual fields," *eLife*, vol. 10, p. e64000, feb 2021. [Online]. Available: https://doi.org/10.7554/eLife.64000

[16] D. Kraft, *A software package for sequential quadratic programming*. Koln, Germany: Berichtswesen d. DFVLR, 1988.

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[18] T. Walter, A. Albi, D. Bath, H. Hugo, F. Oberhauser, M. Groettrup, and D. Mink, "Reproduction Data for: TRex, a fast multi-animal tracking system with markerless identification, and 2D estimation of posture and visual fields," 2020. [Online]. Available: https://doi.org/10.17617/3.4y