Data-Driven Swarm Control Using the Koopman Operator

Final Project for Distributed Optimization (EE424)

Jonathan Bosnich

1 Project Overview

In this final project, I chose to investigate some aspects of multi-agent control. I mainly focused on multi-agent control in the context of swarm robotics, though the methods discussed are applicable to a variety of fields. Because of its interdisciplinary importance, multi-agent control has garnered much interest and academic activity to address fundamental challenges, specifically understanding *network structure* and controlling the networked system in a *distributed manner*. Clearly, this topic is very well-aligned with our course topics. The second ingredient of this project is using data-driven machine learning methods to model the network dynamics and apply distributed control. This is of interest to me because my graduate research has been on data-driven modeling and control of nonlinear systems. Specifically, I have worked on using applied Koopman operator theory as the method and general philosophy for generating and controlling data-driven models. Hence, when I found a paper that used a Koopman-based data-driven method to model multi-agent systems, I knew this project was a perfect fit! Further, I thought of a natural and exciting extension of this work, namely, to apply control to achieve a desired behavior of the closed-loop network dynamics. While I have yet to fully flesh this idea out, I hope to work on it soon! To be clear from the onset, Sections 2-6 are an exposition of existing work, and Section 7 is a description of the future work that I hope to pursue.

2 Motivation

2.1 Multi-Agent modeling and control

Modeling and control of multi-agent systems in a network is an important topic across a variety of fields, including swarm robotics [1], neuroscience [2], and even social sciences [3]. Identifying network topology (e.g., communication topology among agents) and the inter-agent interaction, influence, and forcing is understandably a hard problem to solve. Further, often times one wants not only to derive a model describing a dynamic network of agents but to additionally control the network in a decentralized manner. Decentralized control of a multi-agent system is defined to be that each individual agent computes its own control input based off of its own state and the state information from its neighbors, which is defined by the communication topology of the network. The graph associated with the communication topology must be connected in order for coordinated control of the whole swarm. In real-world scenarios, this graph is rarely the complete graph on N vertices, but rather, some connected subgraph of the complete graph. This is also more efficient in terms of communication and overall computation because having a complete communication graph would be analogous to centralized control with a central computer/oracle broadcasting to every agent.

2.2 Data-Driven modeling and control

The unavoidable reality of real-world engineering is that relevant systems which can be modeled accurately from first principles are vanishingly rare. There are many ways to overcome this modeling inaccuracy, such as trusting your model only over a short time horizon, e.g., what's done in model predictive control. However, with the advent of machine learning and artificial intelligence, using data-driven machine learning algorithms to "learn" approximate models is becoming the go-to modeling methodology. In this area too there are many different data acquisition techniques and machine learning algorithms which are used to create

distinct data-driven modeling and control strategies. One strategy which has gained incredible popularity is called dynamic mode decomposition (DMD), which is closely related to the Koopman operator. This is the data-driven methodology I use in my own research and the methodology which will be used in this project. Section 4 provides a detailed explanation of the Koopman operator and DMD.

3 Swarm Control in Robotics

I will use the modeling and control of a swarm robots as the running example throughout this project. Assume that we have a swarm of N robots, each with their own governing dynamics. Assume that each agent's movement is solely in the plane and that each agent has their own state vector, e.g., agent i has the state $s_i(t) = \begin{bmatrix} x_i(t) & y_i(t) & \theta_i(t) & \dot{x}_i(t) & \dot{y}_i(t) & \dot{\theta}_i(t) \end{bmatrix}^\top$, where $x_i(t), y_i(t)$ are the Cartesian positions in the plane, $\theta_i(t)$ is the heading or direction in which agent i is pointing, and $\dot{x}_i(t), \dot{y}_i(t), \dot{\theta}_i(t)$ are the velocities. The state variables evolve over time according to the equations of motion of each agent. For simplicity, it is common to assume that each agent is completely controllable through the control input $u_i = \begin{bmatrix} u_{acc} & u_{rot} \end{bmatrix}^\top$, where u_{acc} is the forward acceleration of agent i in the direction of its heading, and u_{rot} is the rotational steering to change the heading of agent i. Now, we define the network state vector by $\mathbf{s}(t) = \begin{bmatrix} s_1(t)^\top & s_2(t)^\top & \cdots & s_N(t)^\top \end{bmatrix}^\top$. An example of such a robot swarm can be seen below in Figure 1.

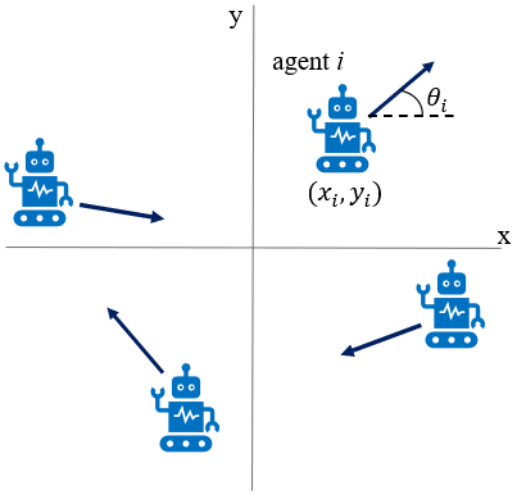

Figure 1: Example of a swarm of 4 robots, each with Cartesian position in the plane and rotational heading.

Next, we will discuss a few common desired formations for a swarm to be in and the control strategies to get them in that formation. Again, the crucial aspect of these control algorithms is that they are decentralized. But first, we must explicitly define the network communication topology in order for the term "neighbors" to make sense.

3.1 Communication Topology

The communication topology is simply that each agent is able to communicate with any other agent if it's within a distance of r away. If an agent is within a distance of r, we draw a edge between these agents representing bidirectional communication, and we refer to these agents as "neighbors". An illustrative toy example on how to create this communication graph is given below in Figure 2.

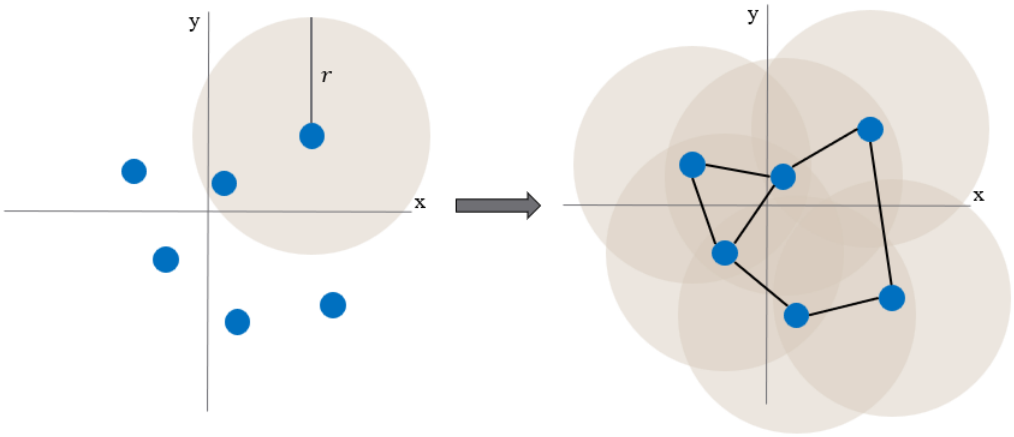

Figure 2: Example of the communication topology amongst six agents.

3.2 Milling Control

Milling behavior occurs when a swarm covers some sort of geometric space according to a distribution defined on that spatial domain. Milling control refers to using decentralized control to drive the swarm to cover a specified spatial distribution. There's a variety ways to do this, both in how a spatial distribution is defined and the decentralized control strategy to achieve coverage of that domain. In [4] the distribution is defined by a human using their finger draw a desired spatial shape on a tablet, where the drawn trajectory is represented as a distribution using a Fourier decomposition, and decentralized ergodic coverage is used to drive the swarm to cover this distribution. Another technique presented in [5] is to create a distribution from images using Legendre moment transformations and then to employ the self-healing first-order distributed control algorithm developed in [6]. In Figure 3 below is an example of this milling control algorithm in action.

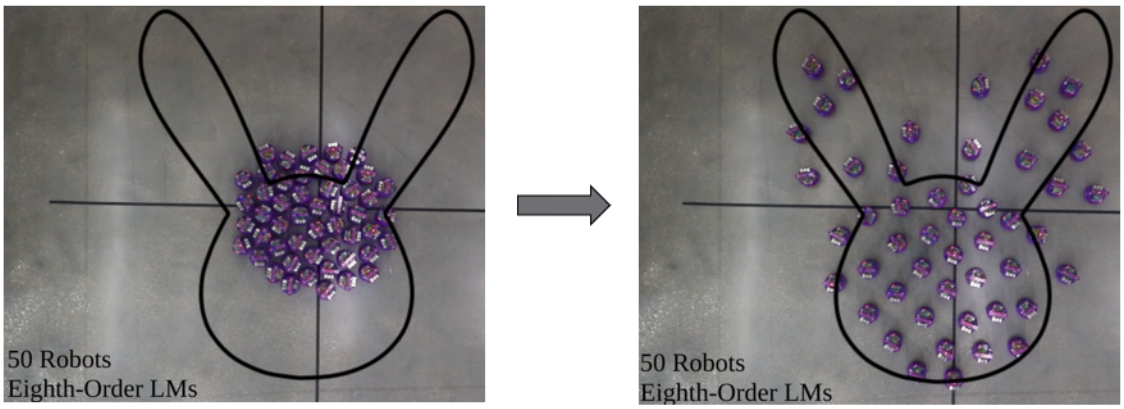

Figure 3: Example of milling control. There is a distribution generated by eighth-order Legendre moments defined over the bunny head. The milling control algorithm drives the agents to roughly cover the distribution. Note: these are screenshots taken from the video supplement for [5].

3.3 Flocking Control

A swarm is said to exhibit flocking behavior when all agents are heading in the same direction. Flocking control is thus using decentralized control with neighbor communication to achieve flocking. Note that this

is in essence a consensus problem. For perfect flocking, we want the heading of agent i to equal the heading of agent j if i and j are neighbors, i.e., $\theta_i = \theta_j$ if $i \sim j$. In practice, this is impossible to achieve and rather we have $\theta_i \approx \theta_j$ if $i \sim j$. This motives the definition of *group polarization* denoted $P(t)$ from [7]:

$$P(t) = \left\| \frac{\sum_{i=1}^{N} v_i(t)}{\sum_{i=1}^{N} \|v_i(t)\|_2} \right\|_2 \in [0, 1], \tag{1}$$

where $v_i = \begin{bmatrix} \dot{x}_i & \dot{y}_i \end{bmatrix}^\top$ denotes the linear velocity of agent i. If $P(t)$ is close to 1, the agents' headings are well-aligned; if $P(t)$ is close to 0, the headings are misaligned.

Next, we discuss two distinct control strategies to achieve flocking, i.e., drive $P(t)$ close to 1.

1. *Neighbor Averaging*: In this approach, each agent communicates with its neighbors and uses a consensus protocol to converge to the average heading amongst its neighbors. This model and protocol is also known as the Vicsek model [8]. An illustrative toy example is given below in Figure 4.

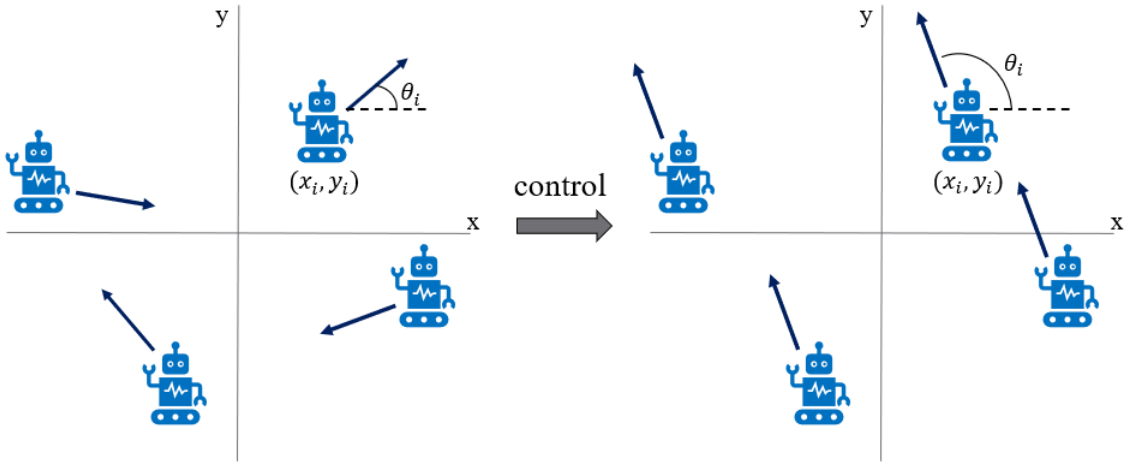

Figure 4: Flocking control using a neighbor averaging shceme.

2. *Leader-Follower*: In this approach, there is a designated leader agent and the other agents must match this leader's heading, as shown below in Figure 5.

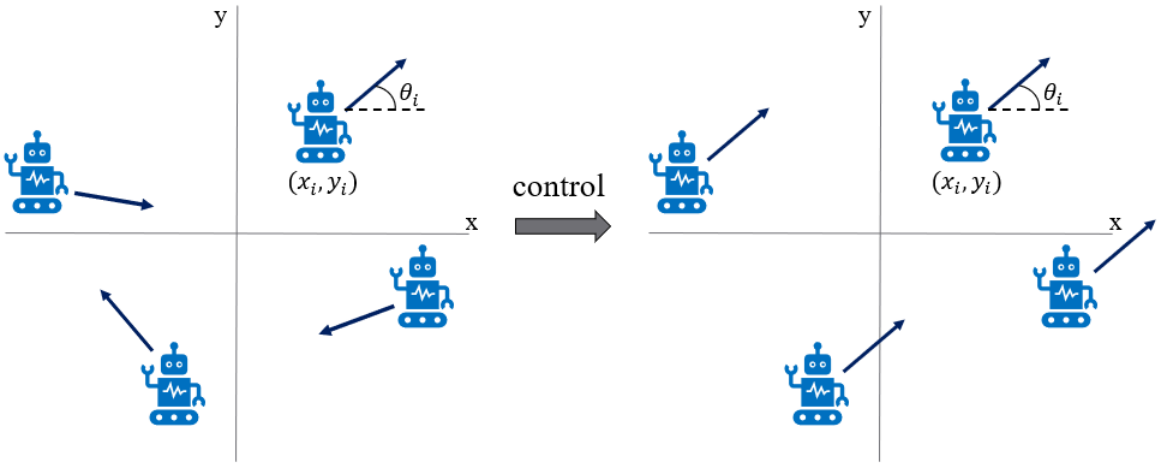

Figure 5: Flocking control using a leader-follower scheme.

Clearly, one can also combine the above two strategies for something in between neighbor averaging and leader-follower; for example, using a neighbor averaging scheme but with weighting the importance of a particular agent (or agents) to be greater than the rest.

4 Koopman-Based Data-Driven Modeling and Control

We now shift our discussion to data-driven modeling. Specifically, we will focus on the class of data-driven models using dynamic mode decomposition (DMD) [9] and its many, many variants, e.g., extended DMD (eDMD) [10], DMD with control (DMDc) [11], robust and verifiable DMD (resDMD) [12], and physics-informed DMD (piDMD) [13], to name a few. What all of these DMD algorithms have in common is that they are fundamentally data-driven approximations of the Koopman operator, which we discuss next.

4.1 Overview of the Koopman Operator

The power of the Koopman operator is that it can embed any nonlinear dynamical system into an infinite-dimensional function space, where functions of the underlying state evolve *linearly*. In short, the Koopman operator linearizes any nonlinear system, however, at the cost of creating an infinite-dimensional linear system.

Briefly, I will discuss the precise mathematical formulation of the Koopman operator. Consider an unforced, discrete-time nonlinear system

$$s_{k+1} = f(s_k), \tag{2}$$

where $f(\cdot)$ is a nonlinear function, and $s_k \in \Omega \subseteq \mathbb{R}^n$ is an n-dimensional state vector. The Koopman operator $\mathbb{K}$ acts on functions of the state $g : \Omega \to \mathbb{C}$ in the following manner

$$[\mathbb{K}g](s_k) = g(f(s_k)) = g(s_{k+1}). \tag{3}$$

Thus, by definition $\mathbb{K}$ linearly evolves functions of the state forward in time. For a schematic illustration, refer to Figure 6 below.

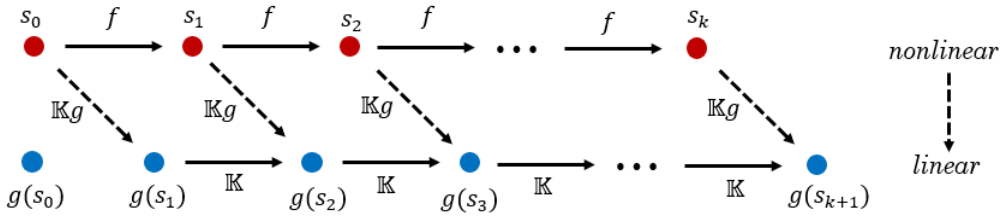

Figure 6: Schematic flow of nonlinear state-space dynamics and linear Koopman dynamics.

The problem is that since we're lifting our state-space dynamics to a function space, which is inherently infinite-dimensional, the Koopman operator is an infinite-dimensional object. In other words, in order to get full reconstruction of the state-space dynamics, we would need our set of functions of the state g to be a full function space basis, e.g., the infinite set of polynomials $\{x^i\}_{i=0}^{\infty}$, which is clearly not practical. Nevertheless, there has been an explosion of applied Koopman operator theory for modeling nonlinear systems. The reason for this is dynamic mode decomposition (DMD), which uses data-driven machine learning methods for finding a finite-dimensional approximation of the Koopman operator, resulting in finite-dimensional Koopman dynamics [1]. For more information on applied Koopman theory in dynamics and control, refer to the review article [14].

[1] While (2) is unforced, external forces/ control inputs can easily by incorporated into the Koopman operator framework and DMD, i.e., DMDc.

5 Dynamic Mode Decomposition

The overall goal of this algorithm is to find a finite-dimensional matrix K that approximately propagates the state forward linearly, i.e., $Ks_k \approx s_{k+1}$. The naive way to solve this problem is quite simple. The first step is to collect data, which are M snapshots of the state vector over time (this could be from a simulation of the dynamics or empirical measurements from a physical experiment). You then construct two data matrices

$$S = \begin{pmatrix} | & | & & | \\ s_1 & s_2 & \cdots & s_{M-1} \\ | & | & & | \end{pmatrix}, \quad S' = \begin{pmatrix} | & | & & | \\ s_2 & s_3 & \cdots & s_M \\ | & | & & | \end{pmatrix}. \tag{4}$$

Thus, finding K such that $Ks_k \approx s_{k+1}$ is equivalent to finding K such that $KS \approx S'$. Finding the best such K can be precisely formulated as a least-squares optimization problem

$$K = \underset{A \in \mathbb{R}^{n \times n}}{\arg\min} \|AS - S'\|_F, \tag{5}$$

which has a closed-form solution

$$K = S'S^+, \tag{6}$$

where $+$ denotes the pseudoinverse. The issue with (6) is that the underlying state-space Ω may be very high dimensional, as is the case in fluid dynamics. Therefore, the data matrices will be very large (especially long) and computing the pseudoinverse will be extremely computationally inefficient.

5.1 The DMD Algorithm

The DMD algorithm offers a computationally efficient alternative to (6). The crux of DMD is that a truncated singular value decomposition (SVD)[2] of the data matrix S is performed and used to only capture the most dominant modes of the approximate Koopman operator K. The algorithm steps are as follows:

1. Collect data and construct data matrices the same way as (4)
2. Compute SVD of S:
$$S = U\Sigma V^*,$$
where Σ is a diagonal matrix of the singular values of S, the columns of U and V are the left-singular and right-singular vectors of S (respectively), and $*$ denotes the complex conjugate transpose.
3. Project K onto the first r most dominant modes[3]:
$$\tilde{K} = U_r^* K U_r = U_r^* S' V_r \Sigma_r^{-1}$$
4. Compute the eigendecomposition of $\tilde{K}$[4]:
$$\tilde{K}\tilde{W} = \tilde{W}\Lambda$$
5. Compute eigenvectors of K:
$$W = S'V_r\Sigma_r^{-1}\tilde{W}$$
6. Finally, we arrive at the DMD approximation of the Koopman operator:
$$K_{DMD} = W\Lambda W^*$$

[2]The singular value decomposition is a generalization of the eigendecomposition.
[3]Σ_r^{-1} is easy to compute since Σ is diagonal.
[4]It can be shown that K and $\overline{K}$ have the same eigenvalues.

In conclusion, just from taking snapshots of system data, we are able to construct a linear approximate system $s_{k+1} = K_{DMD}s_k$. For systems with control input, DMDc performs a very similar procedure. The difference is that you must also collect control input data over the same time snapshots, and the resulting approximate linear system is of the form $s_{k+1} = As_k + Bu_k$

Side note: In DMD, we only measure the state and want to find K that approximately linearly propagates the state forward. In eDMD, we measure the state and *functions* of the state prescribed by the use, e.g., measure s_k, s_k^2, and $\sin(s_k)$. Then we seek a K that approximately propagates this new state vector $\begin{bmatrix} s_k & s_k^2 & \sin(s_k) \end{bmatrix}^\top$ linearly. I'm making this footnote because I realized one could rightly say "why use fancy Koopman operator terminology when DMD doesn't really have anything to do with it". The answer is eDMD is truly a data-driven method for approximating the Koopman operator and DMD is a particular case of eDMD.

6 Multi-Agent DMD

The DMD methodology was recently extended to multi-agent systems in [15], and (unsurprisingly) this algorithm was given the name swarmDMD. The setting is a swarm of N homogeneous agents with communication topology as described in Section 3.1 (i.e., each agent can communicate with any other agent within a radius r). The dynamics governing the motion of each individual agent is unknown, and the inter-agent interaction is unknown. In order to discover the governing dynamics of the whole swarm from data measurements, Hansen et al. use a variation of DMDc, where they are specifically interested in learning the inter-agent interactions by treating them as external forcing/control terms. To do this, the authors make the following changes to DMDc:

(C1) An extended state vector is made by including inter-agent information, such as the relative position d_{ij} between agents i and j. That is, if agent i has m neighbors, agent i's extended state would be

$$z_i(t) = \begin{bmatrix} x_i(t) & y_i(t) & \theta_i(t) & \dot{x}_i(t) & \dot{y}_i(t) & \dot{\theta}_i(t) & d_{i1}(t) & d_{i2}(t) & \cdots & d_{im}(t) \end{bmatrix}^\top.$$

Then, the state and extended state of the whole swarm are

$$\mathbf{s}(t) = \begin{bmatrix} s_1(t)^\top & s_2(t)^\top & \cdots & s_N(t)^\top \end{bmatrix}^\top, \quad \mathbf{z}(t) = \begin{bmatrix} z_1(t)^\top & z_2(t)^\top & \cdots & z_N(t)^\top \end{bmatrix}^\top.$$

(C2) As mentioned before, DMDc learns a linear control model $s_{k+1} = As_k + Bu_k$, where A describes the free dynamics and B describes the influence of an external force. In swarmDMD, the authors make the assumption that $A = I$; that is, they learn a model $s_{k+1} = s_k + Bu_k$. The authors say that they do this so that only the inter-agent interaction/forcing is being modeled by swarmDMD. They don't care about modeling the free dynamics of the agents.

With these changes, the authors perform DMDc to arrive at the swarmDMD approximation of the Koopman operator

$$K_{swarm} = (S' - S)V_r\Sigma_r^{-1}U_r^*, \tag{7}$$

which yield the linear model of the global swarm dynamics

$$\mathbf{s}(t+1) = \mathbf{s}(t) + K_{swarm}\mathbf{z}(t) \tag{8}$$

The authors claim that since the model (8) only takes into account the inter-agent forcing, the operator K_{swarm} "is quite similar to a Laplacian matrix" [15].

6.1 Simulation Result

The swarmDMD algorithm was employed to learn a linear model desribing the dynamics of a swarm of agents following the Vicsek model. Since the swarm is following a Vicsek model, the simulation converges towards swarm flocking, i.e., the swarm ends up pointing in a similar direction. The important simulation

parameters were that the swarm had $N = 200$ agents, there were three different simulations using three different communication radii $r = 0.05$, $r = 0.25$, and $r = 0.5$, the $r = 8$ most dominant modes were used for the truncated SVD in (7), and finally the training period was 5 seconds and post-training prediction test was 5 seconds. Figure 7 taken directly from [15] (minus the titles and legend) summarizes the error in heading and polarization.

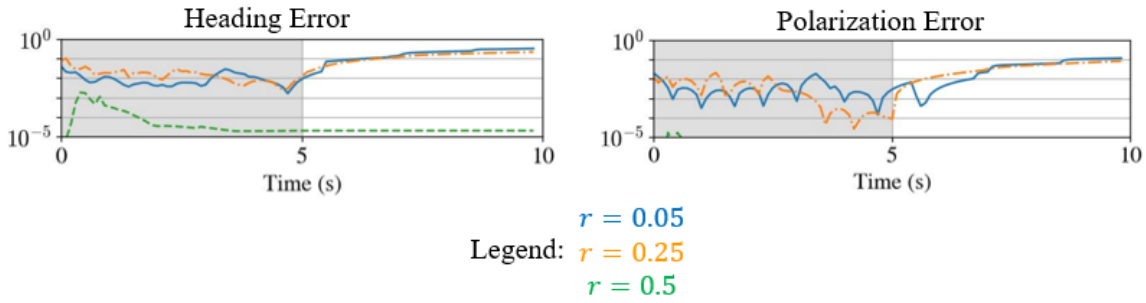

Figure 7: Simulation results for swarmDMD learning a linear model for a swarm following a Vicsek model.

7 My Future Work

I had a few immediate thoughts after reading [15]. The first is that while external control is being taken into account as the inter-agent forcing, it is not being used to drive the swarm to a desired formation or behavior. In the simulation experiment, the training data already exhibits flocking because the swarm trajectories are generated by the Vicsek model. Currently, swarmDMD is exclusively learning swarm dynamics and trying to predict those same dynamics in the future. My research in data-driven control is markedly different in that I use Koopman-based methods to learn the dynamics of a system exhibiting undesirable behavior (e.g., instability) and then synthesize control input to make the closed-loop system achieve desirable behavior (e.g., stability), and do so with theoretical guarantees when/if possible. Therefore, I think an exciting direction would be to use swarmDMD modeling to synthesize control that forces the swarm to achieve desirable behavior that the training swarm didn't have. For example, instead of learning a swarmDMD model using training data that follows the Vicsek model, it would be way cooler if the training data didn't show any flocking (i.e., a polarization $P(t)$ close to zero for all t during the training period), and then to synthesize control on the swarmDMD model that can be input into the real swarm to achieve flocking!

Another thing I would like to investigate is the assumption (C2) made in section 6. I don't quite understand why we aren't modeling the free dynamics of the individual agents. I guess the authors wanted the swarmDMD model to only describe the inter-agent interaction and forcing, but taking the swarm data and saying that you're only going to represent it as external forcing just doesn't seem right. It also limits this work to effectively being restricted to simulations only. In the real-world, simplifications like not representing free dynamics in your learned model is inevitably going to lead to a bad model. For example, in the context of robot swarms, each individual robot has inertia, internal friction, dynamic constraints, and nonlinearities popping up everywhere. I would like to play around with getting rid of (C2) and running swarmDMD on models where the individual agents' free dynamics can't be ignored. I think this would also address your comment about using something fancy like Koopman-based control on basic linear dynamic system problems. Incorporating these nonlinearities into the learned model and consensus task makes this problem harder.

But first, I think there are some more basic things I need to check about the swarmDMD paper. The more time I've spent reading it, looking at the code, and staring at the simulation results, the more suspicious I am of its validity. Given my familiarity with DMD-based modeling and control, I do believe the idea and mathematical algorithm check out. However, it's hard to tell if the code implementation is correct because, frankly, the code base is a hot mess. Now, I'm not the cleanest programmer myself, but it is almost impossible to understand what's being done where and the general flow of things. On the bright side, I have a greater appreciation for docstrings now. I think it is also a bit suspicious that this paper wasn't published

in a conference or journal, but rather on IEEE online access. This may actually turn out to be a positive thing because if I'm able to clean up the ideas and address possible flaws, I could have the first crack at publishing multi-agent DMD with control at a controls conference!

References

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, 2013.

[2] B. R. Eisenreich, R. Akaishi, and B. Y. Hayden, "Control without controllers: toward a distributed neuroscience of executive control," *Journal of cognitive neuroscience*, vol. 29, no. 10, pp. 1684–1698, 2017.

[3] N. E. Leonard, K. Lipsitz, A. Bizyaeva, A. Franci, and Y. Lelkes, "The nonlinear feedback dynamics of asymmetric political polarization," *Proceedings of the National Academy of Sciences*, vol. 118, no. 50, p. e2102149118, 2021.

[4] A. Prabhakar, I. Abraham, A. Taylor, M. Schlafly, K. Popovic, G. Diniz, B. Teich, B. Simidchieva, S. Clark, and T. Murphey, "Ergodic specifications for flexible swarm control: From user commands to persistent adaptation," *arXiv preprint arXiv:2006.06081*, 2020.

[5] C. L. Liu, I. L. D. Ridgley, M. L. Elwin, M. Rubenstein, R. A. Freeman, and K. M. Lynch, "Self-healing distributed swarm formation control using image moments," *arXiv preprint arXiv:2312.07523*, 2023.

[6] I. L. D. Ridgley, R. A. Freeman, and K. M. Lynch, "Self-healing first-order distributed optimization," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 3850–3856, IEEE, 2021.

[7] D. Bhaskar, A. Manhart, J. Milzman, J. T. Nardini, K. M. Storey, C. M. Topaz, and L. Ziegelmeier, "Analyzing collective motion with machine learning and topology," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 12, 2019.

[8] H. Chaté, F. Ginelli, G. Grégoire, F. Peruani, and F. Raynaud, "Modeling collective motion: variations on the vicsek model," *The European Physical Journal B*, vol. 64, pp. 451–456, 2008.

[9] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[10] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 25, pp. 1307–1346, 2015.

[11] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, 2016.

[12] M. J. Colbrook, L. J. Ayton, and M. Szőke, "Residual dynamic mode decomposition: robust and verified koopmanism," *Journal of Fluid Mechanics*, vol. 955, p. A21, 2023.

[13] P. J. Baddoo, B. Herrmann, B. J. McKeon, J. Nathan Kutz, and S. L. Brunton, "Physics-informed dynamic mode decomposition," *Proceedings of the Royal Society A*, vol. 479, no. 2271, p. 20220576, 2023.

[14] S. E. Otto and C. W. Rowley, "Koopman operators for estimation and control of dynamical systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 59–87, 2021.

[15] E. Hansen, S. L. Brunton, and Z. Song, "Swarm modeling with dynamic mode decomposition," *IEEE Access*, vol. 10, pp. 59508–59521, 2022.