UNIVERSITY OF TRIESTE

Study of multi-agent systems with reinforcement learning

by

Mihir Suneel Durve

Under the supervision of
Prof. Antonio Celani

and co-supervision of
Prof. Edoardo Milotti

A thesis submitted in fulfillment for the degree of
Doctor of Philosophy

in the
Faculty of
Physics

February 2020

UNIVERSITY OF TRIESTE

Abstract

Faculty of
Physics

Doctor of Philosophy

by Mihir Suneel Durve

Collective behavior in biological systems is one of the most fascinating phenomena observed in nature. Many conspecifics form a large group together and behave collectively in a highly synchronized fashion. Flocks of birds, schools of fish, swarms of insects, bacterial colonies are some of the examples of such systems. Since the last few years, researchers have studied collective behavior to address challenging questions like how do animals synchronize their motion, how do they interact with each other, how much information about their surroundings do they share, and if there are any general laws that govern the collective behavior in animal groups, etc. Many models have been proposed to address these questions but most of them are still open for answers.

In this thesis, we take a brief overview of models proposed from statistical physics to explain the observed collective in animals. We advocate for understanding the collective behavior of animal groups by studying the decision making process of individual animals within the group. In the first part of this thesis, we investigate the optimal decision making process of individuals by implementing reinforcement learning techniques. By encouraging congregation of the agents, we observe that the agents learn to form a highly polar ordered state i.e. they all move in the same direction as one unit. Such an ordered state is observed and quantified in a real flock of birds. The optimal strategy that these agents discover is equivalent to the well-known Vicsek model from statistical physics.

In the second part, we address the problem of collective search in a turbulent environment using olfactory cues. The agents, far away from the odor source, are tasked with locating the odor source by sensing local cues such as the local velocity of the flow, odor plume etc. By optimally combining the private information (such as local wind, presence/absence of odors, etc.) that the agent has with public information regarding the decisions to navigate made by the other agents in the system, a group of agents complete the given search task more efficiently than as single individuals.

Acknowledgements

I would like to express my sincere thanks to my supervisor Prof. Antonio Celani. He has always encouraged me to learn. I'm in debt to him for patiently guiding me for the past 3 years. He has always encouraged me to discuss, form collaborations with other researchers. He is a very kind, wise person with a passion for science. He is the smartest person I know so far and I hope to continue to learn from him in the future.

I thank my co-supervisor Prof. Milotti for all the encouragement and support.

I would like to thank Dr. Ahmed Sayeed. He has been mentoring me throughout my scientific career. He has spent numerous hours training me for scientific research and for critical thinking. I hope our scientific collaboration continues in future.

I express my sincere thanks to Dr. Arnab saha, Dr. Fernando Peruani, Dr. Massimo Cencini, Prof. Luca Biferale and Lorenzo Piro. I had a great opportunity to work in collaboration with them.

I learnt a lot from my friends Alberto, Matteo, Andrea, Anjan, and Claudio. I thank them all for the numerous discussions and many adventures together. I express my special thanks to Alberto and Matteo for the special bond of friendship that we share.

I am thankful to my long-time friends, Abdul, Abhijeet, Deepesh, Kaustubh and Varada. They were and still are my partners in many adventures, my unshakable pillars of support, my inspiration, my family. As a person, I am greatly influenced by them. My special thanks to Deepesh for all his help in preparing this thesis.

I would like to thank my kind landlord Mr. Sergio Santon, physics department secretary Ms. Rosita Glavina and Q.L.S. group secretary Ms. Erica Sarnataro for their immense help and support throughout the course of my Ph.D.

I thank ICTP and University of Trieste for graduate fellowship and kind hospitality. I thank University of Rome Tor Vergata, Italy and Laboratoire J. A. Dieudonné, Université Côte d'Azur, France for their kind hospitality and support during my visits.

Last but not least, I'm in forever debt to my mother Neeta Durve, my late father Suneel Durve and my wife Kalyani. Whatever my achievements are, they are theirs too.

- Mihir Durve

Contents

Chapter 1

Collective animal behavior - a brief overview

1.1 Introduction

One of the important features of living beings is the ability to move in space. When several individuals move in what appears to be a synchronized motion, that leads to a spectacular and fascinating display of patterns and motions. Starling murmurations [1–4], schools of swimming fish [5–7], bacterial colonies [8–10] observed under the microscope are few of the examples. The observed patterns formed by these animals have an aesthetic appeal. The natural questions that arise are, whether or not these patterns are similar in various groups? Can non-living systems show such mesmerizing patterns and have something in common with living systems? It turns out that both living and non-living systems exhibit a rich variety of patterns. Apart from the fact that many agents move together, there are many other things that are common in these systems despite the fact that these systems differ vastly in types and scales. The specific area of physics, called 'active matter', includes the study of systems consisting of many 'self-propelling' units using tools from statistical physics.

Although the large variety of systems studied in active matter physics, we focus our attention on the study of collective behavior in animals. The questions that arise can be categorized as 'why' and 'how' these animals behave collectively.

Behavioral ecologists study animal groups to address questions like why these animals behave collectively?, what are the motivations in staying together? What are the costs and benefits for individuals to be a part of a group? It is observed that the individuals in a group have better chances of avoiding predators compared to when they are alone. Often, predators are hesitant in attacking larger groups. Also, the predators are not able to fixate and pursue their prey in the background of constantly moving animals. This effect is termed as 'the confusion effect' by Milinski et al [11]. Not only animals are safer in large groups, but also there are instances where a group of animals is more efficient in completing some tasks compared with the lone individuals. One such instance is collective foraging. It is observed that group of fish find random food patches in less time than an individual fish does [12]. On the other hand, being in a group also means sharing the vital resources such as oxygen in the school of fish. An enlightening book on the topic of costs and benefits to an individual that is a part of a group is written by J. Parrish [13].

Since the last few decades, researchers have addressed questions like how can so many individuals synchronize their motion? How do they move? Are there any general principles that individuals belonging to different species obey which lead to the observed collective behavior?

To answer these questions, researchers have used a variety of methods such as using models from statistical physics, using sophisticated technology to observe individual animals to understand their behavior. We take a brief overview of the models proposed and a few experimental studies to understand how animals behave collectively.

1.2 Models for collective behavior

There are certain similarities between the physical systems studied in statistical physics and animal groups. In both cases, a system consists of (a) many particles in motion, (b) particles interacting with each other, (c) particles interacting with their environment by exchanging energy etc, (d) particles affected by noise in the system. Statistical physics taught us that since the animal group consists of interacting units, their collective behavior might display some emergent universal features. These universal features do not depend on the specific individual units but only on the nature of the interaction between them. This led to the development of agent-based models. In these models, the nature of agent-agent interaction is assumed and equations of motion for individuals are written. This nature of the agent-agent interaction varies from model to model. However, in most recent models, the nature of agent-agent interaction consists of three aspects of animal congregation; (a) cohesion of a group, (b) direction consensus amongst the agents, (c) avoidance of collisions between the agents. Agent-based models cannot be solved exactly because they consists of large number of agents. The resulting set of coupled dynamical equations becomes analytically intractable. Thus, it is convenient to solve them using computer simulation methods.

The study of collective animal behavior using computer simulation techniques dates back to 1980. Probably the first widely known simulation study of schools of fish was carried out by Aoki [14] in 1982. The model that they studied had the basic assumption that the direction and speed of an individual fish in the school are stochastic variables. Also, the direction of movement of an individual fish is related to its location within the school and common heading direction of its neighbors. It was observed that schooling of fish can occur in spite of each fish lacking the knowledge of the entire school. The individual fish interacts only with its neighbors, and without an apparent leader, school of fish can move in as a single unit. A few years later, C. Reynolds [15] carried out simulations to reproduce the patterns and trajectories followed by birds flock. In this simulation study, each bird was allowed to take its own trajectory, but at the same time, to take measures to avoid collisions and to stay close to the center of mass of the flock. In this study, many realistic considerations were taken into account such as restricted vision of the birds, presence of obstacles in the flight path etc. With this simulation study, many characteristics of real flocking were observed but it was thought then that more work is needed to quantify the flocking phenomena and check the results of this study with the data from real flocks.

Later, many biology inspired models were proposed to study motion of many interacting non-living agents. Rods on a vibrating table [16, 17], Janus particles [18], etc are examples of such non-living systems. Agents in such systems are moved by a force that can be externally or internally generated. These 'self-propelled' particles interact and like biological systems, can also give rise to complex pattern formation. The models to study self-propelled particles can broadly be classified in two categories viz. models with velocity alignment rule and models without velocity alignment rule. This classification is based on the nature of agent-agent interactions within the system.

1.2.1 Models with velocity alignment rule

In this class of models, the agents can perceive the velocity of their neighbors (subset of agents in the system). Using this local information, equations of motions dictate individual agents to align with their neighbors. One of the most prominent models in this class is constructed by Vicsek et al [19].

1.2.1.1 Vicsek model

In 1995, Vicsek et al. [19] constructed a model for flocking with simple rules of interaction. The model nowadays is widely studied and known as the *'Vicsek Model' (VM)*. The model was inspired by observations of collective behavior in far from equilibrium systems, such as flock of birds and growth process observed in bacterial colonies. Their aim was to construct a model which can show collective behavior with minimal ingredients. Their model can be summarized in one sentence as *"Move as your neighbors are moving"*.

For the implementation of the Vicsek model, simulations are carried out in a box of size $L \times L$. N agents are placed inside the box randomly and uniformly. An agent in the Vicsek model is a point particle and it could mean any entity such as a bird or a robot. Each agent is assigned with a random heading direction $\mathbf{v}$. The usual periodic boundary conditions are imposed in both directions of the simulation box. The initial system set-up is shown in Fig. 1.1A. At a given time, each agent computes the average velocity $\hat{\mathbf{v}}$ of its neighbors. Neighbors are defined as the agents within a distance R from the agent. The neighbors of a randomly chosen agent in the system are pictorially depicted in Fig 1.1B.

The velocities $\mathbf{v}$ and the positions $\mathbf{r}$ of the agents at time $t+\Delta t$ are obtained from the velocities and positions at time t using the following update rules. Firstly, positions of all the agents are simultaneously updated according to

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \mathbf{v_i}(t)\Delta t. \tag{1.1}$$

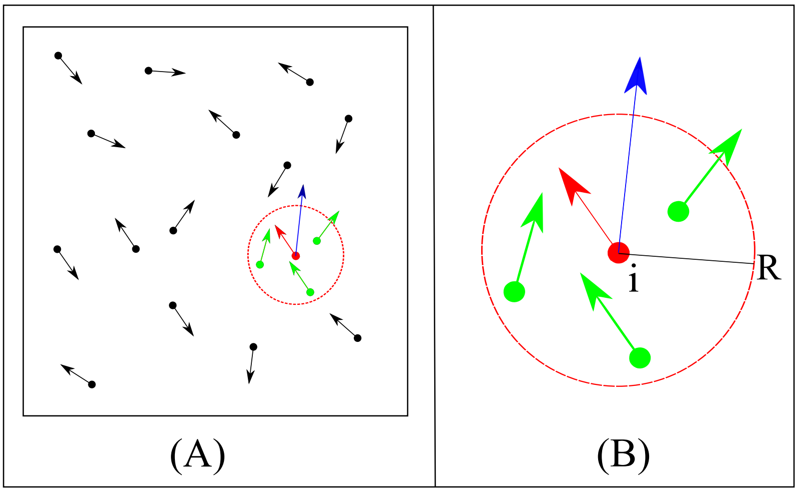


FIGURE 1.1: Ingredients of the Vicsek model. (A) N agents placed in a square box. (B) Neighborhood of interaction (a circle of radius R centered on the particle) of particle i(red). The agents shown by the green color fall within the neighborhood of the agent i and average direction of neighbors is shown by the blue arrow.

Here, $\Delta t = 1$. Later, velocities of the agents are updated according to

$$\mathbf{v_i}(t+1) = v_0 \mathcal{R}(\theta)\hat{\mathbf{v}}(t). \tag{1.2}$$

Here, v_0 is the speed of the agents and is constant for all agents, $\mathcal{R}(\theta)$ is the rotation operator. It rotates the vector it acts upon (i.e., $\hat{\mathbf{v}}(t)$) by an angle θ. The angle θ is a random variable uniformly distributed over the interval $[-\eta\pi, \eta\pi]$, where η is the level (i.e., amplitude) of the noise in the range 0 to 1. $\hat{\mathbf{v}}(t)$ is the unit velocity in the direction of the average velocity of the neighbors of the i^{th} agent (see Fig. 1.1B), and is given by

$$\hat{\mathbf{v}}(t) = \frac{\sum\limits_{j \in \mathrm{S}_i} \mathbf{v}_j(t)}{\left| \sum\limits_{j \in \mathrm{S}_i} \mathbf{v}_j(t) \right|}. \tag{1.3}$$

Here $|\ldots|$ denotes the norm of the vector.

This update scheme (i.e. updating positions first and later velocities of the agents) followed by Vicsek et al. is known as the backward update rule (BUR) in literature [20]. In recent studies, another update scheme is followed which is known as the forward update rule (FUR scheme). In this update scheme, the velocities of the agents are updated first and then the positions of agents are updated using the newly computed velocities. It was expected that the behavior of the system shall qualitatively remains same for both update rules [21]. It was showed by Huepe et al. [22] and Baglietto et al. [20] showed that the update rules led to qualitatively similar but quantitatively different results. Fig. 1.2 shows the order parameter vs noise plot for what they call SVM(Standard Vicsek Model) and OVM(Original Vicsek Model) for two system sizes. The OVM uses the update rules originally used by Vicsek et al.

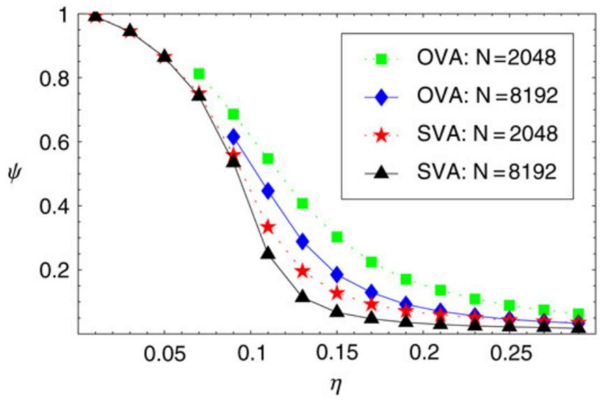


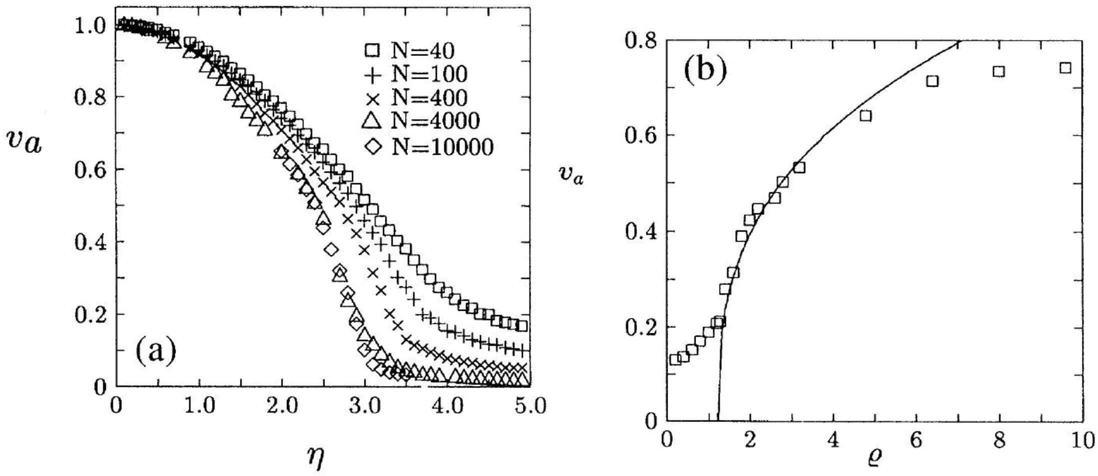
FIGURE 1.2: Order parameter vs noise plot for two different update schemes. '*N*' in the inset corresponds to Number of particles. OVA and SVA are the update schemes discussed above.
Figure source : Huepe et al. [22], with kind permission from Elsevier.

To quantify the degree of order in the collective motion of agents, Vicsek et al. defined a scalar order parameter $\Psi(t)$. It is defined as;

$$\Psi(t) = \frac{1}{N v_0} \left| \sum_{i=1}^{N} \mathbf{v}_i(t) \right|. \tag{1.4}$$

It can be easily seen that in the perfectly ordered state when all the agents are moving in the same direction, $\Psi(t) = 1$ and in the completely disordered state when the directions of motion are completely random, $\Psi(t) = 0$ (in the limit $N \rightarrow \infty$). In this context, we use the phrase 'ordered state' to mean the stationary state of the system for which $\Psi(t) > 0$.

With these minimal ingredients, Vicsek et al. observed that the agents can spontaneously form an ordered state from random initial conditions. Although the agents interact locally, a global ordered state can be formed with the given set of rules. Agents can collectively move in a common direction without any informed leader. Moreover, they studied the system by increasing the noise in the system as well as by increasing density of agents. They observed that the system undergoes a phase transition from ordered state to disordered state when the noise in the system is increased or the density of agents is decreased. Fig. 1.3a shows the plot of order parameter(Ψ) vs noise (η). In the numerical simulation of the model, as the system size (i.e. number of agents N) is increased, the finite size effects are suppressed and the behavior of the order parameter Ψ converges.



(A) (B)

FIGURE 1.3: (A)Order parameter vs Noise plot. *'N'* in the legends corresponds to Number of agents. (B)Order parameter vs Density plot.
Figure source :
Figure source: Vicsek et al. [19] https://doi.org/10.1103/PhysRevLett.75.1226, with kind permission from American Physical Society (APS), ©APS.

That means, if the strength of the noise is low, then agents form an ordered state in which they move in a common direction. However, above a certain critical value of noise, the ordered state cannot be achieved and agents will essentially continue to perform random walks. Also, below a critical density, the agents cannot achieve an ordered state. Vicsek et al. established, through numerical simulations, that in the critical region of noise and critical region of density, order parameter (Ψ) behaves as;

$$\Psi \sim [\eta_c(\rho) - \eta]^\beta \qquad \text{and} \qquad \Psi \sim [\rho(\eta) - \rho_c(\eta)]^\delta$$

They estimated the values of critical exponents β and δ to be $\beta = 0.45 \pm 0.07$ and $\delta = 0.35 \pm 0.06$. One of the important aspects of the results was that Vicsek et al.

showed phase-like transitions in the far from equilibrium system of self-propelled agents.

In a numerical implementation of the Vicsek model, one can fix time-scales and length-scales in the model by choosing $\Delta t = 1$ and $R = 1$ respectively. Then, the order parameter ψ of a system is governed by three parameters, viz. noise η, speed of the agents v_0, and density of agents ρ. It may be noted that in other variants of the Vicsek model, the noise is implemented in different ways. For example, Barberis et al. [23], in their model, used white noise defined by a Gaussian distribution with zero mean and variance σ^2. Chaté et al. [21] implemented a vectorial noise that depends on the local alignment of the agents such that, influence of the vectorial noise decreases with increasing local order. The main features of the Vicsek model, that do not depend on the details of the implementation, are:
1) *Spontaneous symmetry breaking:* In the dynamics of the agents, given by Eq. 1.1-1.2, there is no preferred direction of motion. However, due to a polar alignment term in Eq. 1.2, agents move in a common direction if noise in the system is sufficiently low. This common direction of motion is not chosen a priori. It is rather chosen by fluctuations and initial conditions. In the disordered state the direction of motion of the agents can change continuously in space while in the transition to an ordered state a continuous symmetry is spontaneously broken and agents move in a common direction.
2) *Local interactions:* The agents interact with their neighbors (other agents within some distance from the reference agent) and they move according Eq. 1.1. The neighbors of an agent change in a non-trivial way due to velocity fluctuations. Thus, the connectivity matrix of the agents is not static and changes in non-trivial way. This is where the non-equilibrium aspect manifests in to the Vicsek model. The connectivity matrix will be non-stationary only if interactions are local. In the singular limit $R \to \infty$, most of the interesting properties of the Vicsek model are lost.
3) *Conservation laws:* The only conservation law in the Vicsek model is the number of agents. In particular, it should be noted that the momentum in the system is not conserved.
It is important to note the similarities between the Vicsek model and known statistical physics models. Vicsek model can be seen as an off-lattice XY model in which agents move along the 'spin' directions. In the limit of $v_0 \to 0$, and static connectivity network of the agents, the dynamics of the Vicsek model would yield equilibrium distribution of XY model. In this case, the noise in Vicsek model is a

monotonic function of temperature T in XY model. In the limit of $R \to 0$, Vicsek model dynamics converges to the persistent random walk of many agents. See Ref. [24] for detailed discussion on the physics of the Vicsek model.

1.2.1.2 Modified Vicsek models

After the model was proposed by Vicsek et al, numerous modifications were introduced in the model. During my Ph.D., we investigated effects of non-reciprocal and delayed interactions among agents on collective behavior. This study albeit interesting in itself, is not a main topic of this thesis. Inspired by the fact that animals such as a bird or a fish usually will have a blind spot behind them [25, 26], we restricted the percept of the agent to a 'vision cone' (see Fig. 1.4A). Agents can only sense the velocities of other agents which are within its vision cone.

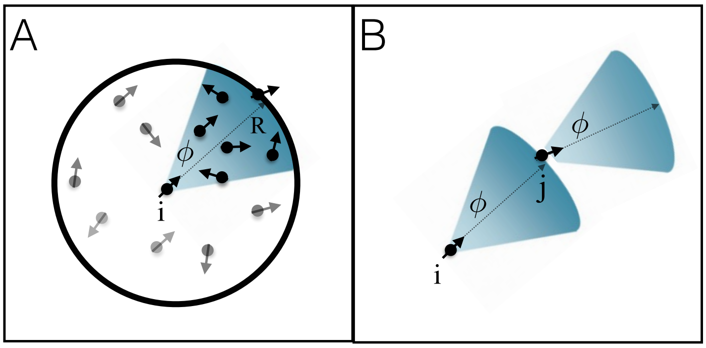


FIGURE 1.4: (A) The neighborhood S_i (blue shaded) of the i-th agent. The i-th agent is shown at the center of a circle of radius R and the neighborhood S_i is the blue sector of the circle. The black dots with arrows as heading directions indicate the agents lying within the neighborhood (including the particle at the center of the circle), and the gray dots with arrows as heading directions indicate agents outside it. The view-angle ϕ is the half opening angle of the neighborhood at the center. (B) An example of non-reciprocal configuration of agents i and j, where i interacts with j but not the other way round.
Figure source : Durve et al [27]. With kind permission of The European Physical Journal (EPJ).

The interaction between the agents with a vision cone not only make interactions anisotropic but also non-reciprocal depending on the orientation of the agents (see Fig 1.4B). We simulated this model with two update schemes (*BUR, FUR*) described above. From the previous studies [28, 29] we expected the qualitative behavior of the system to be similar under both of these update schemes. However, we observed that the order parameter ψ is qualitatively different in both the update schemes. The results of our simulations are shown in the Fig. 1.5.

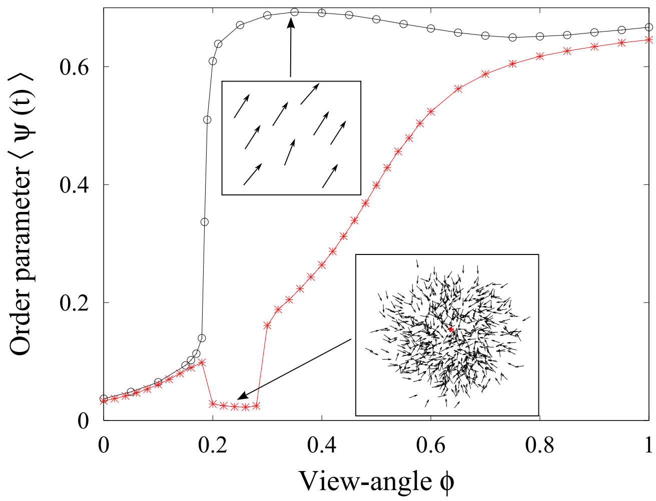


FIGURE 1.5: Order parameter ψ vs view angle ϕ for two different update schemes, the backward update scheme(red), and the forward update scheme(black). Insets shows typical configuration of the system.

Fig.[1.5] shows steady-state average (over time and multiple configurations) of Vicsek order parameter $\langle\psi\rangle$ as a function of the view-angle ϕ. Here we see the most remarkable anomalous behavior of the system. The order parameter $\langle\psi\rangle$ dips to a value close to zero around $\phi = 0.28$, and then again recovers to higher value at lower ϕ values. Within this anomalous range of ϕ (≈ 0.20 to 0.28), the value of $\langle\psi\rangle$ is slightly lower than what is expected for a completely disordered state (which yields a small non-zero value due to finite system size). In this range of ϕ, the system is indeed not in a disordered state, but in a remarkable new, ordered state where the agents, starting from random initial conditions, spontaneously confine themselves in a small, almost immobile clusters. Vicsek order parameter ψ for this state is as close to zero as it is for completely disordered state of the agents. Therefore, polar order parameter $\langle\psi\rangle$ is unable to capture the difference between this drop state and a completely disordered state. Few snapshots of this process are shown in Fig. 1.6.

We analyzed the spatial structure of the agents in this 'drop state' which is shown in Fig. 1.6d. In Fig. 1.7 we show distribution of agents with distance from the center of mass of the cluster. The distribution is obtained by counting the number of agents within the distance r and $r + dr$ from the center of mass in a single realization of the system. We observed that the local density within the cluster is not uniform. The local number density of agents at a certain radial distance from the center of mass has a maximum value. The distance at which the agents have highest local number density is about 0.25 which is half of the step-size of

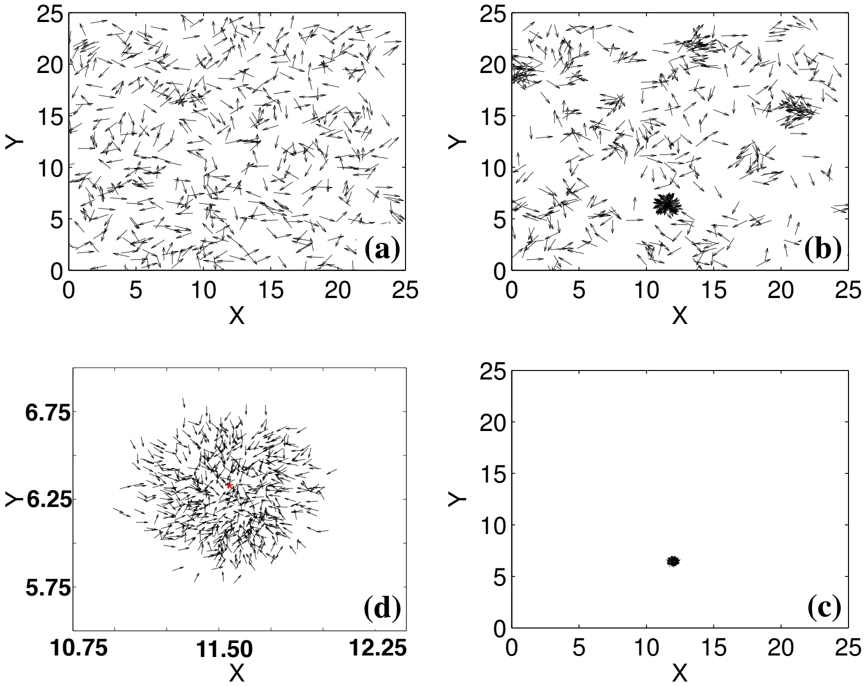


FIGURE 1.6: Snapshot of the system at various time instances (a) 0,(b) 168, (c) 4500. Arrows indicate the directions of motion of the agents. Panel (d) shows a zoomed view of the stable drop at t = 4500. The center of mass of the drop is indicated by a red asterisk. The parameters are $N = 576$, $\eta = 0.3$. Figure source : Durve et al [27]. With kind permission of The European Physical Journal (EPJ).

the agents. In the inset of Fig. 1.7 we show local density in a color plot. We conclude that the shape of the drop fluctuates in time, but the time-averaged shape is circular with non-uniform density in steady state.

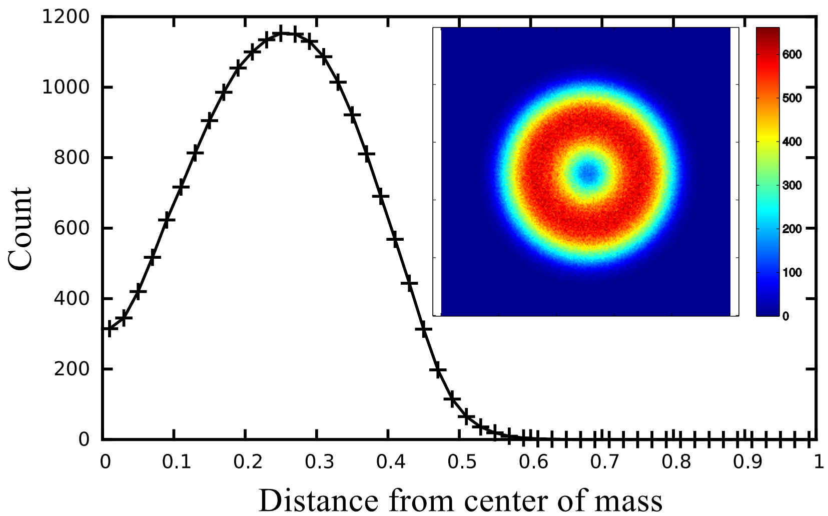


FIGURE 1.7: Distribution of agents from the center of mass of the cluster. In the inset, density of agents within the cluster. Noise $\eta = 0.3$, view-angle $\phi = 0.24$.

There are many questions that arising: Do agents form similar spatial configurations for some other choices of parameter ? What could be the mechanism by which agents are trapped in a local high density cluster without explicit attractive forces ? Are the results robust with parameters such as the density of agents, interaction radius, view-angle, etc. With simulations we address some of these questions, specifically, we studied robustness of our results and study the mechanism by which the agents are trapped in a high density cluster.

We studied robustness of our results by varying noise η and view-angle ϕ in the system. In Fig. 1.8 we show order parameter as a noise and view-angle is varied. In a certain region of the parameter space, order parameter has a value which is below the value for finite-size disordered system. In this region of the parameter space we see emergence of the highly dense local immobile clusters.

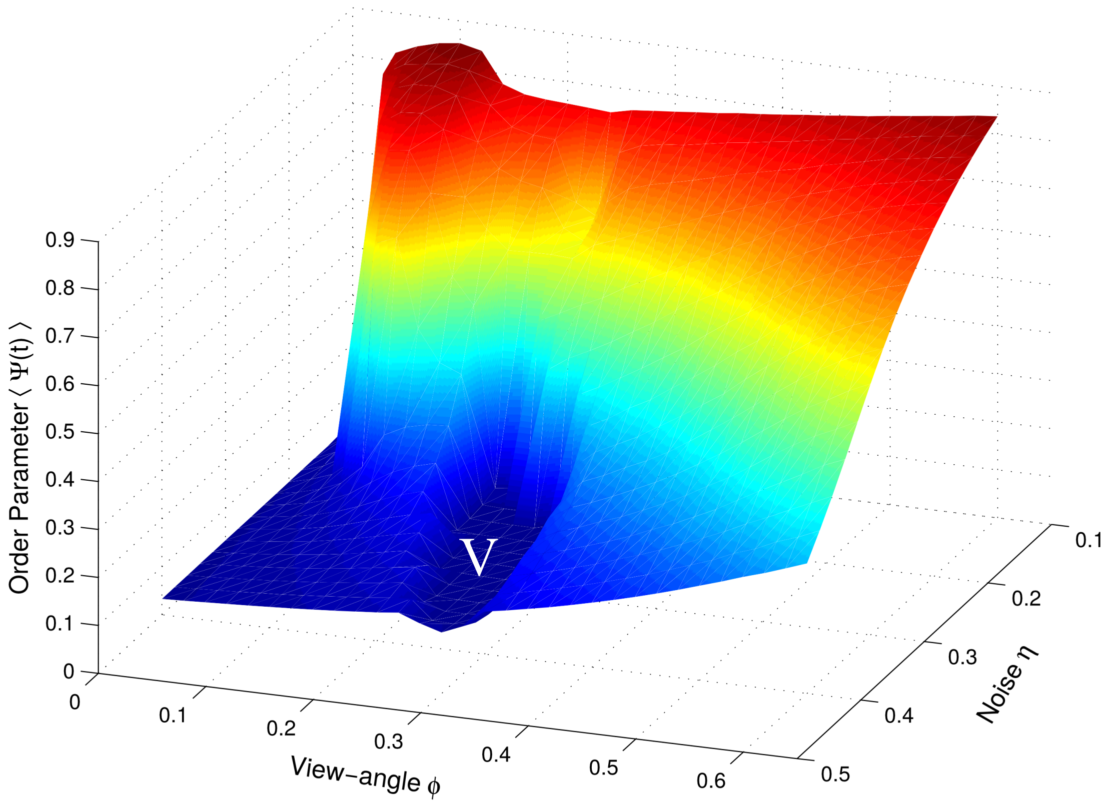


FIGURE 1.8: Order parameter ψ as noise η and view-angle ϕ is varied. The drop states form in the parameter space, in the valley region, marked by letter V.

Now, we focus our attention on the possible mechanism by which the agents form these immobile high-density clusters. In our model, there are two modifications done to the Vicsek model. First modification being, implementation of limited field of view leading to the anisotropic and non-reciprocal interactions, and second modification being, time-delayed response by the agents to their changing environment. By reducing the field of view, we effectively reduce the area of neighborhood. This might give a superficial impression that increasing the density of agents would compensate for the reduced neighborhood area. However, reducing the neighborhood area with a limited field of view and reducing the neighborhood area isotropically by changing the radius of interaction has different effects on the collective behavior of self-propelling units. In our previous work [30] we showed that the system of self-propelling units undergoes a first-order phase transition as view-angle is varied and it undergoes a second-order phase transition as the interaction radius is varied. By reducing the view-angle, the interactions among the agents become non-reciprocal and the interactions among the agents remain reciprocal while reducing the neighborhood area isotropically. It is worthwhile to note that non-reciprocal interactions do occur also in other models of collective behavior. One such example is a model with topological interactions where agents interact non-reciprocally in an inhomogeneous configuration. Non-reciprocal interactions do affect the collective behavior of active systems. In one study Cavagna et al. [31] showed that the relaxation time is significantly shorter for a system with non-reciprocal interactions than with the reciprocal interactions. We attribute the qualitative difference in the behavior of the agents with time-delayed response occurring due to a specific update scheme. With the backward update rule (BUR), agents first update their positions $\mathbf{r}$ and then update their velocities $\mathbf{v}$. Thus, with this scheme, agents inherently respond with a minimal delay of a single time step. Such a delay is absent in the forward update rule where agents first update their velocities $\mathbf{v}$ and then update their positions $\mathbf{r}$ using the recently computed velocity $\mathbf{v}$ [27]. Thus, agents behave in different way when 2 groups of the agents face each other with these 2 update schemes and with limited field of view. With forward update rule, agents would turn in opposite directions and move away from each other. However, with backward update rule, both group of agents would oscillate around the mid-point of these two groups and thus forming a locally dense cluster with approximate size.

In another similar study, Tian et al. [32] introduced anisotropic interactions among the agents mediated with vision cone of the agent. They varied the view-angle ϕ and measured the 'consensus time', that is the time taken for a system to achieve a stationary value of the order parameter ψ, in the absence and presence of noise. They made a counterintuitive observation that the consensus time can be shorter for $\phi < \pi$, i.e. restricting the angular range of the agent can speed up the establishment of the ordered collective motion.

In another variation of the Vicsek model, Gao et al. [33] considered restrictions on the turning angle θ of an agent in a short time span. In their model, they allowed an agent to turn by maximum angle θ_{max} to align with its neighbors. Therefore the agents have a maximum angular velocity ω. They observed that there exists an optimal value of maximum turning angle θ_{max} that maximizes the direction consensus of the agents. i.e. maximize the polar order parameter ψ in the presence of noise.

Combining the restricted vision of an agent with restriction on the angular velocity ω of the agent, Costanzo et al. [34] identified the region in the parameter space (Density ρ, speed v_0, radius of interaction R, angular velocity ω, vision angle α) where they observed milling-like patterns formed by the agents.

In 2002, Couzin et al. [35] proposed a model in which the nature of agent-agent interaction depends on the distance between the agents. Each agent has 3 zones of interactions as shown in Fig 1.9. The agent at the center interacts with a repulsive force with its neighbors in the zone of repulsion (ZOR). This zone represents the neighbors that are dangerously close to the agent and they must repel each other to avoid collisions. An agent orients with its neighbors in the zone of orientation (ZOO). This allows agents to move in a common direction. An agent interacts with attractive force with its neighbors in the zone of attraction (ZOA). This helps in forming a cohesive group. According to the model, an agent can 'see' other agents in these three zones except in the blind spot behind the agent given by angle α. The velocity of the agent is the weighted sum of the contribution of interactions from the zone of orientation and the zone of attraction. If, however, there are neighbors in the zone of repulsion of the agents then the priority of the agent is to avoid collisions. With simulations in 3D space, Couzin et al. observed flocks of different shapes with varying degrees of alignment. They observed milling-like patterns, swarms, and highly polar flocks in these simulations.

1.2.2 Models without velocity alignment rule

The models described in previous section assume that agents align their velocities according to the velocity of their neighbors. However in 2008, Grossman et al. [36]

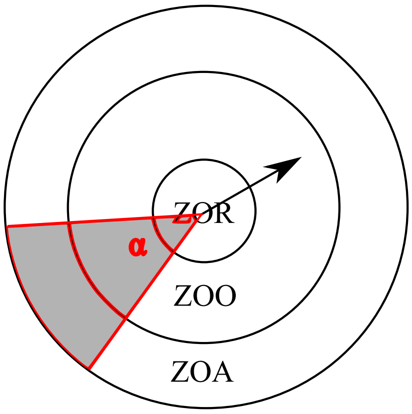


FIGURE 1.9: A 2D representation of the zones of repulsion (ZOR), orientation (ZOO) and attraction (ZOA) with a blind spot (shaded region) spanned by angle α behind the agent (black) in the model constructed by Couzin et al [35].

studied collective behavior of *SPPs* with a minimal model that does not include velocity alignment rule explicitly. In this model, the self-propelled isotropic agents are represented by round smooth inelastically colliding disks moving on a two-dimensional (2D) frictionless flat surface. According to their model, if particles come close to each other, they do not change their orientation by sensing the direction of their neighbors. Instead, they undergo an inelastic collision. The interaction among the disks are described by what is known as the spring-dashpot model [37]. With this model, Grossman et al. observed various patterns such as vortex-like pattern when the disks are placed in the circular arena and polar ordered state when the disks are placed in periodic boundary conditions. They showed that without any explicit velocity alignment rule, these soft, colliding agents can form a polar ordered state.

In 2016, Barberis et al. [23] constructed another model without an explicit velocity alignment rule. In this model, an agent interacts with its neighbors by a short-ranged, position-based, attractive force. The neighbors of the agent are the other agents within its field of view. An agent perceives the position of its neighbors and at each time step , it updates its own position $\mathbf{x}$ and orientation θ as;

$$\dot{\mathbf{x}}_i = v_0 \mathbf{V}(\theta_i); \dot{\theta}_\mathbf{i} = \frac{\gamma}{n_i} \sum_{j \in \Omega_i} sin(\alpha_{ij} - \theta_i) + \sqrt{2D_\theta} \xi_i(t). \tag{1.5}$$

Here, $\mathbf{x}_i$ is the position of the agent and θ_i is its heading direction. v_0 is the speed of the agent, γ is the strength of interaction and $\xi_i(t)$ is the delta correlated white noise with amplitude D_θ. α_{ij} is the angle of the vector $(\mathbf{x}_j - \mathbf{x}_i/|\mathbf{x}_j - \mathbf{x}_i|)$. Ω_i

represent the neighbors of the agent which are the agents in a sector of a circle with radius R and opening angle β centered on the agent.

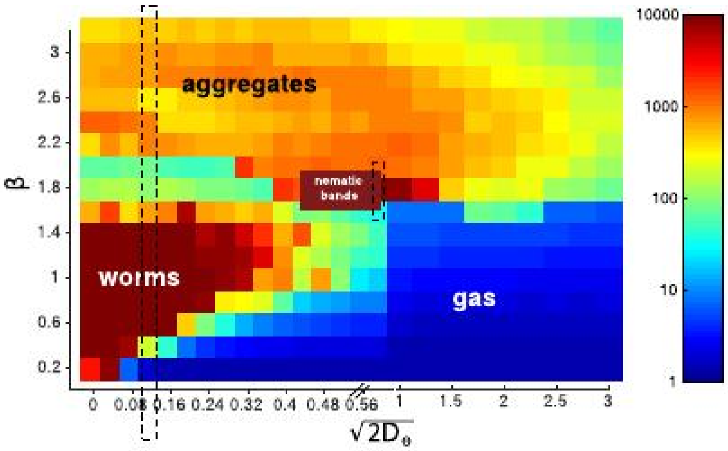


FIGURE 1.10: Phase diagram observed by Barberis et al. [23].
Figure source: Barberis et al. [23] https://doi.org/10.1103/PhysRevLett.117.248001, with kind permission from American Physical Society (APS), ©APS

With this model, they observed that depending on the parameter values, the agents form a gaseous phase, an aggregate phase, a worm phase and a nematic phase. The phase diagram is shown in Fig. 1.10

The spatial structures observed in different phases are shown in Fig. 1.11

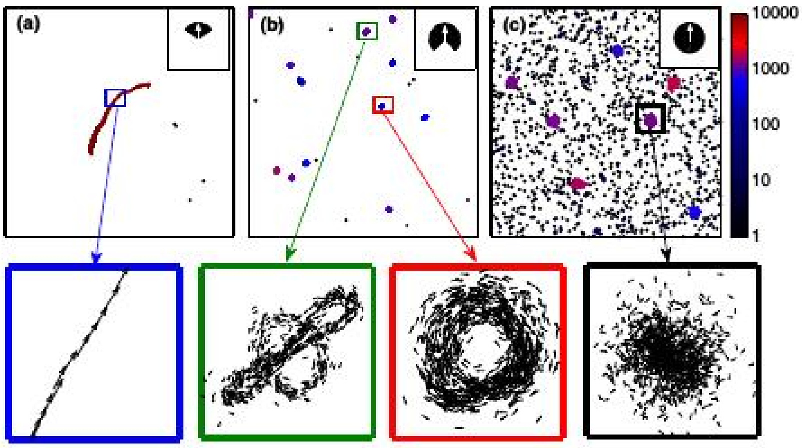


FIGURE 1.11: Spatial patterns observed for various values of vision cone angle β by Barberis et al. [23].
Figure source: Barberis et al. [23] https://doi.org/10.1103/PhysRevLett.117.248001, with kind permission from American Physical Society (APS), ©APS

Thus, the authors observed various pattens that are similar to the patterns formed by animal groups with a model that doesn't assume the velocity alignment between agents.

In another study, Cavagna et al. [38] constructed a model to account for the collective turn observed in real flocks. They observed that in real flocks, when a bird starts to turn, this information is propagated unattenuated in the whole group. Therefore, the entire flock performs a collective turn. The model constructed by Cavagna et al. is based on the conservation of internal momentum of the agent. They called this internal momentum as 'spin'. The equations of motion were written down as;

$$
\begin{aligned}
\frac{d\vec{v}_i}{dt} &= \frac{1}{\chi}\vec{s}_i \times \vec{v}_i \\
\frac{d\vec{s}_i}{dt} &= \vec{v}_i(t) \times \left[\frac{J}{v_0^2}\sum_j n_{ij}\vec{v}_j - \frac{\eta}{v_0^2}\frac{d\vec{v}_i}{dt} + \frac{\vec{\xi}_i}{v_0}\right] \\
\frac{d\vec{r}_i}{dt} &= \vec{v}_i(t).
\end{aligned}
\tag{1.6}
$$

In these equations of motion, $\vec{s}_i$ is the internal spin of the agent, the parameter χ is a generalized moment of inertia, ξ is the delta correlated noise term, η is a friction coefficient, and J is the strength of alignment with neighbors of the agent. With this inertial spin model, they showed that the model accurately accounts for the collective turns observed in real flocks.

Apart from these approaches of constructing a model and testing its relevance with collective behavior, researchers also considered the opposite approach to infer the rules of collective behavior from observations of animal groups.

1.3 Inferring the rules of interaction

In another approach to study collective behavior in animals, researchers analyzed data from experimental observations of animal groups and inferred rules that govern animal interactions. In 2008, Ballerini et al. [2] reported interesting observations of flocks of starlings. They used photographic techniques to track the motion of the starling flocks. They measured the angular orientation of nearest neighbor of a reference bird with respect to the flock's direction of motion. They repeated this measurement by taking all individuals within a flock as reference bird constructed the average angular position of nearest neighbors with respect to the flock's direction. They observed that in a flock, there is lack of nearest neighbors flying along the motion of a reference bird. Thus they concluded that the structure of individuals in a flock is strongly anisotropic. They suggested that the possible reasons for this anisotropy could probably be related to the visual range of the birds. This anisotropy in the spatial structure of a flock is crucial for its cohesive motion. They quantified decay of the anisotropy as a function $\gamma(n)$ of n-th nearest neighbors. This function $\gamma(n)$ measures to the extent to which the spatial distribution of the n-th nearest neighbor around a reference bird is anisotropic. From the observed data, they concluded that the threshold value of anisotropy γ for a flock is reached when a bird interacts with 6-7 of its nearest neighbors. Thus they suggested that the birds interact with a fixed number of neighbors, typically 6 or 7 to maintain the flock. They called it 'topological interactions'. This hypothesis is in disagreement with the then previously thought hypothesis that birds interact with all other birds which are in the neighborhood of a certain fixed size (metric interactions). However, topological interactions makes more sense due to the fact that biological agents are limited in their cognitive capacities and thus, they can pay attention to few other agents [39]. Ballerini et al. supported their observations by numerical simulations and showed that such topological interactions can have better cohesion in a flock than that of the metric interactions. They also developed photographic techniques to capture data from the flock of few hundred birds. This work is highly significant as it demonstrates the techniques to study large flocks moving in 3 dimensional space. Fig. 1.12below is snapshot of a flock consisting of 1246 starling birds and construction of the 3 dimensional image reported by Ballerini et al.

In another study with schools of fish, Herbert-Read et al. [40] tracked trajectories of mosquitofish schools placed in a square arena. They observed that a fish responded to the position of its neighbors through short-range repulsive and longer-range attractive forces. A fish responds by changing its speed and changing its direction of motion. A fish is attracted towards its neighbors that are in the attraction zone i.e. if they are farther than critical distance d from the fish. The critical distance was observed to be 6 cm. The fish was observed to be accelerating towards the position of its neighbors in front of them and decelerating in response to neighbors behind it. If a fish is at a distance less than the critical distance d

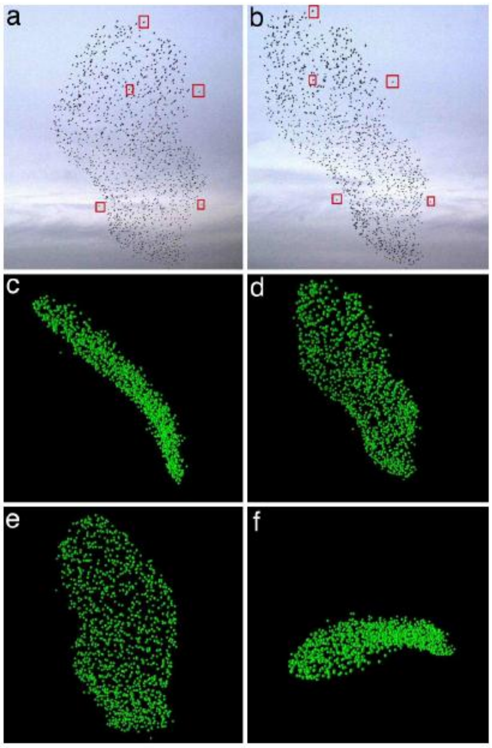


FIGURE 1.12: (a)-(b) Snapshot of a flock of 1, 246 starlings taken from two different positions. Photo from left column is matched with photo from right column to construct a 3D image. 5 such matching pairs are shown by the red squares. (c)–(f) 3 dimensional reconstruction of the same flock. Image source : Ballerini et al. [2]. With kind permission of PNAS.

from its neighbors then it was observed to be repelled by its neighbors. There was no evidence of explicit orientation alignment between the fish in the attraction zone. They observed that moving direction of a fish is maximally correlated with the direction of a fish in front of it after a small time delay, suggesting that the fish behind follows the fish in front, thereby coming into alignment with this neighbor. Thus, with this mechanism a school of fish can move in a common direction.

In another observational study, Katz et al. [41] analyzed the trajectories of golden shiners (Notemigonus crysoleucas) and they observed that a fish in a shoal responds to the change in speed of the other fish present in front of it. They also concluded that the fish interact with each other via attractive and repulsive forces. Similarly, Lukeman et al. [42] studied flocking surf scoters consisting of hundreds of individuals on the water surface. They observed that these animals interact with strong short-range repulsion, intermediate-range alignment, and longer-range attraction. They also found evidence that individuals are influenced significantly by other animals in the front. However in these studies, the authors keep the question about universality of their findings open to answer.

All these studies aim to decipher complex collective behavior shown by animals that accomplish non-trivial tasks. On the other hand, individual animals also achieve remarkable feats. One such example is locating odor sources in turbulent environment using ability to smell minute amounts of odors dispersed in the surroundings. During my Ph.D. I studied two scientific problems, one concerned with understanding collective animal behavior(Chapter 2) and second concerned with improving performance of individuals in searching for odor source in turbulent environment by taking advantage of collective behavior(Chapter 3).

1.4 Olfactory search strategies in animals

In Chapter 3, we study and present results of collective olfactory search strategies. For many animals, searching for resources such as food, mates, sites for oviposition is a recurrent task. Male moths searching for females, mosquitoes looking for a human host are some of the examples. In such cases, the desired target, such as a female moth, releases its specific odor chemicals in the environment. The search of the locating animal is guided by this wind-born odor. However, due to turbulent nature of the environment, the search task becomes highly non-trivial.

The dispersion of the odor in the turbulent environment is dominated by advecting flow. The diffusion coefficient of molecules such as ethanol, hexadecanol (similar in size with moth pheromones) is of the order of 10^{-5} m^2 s^{-1} [43]. This small rate of molecular diffusion indicates that the dispersion of odor is mainly due to advecting flow than molecular diffusion. Thus, due to turbulence, odor plume contains unevenly distributed patches of odor chemicals that persist for long distances downwind [44]. In such environments, strategies such as gradient ascent are not effective to find source of the odor. Instead, animals have developed effective search strategies using the environmental cues (wind direction, temp, humidity, etc.) and intermittent odor detection to locate the source.

A female mosquito, depending on the species of the mosquito, has a preference to feed blood from different regions of the human body. Female yellow fever mosquito (*Aedes aegypti*) uses fluctuating concentration of carbon-dioxide exhaled by human host to locate the host from longer distances [45]. However, close to the host, she selects the landing site using other cues such as other body odors, visual appearance of the host, and presumably elevated heat and humidity levels [46]. In

1998, Geier et al. [47] showed experimentally that, in the controlled environment of uniform concentration of carbon-dioxide, mosquitoes (*Aedes aegypti*) did not travel upwind but rather the upwind travel of mosquitoes was observed only in the non-uniform concentrations of the carbon-dioxide. The opposite effect was observed with the other odors that are released by human skin. The upwind flight of the mosquitoes was elicited only in the uniform concentration of these odors. This suggests that the mosquitoes use fluctuating intensity of carbon-dioxide to locate the host from the distance and use additional cues to identify the host and to land.

Male gypsy moths find mates using the pheromones released by females. Male moths are remarkably capable of locating the female from large distances, typically tens of meters [48]. To achieve this remarkable goal, male moths use their capability to detect minute amount of pheromones dispersed in the air and capability to sense wind direction. Male moths of *Cadra cautella* do respond even to single filament of its pheromone. Equipped with these sensing capabilities, the observed behavioral response of the moths is as follows. Moths move upwind if they detect pheromone signal intermittently but sufficiently frequently. It is observed that, in wind tunnels the intermittency of the signal is important to elicit the upwind flight response of the moth. Male moths did not fly upwind with continuous pheromone stimulus. Instead, the upwind flight was elicited by intermittent pheromone signal [49]. While navigating a moth can lose signal for 3 reasons; (a) large gaps between two detections due to the turbulent environment, (b) the upwind direction may not follow the odor plume, (c) moth's own maneuvers take it outside the odor plume. In the absence of the signal the moth showed 'casting' behavior. It performed flights in the transversal direction to the wind direction with increasing lengths forming a zig-zag pattern, until it regained contact with the odor plume [50–52]. In general, these counter-turns occur in quick successions, typically 3.5 to 4 turns per second. The frequencies of these turns were observed to be characteristics of the moth species [53, 54]. Thus, typical strategy adapted by male moths can be summarized as following. Male moths sustain upwind flight as long as they receive intermittent odor signal sufficiently frequently and in absence of it, they search for the odor signal by traveling transversally. This behavior has inspired us to construct collective olfactory search algorithm described in Chapter 3.

1.5 Introduction to reinforcement learning

As we have seen that our understanding of animal behavior is shaped by parallel development of mathematical models and experimental studies. In some studies, specific models were developed to account for the specific behavior observed in biological systems [10, 24, 55]. Other than these approaches, in Chapter 2, we present a novel approach to study collective behavior in animals. We study decision making process of individuals to achieve a certain goal. Our aim is to understand the general laws that these animals might be obeying to exhibit collective behavior by understanding their decision making process. To understand the decision making process of the individuals in an animal group, we implemented reinforcement learning techniques. Reinforcement learning is one subset of broad field of machine learning. The general scheme of reinforcement learning is presented in the following section.

Biological agents learn to behave in a certain way that is shaped by prolonged interactions with the surroundings. Reinforcement learning [56] is a broad scheme that models this phenomenon. Reinforcement learning is based on the framework of Markov decision processes. The goal of the agent is to maximize the total gain by taking sequence of actions in the environment. The general scheme of reinforcement learning is shown in Fig 1.13.

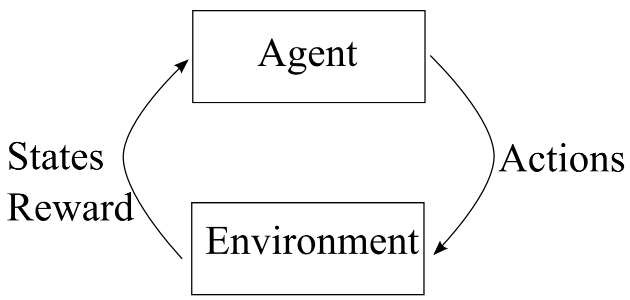


FIGURE 1.13: Broad scheme of reinforcement learning.

The key ingredients of the reinforcement learning are;

Agent : agent that can sense environmental cues and chooses to take action.
States $\mathcal{S}$: a set of possible states $\mathcal{S}$ that represent the dynamic environment. The agent perceives the state of environment $s \in \mathcal{S}$ at each time step.
Actions $\mathcal{A}$: a set of possible actions $\mathcal{A}$ that the agent can select from at each time step. After executing the selected action, the system is transformed to the next

state and the agent receives a reward.

In this scheme, an agent senses the state of the environments_t and performs an action a_t. As a consequence of the action just performed, the environment issues a reward r_{t+1} and a new state s_{t+1} of the environment to the agent. Thus by repeating this process, the experience of the agent is given by the sequence of;

$$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2} \dots$$

In this process, the new state of the environment s' is determined by the transition probability $P^a_{ss'}$. The transition probability gives the probability that the system goes to the new state s' when action a is performed in the state s. If the environment has the Markov property, then the effect of taking an action a in a state s only depends on the current state-action pair and not on the prior history i.e.

$$\begin{aligned} P^a_{ss'} &= P(s_{t+1} = s', r_{t+1} = r | a_t, s_t), \\ P^a_{ss'} &= P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t \dots r_1, s_0, a_0). \end{aligned} \tag{1.7}$$

This property dictates that the current state s_t contains sufficient information for the optimal future decisions. The goal of the agent is to maximize the total discounted reward R_t given as;

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{1.8}$$

The discount factor $0 \leq \gamma < 1$ keeps the sum finite and it is a parameter that sets how much future expected rewards are to be taken into account for optimal decision making. The goal of the learning algorithm is to find the optimal policy $\pi^*(a|s)$ that maximizes the reward R_t. A policy maps the states with actions. i.e. $\pi(a|s)$ is a probability of selecting action a in the state s. To maximize the reward R_t, it is desired to know the goodness of action a in state s. Thus, we define the action-value function of an action a under a policy π as the expected return by selecting an action a_t in the state s_t and following policy π thereafter. It is written as

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a], \tag{1.9}$$

which can be written as;

$$Q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')]. \tag{1.10}$$

It has been proven that the optimal policy consists in choosing the action a with the greatest $Q^*(s,a)$ i.e. $a = \underset{a'}{\operatorname{argmax}}\ Q(s,a')$ and the optimal values of states and actions satisfy;

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \max_{a'}\ Q(s,a')]. \tag{1.11}$$

Eq. 1.11 is known as the Bellman optimality equation. In many cases, the model of the environment is not known, and in such cases the optimal Q-function can be obtained with replacing model of the environment with experience gained through prolonged interactions with the environment. To obtain optimal quality function Q^*, we implemented algorithm known as Q-learning [56, 57]. In this algorithm current estimate of the quality function is updated using what is know as the temporal difference (T.D.) error. The T.D-error is a difference between expected reward and reward actually received by the agent.

However, implementation of reinforcement learning techniques to multi-agent systems have some additional challenges [58, 59]. There are two ways to implement reinforcement learning techniques to multi-agent systems and each way has some challenges of its own. The first way is called 'team learning'. In team learning, there is a single learner involved: but this learner is discovering a set of behaviors for a team of agents, rather than a single agent. This lacks the aspect of multiple learners, but still poses challenges because as agents interact with one another, the joint behavior can be unexpected. A major problem with team learning is the large state space for the learning process. For example, if agent 1 can be in any of 10 states and agent 2 can be in any of another 10 states, then the team of agents can be in 10×10 states. Although the reinforcement learning techniques, in principle, are still applicable to the system, they become overwhelming from the computation point of view.

Another way to implement reinforcement learning techniques to multi agent systems is called concurrent learning. In concurrent learning, each agent is learning individually. Concurrent learning may be preferable in those domains for which some decomposition of learning process is possible and helpful. That is, if the individual agent's learning can be independent of others to some extent then the concurrent learning is preferential. The problem with concurrent learning is that as the agents learn, they modify their behaviors, which in turn can ruin other agent's learned behaviors and not yield the desired results.

Regardless of method chosen, one more challenge is to assign credit to every agent. credit can be equally divided among agents such as a goal in a football match. This scheme is called global reward scheme. However, this scheme does not provide proper feedback to the agents on goodness of their actions and hamper the learning process. On the other hand one may assign local reward scheme that assign credit to individual agents but this scheme may develop greedy behavior among agents which may not achieve the ultimate goal as a team.

We implemented the framework of multi-agent reinforcement learning to study the decision making process of the agents in order to stay together as a flock. The detailed description of our study is presented in the next chapter.

Chapter 2

Learning to flock with reinforcement learning

2.1 Introduction

The spectacular collective behavior observed in insect swarms, birds flocks, and ungulate herds have long fascinated and inspired researchers [2, 60–64]. There are many long-standing and challenging questions about collective animal behavior: How do so many animals achieve such a remarkably synchronized motion? What do they perceive from their environment and how do they use and share this information within the group in order to coordinate their motion? Are there any general rules of motion that individuals obey while exhibiting collective behavior? Since the last few decades, these questions have been addressed with systematic field observations coupled with mathematical models of animal behavior. Data from experimental observations have been analyzed in order to infer the rules that individuals follow in a group [40–42, 65, 66] and numerous models have been proposed to explain the observed flocking behavior [14, 19, 23, 38, 67–69]. Basic models of flocking are essentially based on three rules: i) short-range avoidance, ii) alignment, and iii) long-range attraction. In the following we will will ignore for simplicity the separating force at short distances which may arise from collision avoidance or the indirect cost of sharing of vital resources [13] and put at the center of the stage the last of the previous three rules, that is the drive towards cohesion of the group. In real flocks or schools this tendency guarantees an increased safety from predators [11] as well as the benefits of collective foraging [12]. As for alignment, which is a key ingredient of many models of flocking, the main result of this work is that it actually follows from the requirement of cohesion, rather than being an independent rule to enforced. The great majority of flocking studies [2, 14, 19, 38, 40–42, 60–69], except for few exceptions (see [23] references therein), are based on a velocity alignment mechanism that ensures that neighboring individuals move in the same direction.

However, the origin of such a velocity alignment, from a cognitive point of view, is not known, and neither its biological function.

The natural mathematical language that we will use here to discuss collective motion is the framework of Multi Agent Reinforcement Learning (MARL) [58, 59]. In this scheme, the agents can perform actions in response to external cues that they can sense from the environment as well as from other agents. The goal of each agent is to achieve a given objective. In the case at hand, the agents are individuals who can observe the behavior of their close neighbors and react by steering according to some rule. Since it has been hypothesized that there exist many benefits associated to group-living, such as predator avoidance [11] and collective foraging [12], we assume that the objective of the agents is to increase or maintain the cohesion of the group. The essence of Reinforcement Learning (RL) is that, by repeated trial and error, the agents can learn how to behave in an approximately optimal way so as to achieve their goals [56]. Here, we show that velocity alignment emerges spontaneously in a RL process from the minimization of the rate of neighbor loss, and represents a optimal strategy to keep group cohesion.

2.2 Techniques

In the following we will consider individual agents that move at constant speed in a two-dimensional box with periodic boundary conditions. The density of agents is kept fixed to $\rho = 2$ agents/(unit length)^2. Updates are performed at discrete time steps as follows. For the i-th agent, the position update is:

$$\mathbf{r}_i^{t+1} = \mathbf{r}_i^t + v_0 \mathbf{v}_i^t \Delta t \ , \tag{2.1}$$

where $\mathbf{r}_i^t$ and $\mathbf{v}_i^t$, with $||\mathbf{v}_i^t|| = 1$, are the position and moving direction, respectively, of the agent at time t; the term v_0 corresponds to the speed, which we fix to

$v_0 = 0.5$, and $\Delta t = 1$. At each time step, each agents makes a decision on whether keeping the current heading direction or performing a turn. The decision-making process is based on the sensorial input of the agent, which corresponds to the angular difference between the (normalized) average velocity defined by $\mathbf{P}_i = (\sum_{|\mathbf{r}_j - \mathbf{r}_i|<R} \mathbf{v}_j^t)/n_i$ (with n_i the number of neighbors of agent i within its perception range R) and the moving direction of the agent $\mathbf{v}_i^t$. Below, we take $R = 1$. We can express the state as

$$s_i^t = \arg(\mathbf{P}_i, \mathbf{v}_i^t)\,, \tag{2.2}$$

where the function $\arg(\mathbf{P}_i, \mathbf{v}_i^t)$ is defined as $\arccos(\mathbf{P}_i \cdot \mathbf{v}_i^t/||\mathbf{P}_i||)$ for $\mathbf{P}_i \cdot (\mathbf{v}_i^t)_\perp > 0$ with $(\mathbf{v}_i^t)_\perp$ obtained by rotating $\pi/2$ counter clockwise the unit vector $\mathbf{v}_i^t$, and minus this quantity otherwise. This means that $s_i^t \in [-\pi, \pi)$. For computational simplicity, we discretize s_i^t by dividing 2π into K_s equally spaced elements (see appendix A). In the RL language, the relative angle s_i^t is the contextual information that defines the current state of the i-th agent. Knowing s_i^t, the agent updates $\mathbf{v}_i^t$ by turning this vector an angle a_i^t

$$\mathbf{v}_{t+1} = \mathcal{R}(a_i^t)\mathbf{v}_t\ , \tag{2.3}$$

where $\mathcal{R}(a_i^t)$ is a rotation matrix. Note that there are K_a possible turning angles, equally spaced in $[-\theta_{max}, \theta_{max}]$ (see appendix A). In the RL jargon, choosing the turning angle a_i^t represents an "action" performed by the agent. The association of a given state s_t^i with an action a_t^i is called a policy.

Policy evaluation takes place at each time step as the agent receives a (negative) reinforcement signal in the form of a cost c_i^{t+1} for losing neighbors within its perception range R

$$c_i^{t+1} = \begin{cases} 1, & \text{if } n_i^{t+1} < n_i^t \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

where n_i^t is the current number of neighbors. The goal of the learning agent is to find a policy that minimizes the cost. To achieve this goal the agent makes use of a simple learning rule [56, 57]. The i-th learning agent keeps in memory a table of values $Q_i(s, a)$ for each state-action pair (here a matrix $K_a \times K_s$) which is updated at each step of the dynamics – only for the entry that corresponds to the state just visited and the action just taken – according to

$$Q_i(s_i^t, a_i^t) \leftarrow Q_i(s_i^t, a_i^t) + \alpha[c_i^{t+1} - Q_i(s_i^t, a_i^t)]\,. \tag{2.5}$$

This update rule effectively constructs an estimator for the expected cost that will be incurred by starting in a given state and performing a given action. The policy at each time-step is based upon the current Q_i according to so-called $\epsilon-$greedy exploration scheme:

$$a_i^t = \begin{cases} \underset{a'}{\text{argmin}}\; Q_i(s_i^t, a') & \text{with prob.} \quad 1-\epsilon \\ \text{an action at random} & \text{with prob.} \;\; \epsilon \end{cases}\,. \tag{2.6}$$

In the simulations we have used $\alpha = 0.005$ and various different schedules for the exploration probability [56]. We wrote code to carry out simulations in Fortran programming language and the code is given in appendix G. [1]. The results of simulations are presented in next sections.

2.3 Results: Single agent

We start by considering the case when there is a single learning agent in a crowd of N teachers (see Figure 2.1) who have a hard-wired policy:

$$a_i^t(s_i^t) = \begin{cases} s_i^t & \text{if } |s_i^t| \leq \theta_{max} \\ \theta_{max} & \text{if } s_i^t > \theta_{max} \\ -\theta_{max} & \text{if } s_i^t < -\theta_{max} \end{cases}\,. \tag{2.7}$$

This decision rule is nothing else but a version of the Vicsek model of flocking with a discrete number of possible moving direction and limited angular speed. Thus, teachers display robust collective motion. For more details on the behavior of teacher agents see appendix B. The learning agent, on the contrary, does not have a fixed policy, but one that evolves over time as it acquires experience, i.e. by visiting a large number states, and evaluating the outcome of executing its actions. In this case we find that for suitably chosen learning rates α and exploration probability ϵ the algorithm approaches an approximately optimal solution to the decision-making problem after some period of training.

[1] For any problems to use the code as it given, I request to you to write to me (mihirdurve@gmail.com) for further support and source code files.

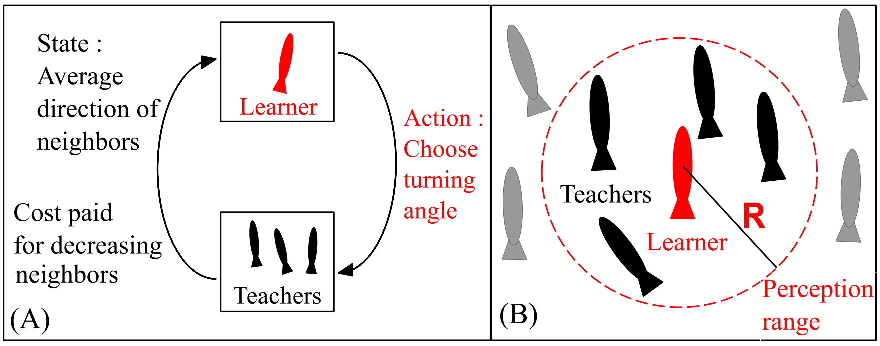


FIGURE 2.1: Learning to flock within a group of teachers. (A) Scheme of reinforcement learning. (B) Neighbors (black) within the perception range of the learner (red).

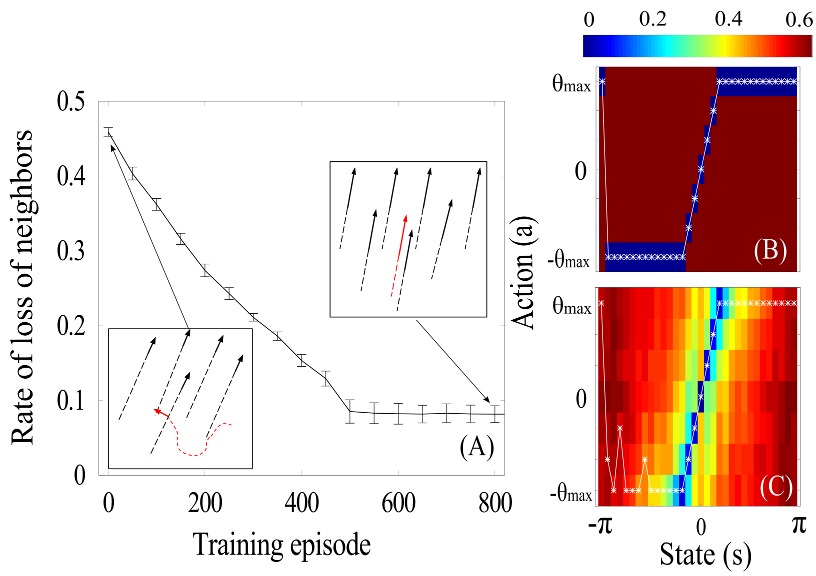


FIGURE 2.2: Single learner results. (A) Performance of the learner as training progresses. The error bars indicate standard deviation in the values in 20 training sessions. In the inset, some short trajectories of the learning agent (red) and teachers (black) at the stages indicated by the arrows. The number of teachers is $N = 200$. The maximal turning angle is $\theta_{max} = 3\pi/16$. (B) The Q-matrix, i.e the average cost incurred for a given state-action pair by the teachers. White stars shows the action a taken by the teacher when in state s. (C) Q-matrix of the learner at the end the training session. White stars denote the best estimated action of the learner for each state.

In simulations, we break the training session into a number of training episodes of equal prescribed duration of 10^4 time steps. In each episode the teachers start with random initial positions and velocities. After a transient, they form an ordered flock and at this time we introduce the learner and implement the learning algorithm. At the very beginning, the learner starts with a Q-matrix with all zero entries, which in a case of optimistic initialization (the naive learner expects to incur no costs), a choice that is known to favor exploration [56]. From one episode to the following, the learner keeps in memory the Q-matrix that it has learned so far. During the training session, we measure the success of the learning process with the average cost that a learner pays per time step, that is the rate at which it is losing contact with the teachers (see Fig. 2.2A). As the training progresses, the rate at which neighbors are lost starts from an initial value of 0.5, meaning that on average the learner loses contact with some neighbor every other step, to decrease and eventually saturate down to a value around 0.1 meaning that the contact is kept for 90% of the time. In the insets of Fig 2.2A we show samples of short trajectories of the learner and some teachers at the early and later phase of the training process. We observe that in the early phase of the training, the learner essentially moves at random (see movie1.mp4) and eventually it learns to stay within the flock (see movie2.mp4). In Fig 2.2C we show that the policy discovered by the learner is identical with the pre-defined policy of the teachers, see Eq. (2.7) and Fig. 2.2B. It is important to remark that the one and only goal of the learner is to keep contact with its neighboring teachers, not to imitate their behavior. It simply turns out that the best strategy for the learner is in fact the underlying rule that was assigned to teachers. See appendix C for additional results.

2.4 Results: Multi-agent

Now, let us move our focus to the situation where there are no teachers, but only N independently learning agents (see Figure 2.3). A distinctive difficulty of applying reinforcement learning to the multi-agent setting is that all individuals have to concurrently discover the best way to behave and cannot rely upon knowledge previously acquired by their peers. However, we find that N learning agents are able to overcome this hurdle and are actually capable of learning to flock even in the case when all of them start as absolute beginners (all Q-matrices initialized to zero).

To characterize the performance of the learners, we measure the average rate of loss of neighbors. In Fig 2.4A we show the average cost for various groups sizes and state-action space discretizations $\{K_s, K_a\}$. The cost reaches a small and steady value after few hundreds of episodes. As the group size grows, the performance remains essentially the same. Conversely, refining the discretization allows to further reduce the costs: for 128 relative alignment angles and 28 turning angles the agents do not lose neighbors for about 97% of the time.

The resulting Q-matrix at the end of the training, averaged over all learners, is shown in Fig. 2.4B. The colors represent the numerical values in the Q-matrix and the discovered policy is shown with white points. We observe that the discovered policy is the same that the one learned by the single agent with N teachers.

It is worth stressing that all agents independently learn the same strategy. We have collected the values of the Q-matrix for a given state ($s = 0$) and different actions, for all agents, at the end of training. The histogram for the frequency of Q values is shown in Fig. 2.5 where one can observe that there is a clear gap that separates the estimated costs for the optimal turning angle, which lie around 0.1, from the suboptimal actions that have significantly larger costs.

A customary measure of the degree of alignment is the polar order parameter:

$$\psi(t) = \frac{1}{N} \left\| \sum_{i=1}^{N} \mathbf{v}_i^t \right\|. \tag{2.8}$$

If all the agents are oriented randomly then, as $N \to \infty$, $\psi \to 0$ whereas if all the agents are oriented in the same direction then $\psi = 1$. In Fig 2.6 we show the evolution of order parameter versus the average cost as the multi-agent learning is advancing. We observe that in the early phases of training the rate of loss of neighbors is comparatively high and the direction consensus among the agents is low, in agreement with the notion that the agents are behaving randomly (see movie3.mp4). As the learning progresses, the agents discover how to keep cohesion, and in doing so they achieve a highly ordered state (see movie4.mp4).

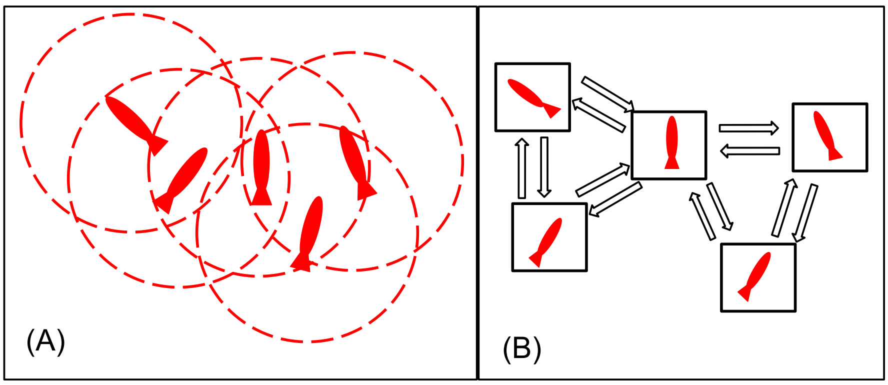


FIGURE 2.3: Multi-agent concurrent learning. (A) Multiple agents with their perception range. (B) Agents interact with short-range, reciprocal interactions.

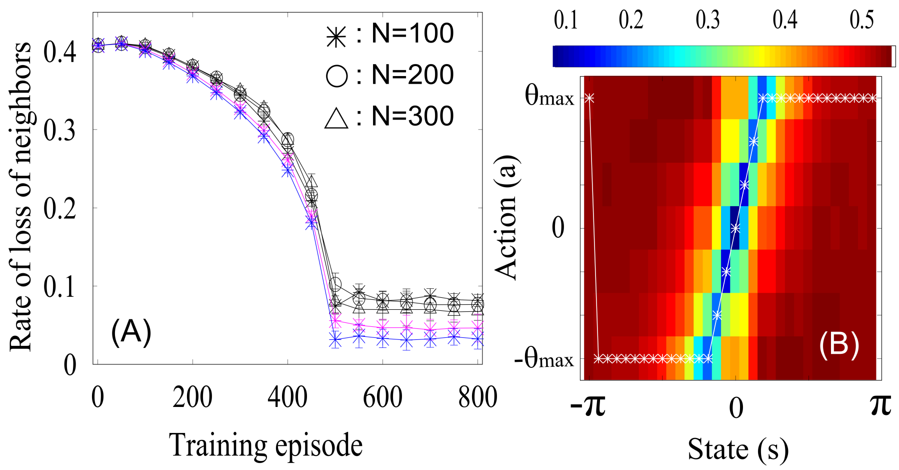


FIGURE 2.4: Results for the multi-agent concurrent learning. (A) Average performance of learners in groups of different sizes. Black, magenta and blue colors corresponds to state-action spaces of size $\{Ks, Ka\}$=$\{32, 7\}$, $\{64, 14\}$, $\{128, 28\}$ respectively. Error bars indicate standard deviation in the average values for each agent. (B) Average Q-matrix at the end of the training for $N = 200$ agents with combination of $\{Ks, Ka\}$=$\{32, 7\}$. White points indicate actions with estimated minimum cost for given state. The colors represent values in the Q-matrix.

We checked the robustness of our results by varing parameters such as density of agents, agent-agent interaction radius, number of allowed actions, etc. We observed that our results are robust with change in parameter values. See appendix D for more details. In our study we assumed that the learners can make error-free measurements. However, biological agents have limited capabilities to perceive their environment. A natural question that arises here is if agents would learn to flock with noisy observations and limited field of view, etc.? We addressed some of these questions and the results are presented in appendix E.

2.5 Conclusions

We conclude that the obtained results proves that the velocity alignment mechanism of the Vicsek model (see Eq. (2.7)) – based on energy minimization of spin-spin interaction of the XY model – can spontaneously emerge, counterintuitively, from the minimization of neighbor-loss rate, and furthermore represents an optimal strategy to keep group cohesion when the perception is limited to velocity of neighboring agents. In summary, if the agents want to stay together, they must learn that they have to steer together.

In more general terms, we have shown that Multi-Agent Reinforcement Learning can provide an effective way to deal with questions about the emergence of collective behaviors and the driving forces behind them. Our present contribution is just an initial step in this direction and we feel that prospective applications of this approach remain largely unexplored.

For instance, in the present work we have decided at the outset the structure of the perceptual space of the agents, namely the choice of the radius of perception as the relevant parameter and of the relative angle as the relevant state variable. In doing so, we bypassed fundamental questions like: Is the metric distance the most appropriate choice for ranking neighbors ? How should the information given by other individuals be discounted depending on their ranking ? A more ambitious approach would tackle these issues directly through MARL and try to

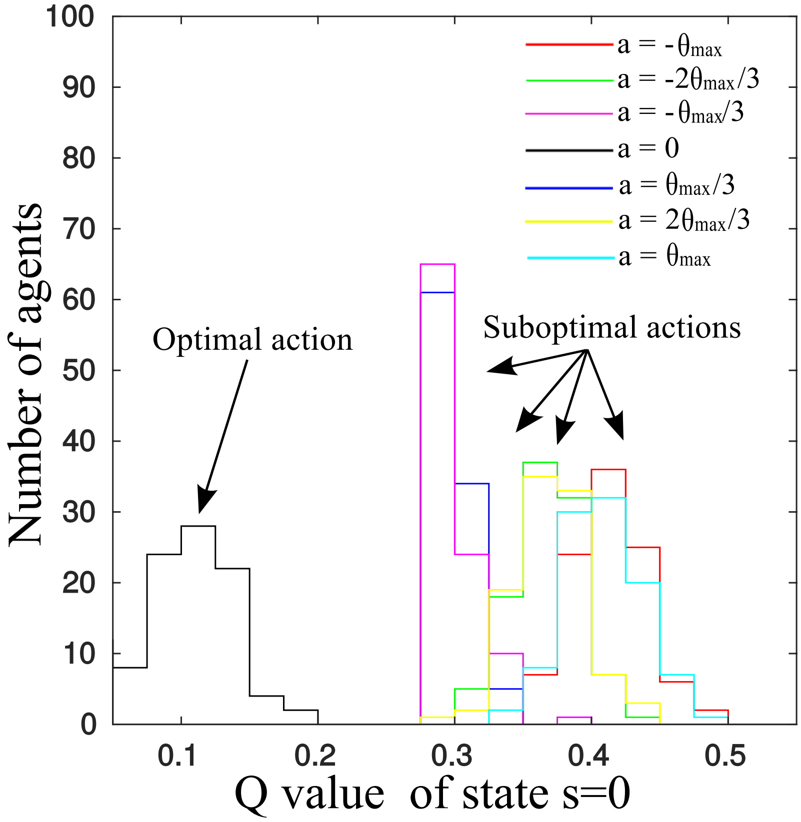


FIGURE 2.5: All agents independently learn the same optimal strategy. The histogram shows the frequency of a give numerical value of $Q(0, a)$ across all learners, at the end of training. The best action $a = 0$ always performs better than any other action. The same holds for other states (not shown). Data obtained with $N = 100$ agents, $K_s = 32$, $K_a = 7$.

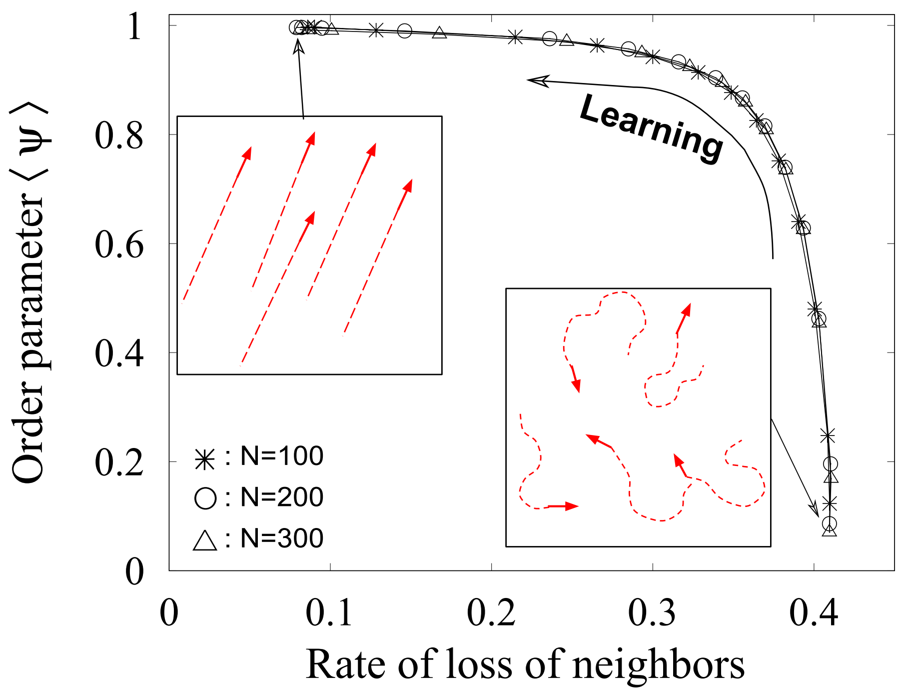


FIGURE 2.6: Average polar order parameter $\langle\psi\rangle$ versus the rate of loss of neighbors. In the insets we show a few short trajectories of naive and trained agents.

learn from experience what are better choices of the state variable that allow to achieve optimal cohesion.

As another example, here we have tasked our agents with the goal of keeping contact with neighbors, which in itself is understood to be a secondary goal motivated by the primary need of avoiding predators (safety by the numbers) or of increasing the efficiency of foraging. Can one recapitulate the congregation behavior by tasking agents with the primary goal itself ? More explicitly, would agents learn to align themselves by rewarding behaviors that reduce their risk of being predated or increase their chance of getting some food ?

Also, in this work we have considered a group of identical agents. When agents differ for their perceptual abilities or their dexterity in taking the appropriate actions, then competitive behaviors may arise within the group and the problem acquires a new challenging dimension. How much heterogeneity and competition can be tolerated before it starts impacting the benefit of staying in a group ?

These and many other questions lend themselves to be attacked by the techniques of MARL and we believe that the approach that we have delineated here will show its full potential in the near future.

Chapter 3

Multi-agent olfactory search in turbulent environment

3.1 Introduction

Animals are often on the move to search for something: a food source, a potential mate or a desirable site for laying their eggs. In many instances their navigation is informed by airborne chemical cues. One of the best known, and most impressive, olfactory search behavior is displayed by male moths [45, 48, 51, 70]. Males are attracted by the scent of pheromones emitted in minute amounts by calling females that might be at hundreds of meters away. The difficulty of olfactory search can be appreciated by realizing that, due to air turbulence, the odor plume downwind of the source breaks down into small, sparse patches interspersed by clean air or other extraneous environmental odors [71, 72]. The absence of a well-defined gradient in odor concentration at any given location and time greatly limits the efficiency of conventional search strategies like gradient climbing. Experimental studies have in fact shown that moths display a different search strategy composed of two phases: surging, i.e. sustained upwind flight, and casting, i.e. extended alternating crosswind motion. These phases occur depending on whether the pheromone signal is detected or not. This strategy and others have inspired the design of robotic systems for the identification of sources of gas leaks or other harmful volatile compounds [73–77]. Albeit the effectiveness of individual search is already remarkable in itself, the performance can be further boosted by cooperation among individuals, even in absence of a centralized control [12, 78–82].

In recent years algorithms based on have been developed for individual search processes [83]. These search processes involve searching for a target that emits a signal at low rates in random directions. the challenge for the lone searcher (agent) is to utilize the intermittent odor signal to locate the target as quickly as possible. The infotaxis algorithm prescribes dynamical rules for the agent to maximize the information it would get about the location of the target. At each time t an agent constructs a probability map by computing probabilities $P(\mathbf{r})$ of finding the target at possible locations $\mathbf{r}$. At each time an agent might 1) find the target, 2) detect the signal emitted by the target, 3) detect nothing. An agent updates its probability map based on these events. The agent's decision to move is based on the expected information it would gain from each possible move. The expected gain in information is given by the expected change in entropy of each of the possible events in that location (finding the source, detecting signal, or detecting nothing), weighted by the probability of occurrence of each of these events. This general idea of infotaxis is extended to multi-agent search strategies to enhance performance [84–87]. i.e. to locate the target in minimum time. The core challenge in the multi-agent infotaxis based search is how individual agents utilize the information available with other agents (via social interactions) to make its own infotaxis decisions. There are several models that define different methods for the agents to integrate its own information and information available with other agents. For example in one possible way, observations from all the agents are integrated to construct a common probability map called joint probability map [88, 89]. New experiences gained by individual agents contribute to updating the map. Individual agents base their decisions to move on this joint probability map. In another possible way, an agent utilizes the difference in its own probability map and probability map of its neighbors to make infotaxis decisions [84, 90]. Regardless of the method to integrate private and public information in the infotaxis based algorithms, the requirement of exchanging observations and probability maps add to the overheads to computation and communication which is undesirable for real-time applications [86].

In this work, we tackle the problem of collective olfactory search in a turbulent environment. When the search takes place in a group, there are two classes of informative cues available to the agents. First, there is private information, such as the detection of external signals – odor, wind velocity, etc – by an individual. This perception takes place at short distances and is not shared with other members of the group. Second, there is public information, in the form of the decisions made by other individuals. These are accessible to (a subset of) the other peers, usually relayed by visual cues, and therefore with a longer transmission range. Since the action taken by another individual may be also informed by its own private perception of external inputs, public cues indirectly convey information about the odor distribution and the wind direction at a distance. However, the spatial and temporal filtering that is induced by the sharing of public cues may in principle destroy the relevant, hidden information about the external guiding signals.

These considerations naturally lead to the question if the public information is exploitable at all for the collective search process. And if it is, how should the agents combine the information from private and public cues to improve the search performances ? Below, we will address these questions by making use of a combination of models for individual olfactory search and for flocking behavior, in a turbulent flow.

3.2 A model for collective olfactory search

The setup for our model is illustrated in Fig 3.1A. Initially, N agents are randomly and uniformly placed within a circle of radius R_b at a distance L_x from the source S. The odor source S emits odor particles at a fixed rate of J particles per unit time. The odor particles are transported in the surrounding environment by a turbulent flow $\boldsymbol{u}$ with mean wind $\boldsymbol{U}$ (details are given below). to. Notice that the odor particles are not to be understood as actual molecules, but rather represent patches of odor with a concentration above the detection threshold of the agents. The entire system is placed inside a larger square box of size bL_x with reflecting boundary conditions for the agents. A complete list of parameters with their numerical values is given in the appendix F.

3.2.1 Response to private cues

The behavior elicited by private cues such as odor and wind speed is inspired by the cast-and-surge strategy observed in moths. We adopted a modified version of the “active search model" [91] that works as follows.

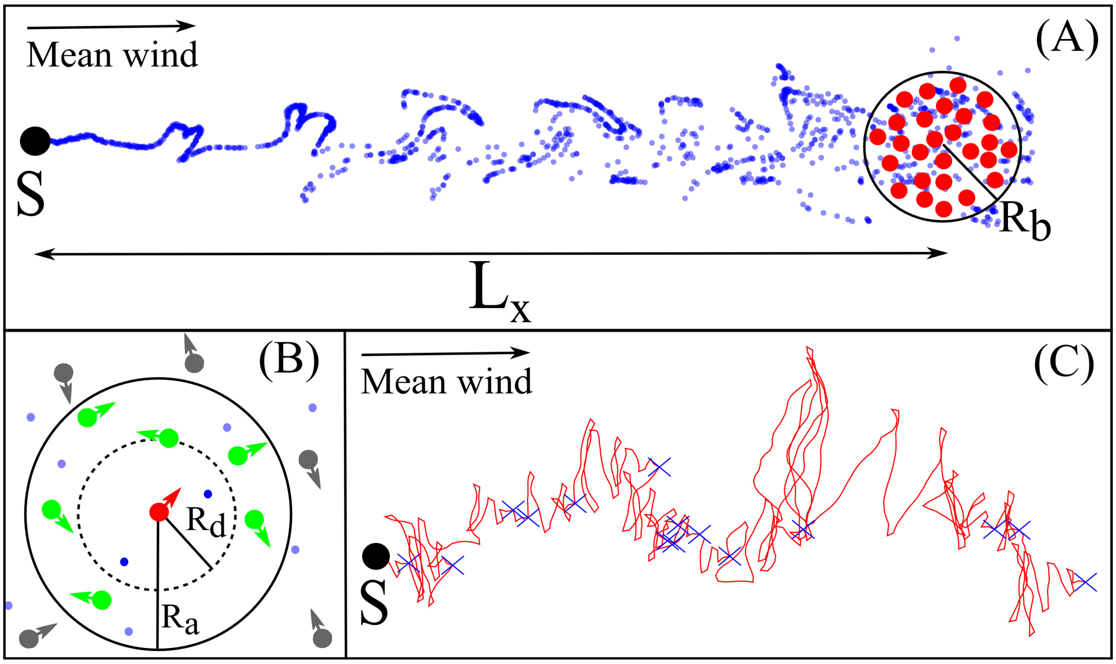


FIGURE 3.1: Collective olfactory search. (A) Odor particles dispersed by the turbulent environment are shown by semi-transparent blue dots emitted by the source S. Agents (red) are initially placed far from the source in a packed configuration. (B) Perception of an agent (red). Detected odor particles by the agent are shown as darker blue dots and neighbors of the agent are shown in green. Arrows indicate the instantaneous moving direction of agents. We set $L_x = 250R_d$, $R_b = 25R_d$, $R_a = 5R_d$, $R_d = 0.2$ $b = 2.5$ (C) Trajectory of an isolated agent performing the cast-and-surge program (see text). The locations where the agent detects the presence of odor particles are shown as blue crosses.

We assume that the agents have access to an estimate of the mean velocity of the wind as moths actually do via a mechanism named optomotor anemotaxis [92]. In the model this estimate $\hat{\boldsymbol{u}}(t)$ is an exponentially discounted running average of the flow velocity $\boldsymbol{u}$ perceived by the agent along its trajectory: $\hat{\boldsymbol{u}}(t) = \lambda \int_0^t \boldsymbol{u}(s) \exp[-\lambda(t-s)]ds$. The parameter λ is the inverse of the memory time: for $\lambda \to 0$ the estimate converges to the mean wind, while for $\lambda \to \infty$ it reduces to the instantaneous wind velocity at the current location of the agent. In the following we have taken $\lambda = 1$ which is of the same order of magnitude of the inverse correlation time of the flow. It is worth pointing out that the only effect of the wind is to provide contextual information about the location of the source. Indeed, in our model the agents are not carried away by the flow, an assumption that is compatible with the fact that the typical airspeed of moths and birds largely exceeds the wind velocity.

At each time interval Δt, the agent checks if there are odor particles within its olfactory range R_d (see Fig. 3.1B). If this is the case, then it moves against the direction of the current estimated mean wind at a prescribed speed v_0. When the agent loses contact with the odor cue, it starts the "casting" behavioral program: it proceeds by moving in a zig-zag fashion, always transversally to the current estimated mean wind, with turning times that increase linearly with the time from the last odor detection (a sample trajectory is shown in Fig 3.1C, see the appendix F for details about the implementation). We denote by $\boldsymbol{v}_i^{priv}(t)$ the instantaneous velocity of agent i prescribed by this cast-and-surge program. This is uniquely based on private cues and would indeed be the actual velocity adopted by the agent if it were acting in isolation.

3.2.2 Response to public cues

To describe the interactions among agents we have drawn inspiration from flocking and adopted the Vicsek model to describe the tendency of agents to align with their neighbors (see [93, 94] and references therein). We assume that an individual can perceive the presence of its peers within a visual range R_a (see Fig. 3.1B) and actually measure their mean velocity. According to this model, the behavioral response elicited in agent i by its neighbors is

$$\boldsymbol{v}_i^{pub}(t) = v_0 \sum_{j \in D_i} \boldsymbol{v}_j(t) \Bigg/ \left\| \sum_{j \in D_i} \boldsymbol{v}_j(t) \right\| , \tag{3.1}$$

where D_i is the disk of radius R_a centered around the position of the $i-$th individual. In order to account for errors in the sensing of the velocities of the neighbors we have added, as is customarily done, a noise term in the form of a rotation by a random angle $\boldsymbol{v}_i^{pub}(t) \leftarrow R(\theta)\boldsymbol{v}_i^{pub}(t)$. Here θ is independently sampled for each agent and at each decision time from a uniform distribution in $[-\eta\pi, \eta\pi]$. The strength of the noise η may range from zero (no noise) to unity (only noise): in the following we set $\eta = 0.1$.

In the absence of external cues, and for small enough noise, the group of agents described by this dynamics displays collective flocking and moves coherently in a given direction – totally unrelated with the source location, however.

3.2.3 Combining private and public information

To study collective olfactory search we then merged the two models above as follows. The velocity of the $i-$th agent is a linear combination of the two prescriptions arising from private and public cues, resulting in the update rule

$$\begin{aligned} \boldsymbol{v}_i(t) &= (1-\beta)\boldsymbol{v}_i^{priv}(t) + \beta \boldsymbol{v}_i^{pub}(t), \\ \boldsymbol{r}_i(t+\Delta t) &= \boldsymbol{r}_i(t) + v_0 \frac{\boldsymbol{v}_i(t)}{||\boldsymbol{v}_i(t)||}\Delta t. \end{aligned} \tag{3.2}$$

The parameter β, that we have dubbed "trust", measures the balance between private and public information. For $\beta = 0$ the agents have no confidence in their peers, they ignore the suggestion to align and behave independently by acting on the basis of the cast-and-surge program only. Conversely, for $\beta = 1$ agents entirely follow the public cues and discard the private information.

While it is reasonable to expect that for $\beta = 1$ the unchecked trust in public cues leads to poor performances in olfactory search, the nontrivial question here is rather if there is any value at all in public information; that is, in other words, if the best results are obtained for a finite β strictly larger than zero.

3.2.4 Modeling the turbulent environment

To complete the description of our model, we have to specify the underlying flow and the ensuing transport of odor particles. In our simulations the flow is given by an incompressible, two-dimensional velocity field, $\boldsymbol{u}(\boldsymbol{x}, t)$ with a constant, uniform mean wind $\boldsymbol{U}$ and statistically stationary, homogeneous and isotropic velocity fluctuations. The odor particles are considered as tracers whose position, $\boldsymbol{x}$, evolves according to $\dot{\boldsymbol{x}} = \boldsymbol{u}(\boldsymbol{x}, t)$. For the velocity fluctuations we first considered a stochastic flow and then moved to a more realistic dynamics where the flow obeys the Navier-Stokes equations. We wrote code to carry out simulations in Fortran programming language and the code is given in appendix G. [1]. The results of simulations are presented in next sections.

[1]For any problems to use the code as it given, I request to you to write to me (mihirdurve@gmail.com) for further support and source code files.

3.3 Results for the stochastic flow

This model flow is characterized by a single length and time scale and is obtained by superimposing a few Fourier modes whose Gaussian amplitudes evolve according to an Ornstein-Uhlenbeck process with a specified correlation time. The resulting flow is spatially smooth, exponentially correlated in time and approximately isotropic (see appendix F for details).

We studied the performance of collective search as a function of the trust parameter β while keeping the other parameters fixed to the values detailed in the appendix F. Initially, the agents are waiting in place without any prescribed heading direction until one of the agents detects the odor particles carried by the flow. After this event, agents move as per the equations of motion Eq. (3.2). Since the search task is a stochastic process, we run many episodes for each value of β to compute the average values of several observables of interest. A given episode is terminated when at least one of the agents is within a distance R_a from the source. At this stage we say that the search task is accomplished and agents have (collectively) found the odor source.

We focused our attention on four key observables: *(i)* the mean time needed to complete the task which measures the effectiveness of the search; *(ii)* the average fraction of agents that, at the time of completion, are close to the source; *(iii)* the order parameter which measures the consensus among members of the group about their heading direction; *(iv)* the degree of alignment of the agents against the mean wind.

In Fig. 3.2A we show the average time T for the search completion in units of the shortest path time $T_s = L_x/v_0$, which corresponds to a straight trajectory joining the target with the center of mass of the flock at the initial time. We observe that there exists an optimal value of the trust parameter $\beta \approx 0.85$ for which agents find the odor source in the quickest way. Remarkably, for this value we obtain $T \simeq 1.03\,T_s$: this means that the agent which arrives first is actually behaving almost as if it had perfect information about the location of the source and were able to move along the shortest path (see movie Beta=0.85.mp4).

This result has to be contrasted with the singular case of independent agents who act only on the basis of private cues ($\beta = 0$) which display a significantly worse performance (the time to complete the task is more than threefold longer) and

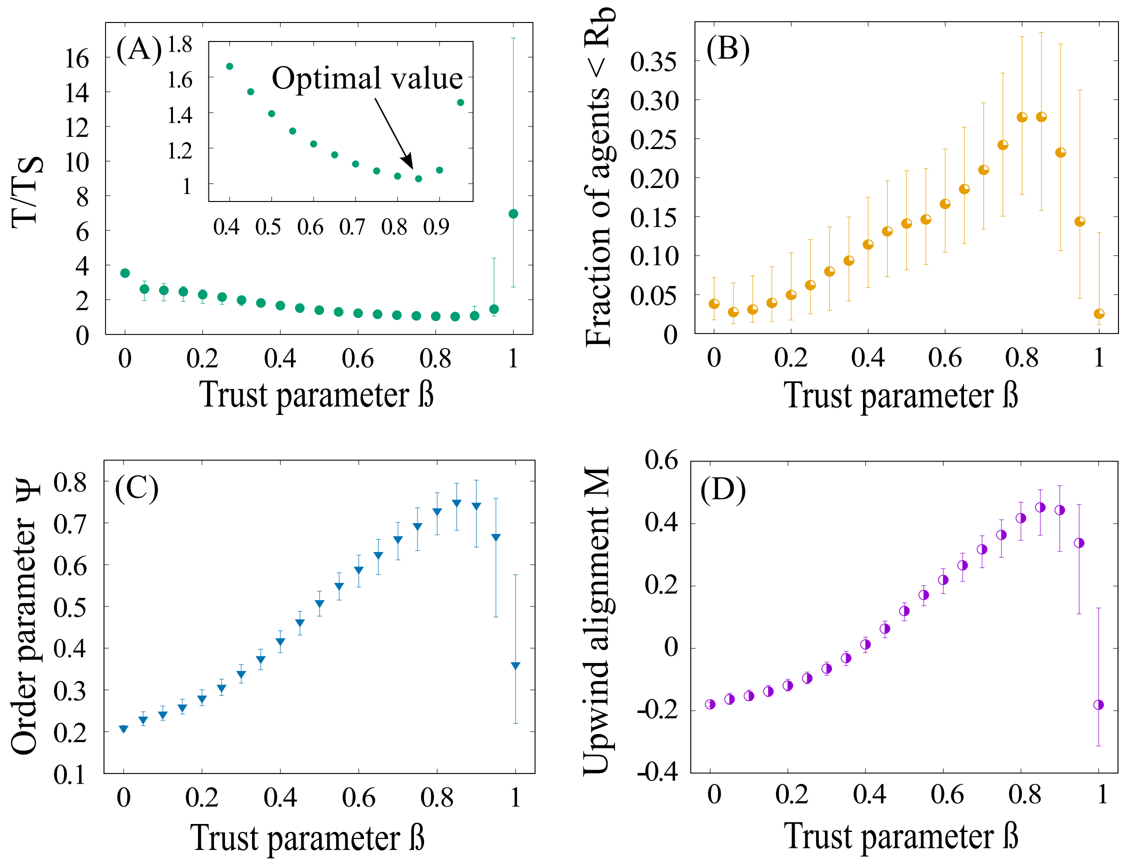


FIGURE 3.2: Collective olfactory search in a stochastic flow. (A) Average search time T for the first agent that reaches the target normalized to the straight-path time, $T_s = L_x/v_0$. The inset shows a blow-up of region close to the minimum. (B) Fraction of agents within a region of size R_b around the source at the time of arrival of the first agent reaching the target. (C) Averaged order parameter ψ (D) Average alignment against the mean wind M. For all data, the error bars denote the upper and lower standard deviation with respect to the mean value. Statistics is over 10^3 episodes. The parameters were set as $\lambda = 1$, $N = 100$, $J = 1$, $\eta = 0.1$, $v_0 = 0.5$, $\Delta t = 1$, $L_x = 50$.

move in a zig-zagging fashion (see movie Beta=0.00.mp4). It is also important to remark that the average time grows very rapidly as β increases above the optimum. As β approaches unity, agents are dominated by the interactions with their neighbors and pay little attention to odor and wind cues. As a result, they form a flock which moves coherently in a direction that is essentially taken at random. If by chance this direction is aligned against the wind, the task will be completed in a short time. However, in most instances the flock will miss the target and either turn because of the noise η or bounce on the boundaries until, again by sheer chance, some agent will hit the target (see movie Beta=0.95.mp4). This behavior results in a very long average time accompanied by very large fluctuations. As the outer reflecting boundaries are moved away by increasing b, this effect

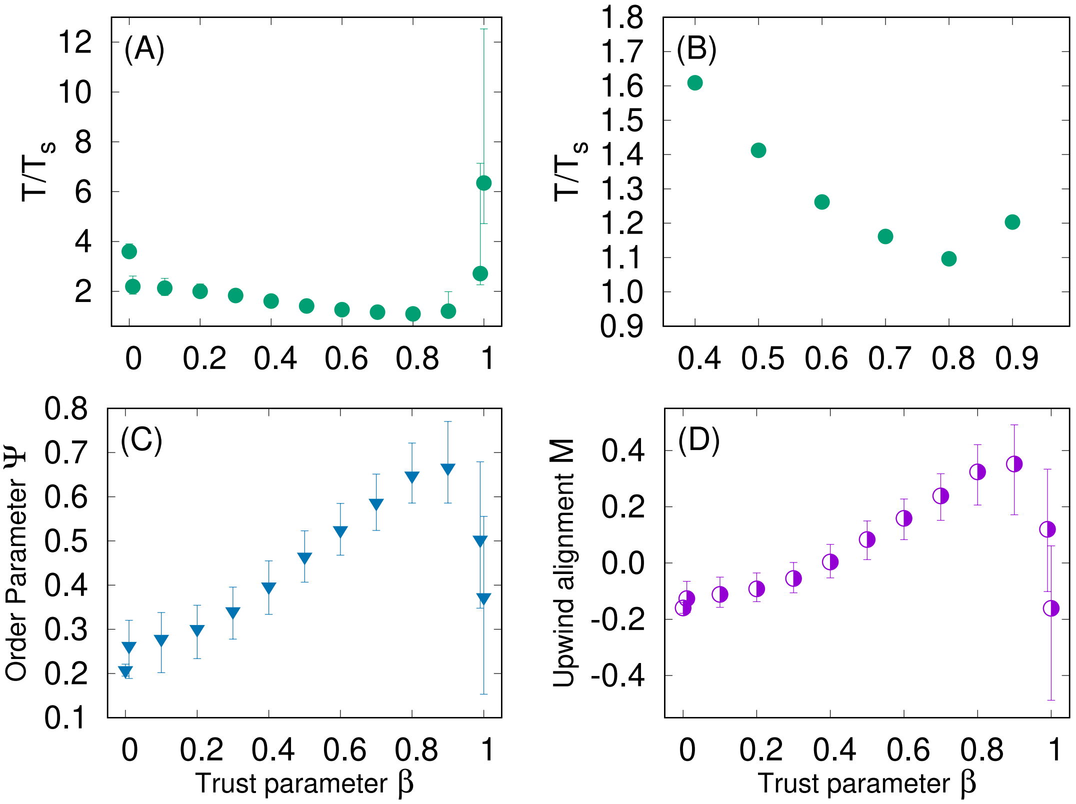


FIGURE 3.3: Collective olfactory search in a turbulent flow. A: Search time T for the first agent reaching the target normalized by the shortest-path time T_s. B: An enlargement of A that highlights the region close to the minimum. C: Mutual alignment order parameter ψ averaged over time and episodes. D: Average wind alignment M.

becomes more and more prominent.

Since we focused on the time of arrival for the first agent that reaches the source, it is natural to ask what has happened to the other members of the group that have been trailing behind. In Fig. 3.2B we show the average fraction of agents that are within a distance R_b (the initial size of the group) when the first agent reaches the target and the task is completed. This quantity is an indicator of the coherence of the group at the time of arrival. It turns out that this fraction has a maximum value ≈ 0.3 at about the same value of $\beta \approx 0.85$ that gives the best performance in terms of time. This means that on average about one third of the group has been moving coherently along the straight path that connects the initial center of mass of the flock to the target.

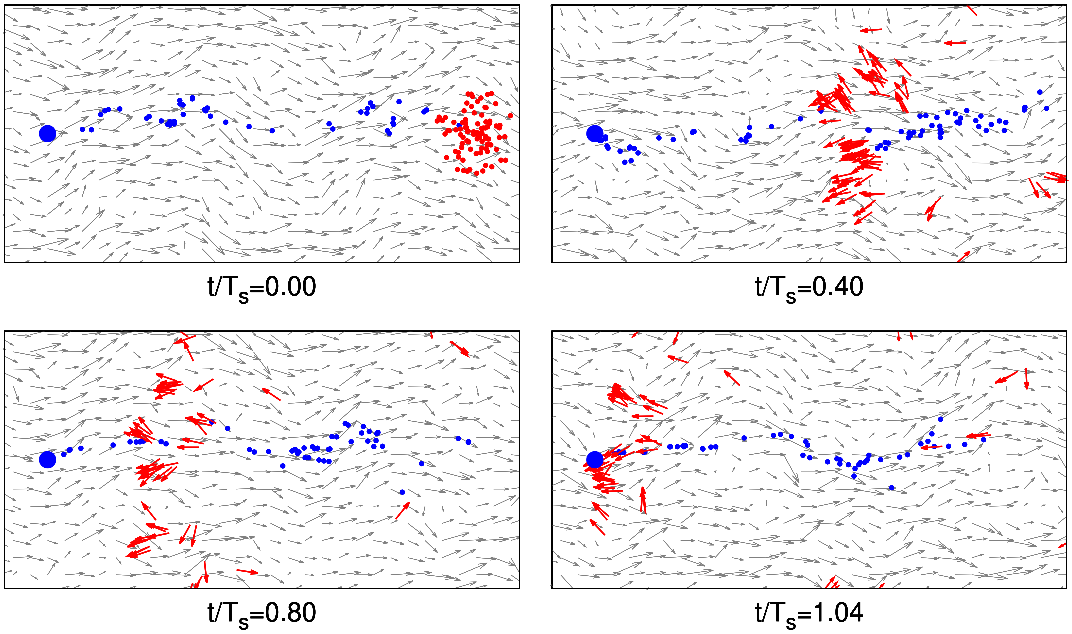


FIGURE 3.4: Collective olfactory search in a turbulent flow. Snapshots of the velocity field (grey arrows) at four different times t. The agents (red arrows) navigate in the turbulent flow with the optimal trust parameter $\beta = 0.8$. Blue dots represent odor particles dispersed by the flow, while the large blue circle corresponds to the source.

To quantify the consensus among agents about which direction they have to take, it is customary to introduce the order parameter

$$\psi(t) = \frac{1}{N v_0} \left\| \sum_{i=1}^{N} \boldsymbol{v}_i(t) \right\| . \tag{3.3}$$

When all the agents move in the same direction, whichever it may be, then $\psi = 1$. Conversely, if the agents are randomly oriented then $\psi \simeq N^{-1/2} \ll 1$. In Fig. 3.2C we show the order parameter averaged over all agents and all times along the trajectories. As in the previous case we observe a maximum around the range of values of β where performance is optimal.

Another parameter of interest is the upwind alignment of the agents

$$M(t) = 1 - \frac{1}{N} \sum_{i=1}^{N} ||\hat{\boldsymbol{U}} + \hat{\boldsymbol{v}}_i(t)|| . \tag{3.4}$$

When all the agents move upwind one has $M = 1$ whereas if they all move downwind $M = -1$. As shown in Fig. 3.2D the upwind alignment, averaged over time, has a maximum around $\beta = 0.85$ which again confirms that a large fraction of the group is heading against the mean wind even if it has access only to a local running time average (the memory time is $\lambda^{-1} = 1$, much shorter than $T_s = 100$).

The previous results point to the conclusion that there is a relatively narrow range of the trust parameter β, around 0.85, for which the collective olfactory search process is nearly optimal, i.e. the time to reach the target is close to the shortest possible one, and takes place with a remarkable coherence of the group.

3.4 Results for a turbulent flow

To test the robustness of our findings in a somewhat more realistic situation we also considered the case where the wind velocity is obtained from a direct numerical simulation (DNS) of 2D Navier-Stokes equations

$$\partial_t \omega + \boldsymbol{u} \cdot \boldsymbol{\nabla} \omega = \nu \Delta \omega - \alpha \omega + f \,, \tag{3.5}$$

where $\omega = \boldsymbol{\nabla} \times \boldsymbol{u}$, the forcing f acts at small scales so to generate an inverse kinetic energy cascade, that is stopped at large scales by the Ekman friction term with intensity α. In order to attain a statistically steady state, the viscous term with viscosity ν dissipates enstrophy at small scales. In this way we obtain a multiscale flow which is non-smooth above the forcing scale and smooth below it (see [95–97] for phenomenological and statistical flow properties). DNS have been carried out using a standard 2/3 dealiased pseudo-spectral solver over a bi-periodic $2\pi \times 2\pi$ box with 256^2 collocation points and 2^{nd} order Runge-Kutta time stepping, see appendix F for technical details. fields. In this flow the large scale of the velocity field is about half the size of the simulation box. The numerically obtained velocity field, for a duration of about 10 eddy turnover times, was then used to integrate the motion of odor particles in the whole plane exploiting the periodicity of the velocity field. Finally, the mean wind is then superimposed. Further details about the simulations are available in the appendix F.

Fig. 3.3 summarizes the main results obtained with the turbulent flow. As shown in the left panel, the average time taken by the first agent to reach the source is very similar to the one obtained for the stochastic flow. It displays a minimum time close to the shortest-path time $T_s = L_x/v_0$ at values of the trust parameter $\beta \approx 0.8$. The other observables display very similar features as the ones observed with the stochastic flow.

In Fig. 3.4 we show four snapshots of the agents at different times during the search process, for $\beta = 0.8$, i.e. close to optimality. The flock appears to be moving coherently in the upwind direction and the task is completed in a time $1.04\,T_s$ just a few percent in excess of the nominal minimal time.

3.5 Conclusions and discussion

We have shown that there is an optimal way of blending private and public information to obtain nearly perfect performances in the olfactory search task. The first agent that reaches the target completes the task by essentially moving in a straight line to the target. This behavior is striking, since in isolation agents move in a zig-zagging fashion (see Fig.3.1C). Interestingly, the information about odor and wind is essential to achieve this behavior, but its weight in the decision making is numerically rather small, about 20%. Although we do not expect that this number stays exactly the same upon changing the various parameters of the model, we suspect that there is a common trend for having optimal values of the trust parameter β at the higher end of its spectrum, that is, closer to unity. This may reflect the existence of a general principle of a "temperate wisdom of the crowds" by which public information must be exploited – but only to a point. In the present case, one way of summarizing our findings would be the following rule: follow the advice of your neighbors but once every four or five times ignore them and act based on your own sensations.

With reference to the remarkable similarity between searching in stochastic and turbulent flows shown by Figs. 3.2 and 3.3, we stress that this is likely due to the specific sensing mechanisms that we have chosen, which is essentially based on single-point single-time measurements. If private cues included consecutive inputs along the agent's trajectory and/or on spatially coarse-grained signals we expect that the results could have been more sensitive to the structure of small-scale and high-frequency turbulent fluctuations.

Our results suggest how to build efficient algorithms for distributed search in strongly fluctuating environments. It is important to point out, however, that our construction is inherently heuristic. Our model heavily draws inspiration from animal behavior, combining features of individual olfactory search in moths and collective navigation in bird flocks. A more principled way of attacking the collective search problem would be to cast it in the framework of Multi Agent Reinforcement Learning [59] and seek for approximate optimal strategies under the same set of constraints on the accessible set of actions and on the available private and public information. It would then be very interesting to see if the strategy discovered by the learning algorithms actually resembles the one proposed here, or points to other known behavior displayed by animal groups, or perhaps unveil some yet unknown way of optimizing the integration of public and private cues for collective search.

Appendix A

Description of states and actions

A.1 Description of states

Each agent has a fixed frame of reference attached to it. (see Fig A.1) For a purpose of implementing reinforcement learning algorithms, we discretized directions that an agent can perceive. In practice, we divided the full angular range of 2π in K_s number of bins. We labeled these bins from -16 to 15 covering the angular range $[-\pi$ to $+\pi)$. In this frame of reference, velocity $\mathbf{v}$ of the agent always falls in the bin labelled as '0' (see Fig. A.1). The agent perceives average direction of its neighbors as seen within its frame of reference. Thus, the state perceived by the agent has also falls in any of the bins lebeled from -16 to 15.

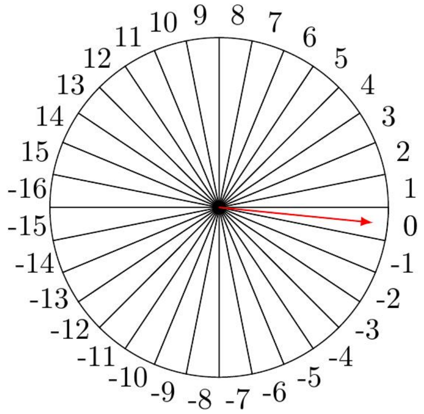


FIGURE A.1: Illustration of frame of reference attached to the agent and the way we lebeled the states as perceived by the agent. Red arrow indicates the velocity $\mathbf{v}$ of the agent.

A.2 Description of actions

A set of allowed actions $\mathcal{A}$ consists of turning and aligning with K_a directions. As before, the total angular range of 2π has been discretized. Fig A.2 shows the possible directions that the agent can take (black,red arrows) when $K_a = 32$. Fig A.3 shows the set of actions for $K_a = 7$ with maximum turning angle allowed θ_{max}.

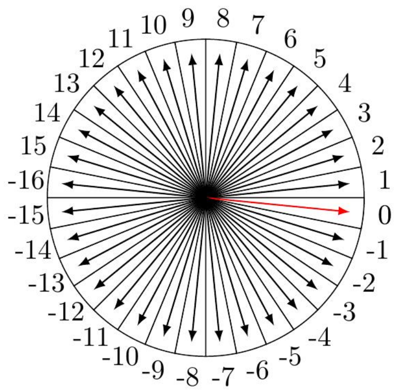


FIGURE A.2: Set of actions $\mathcal{A}$ for $K_a = 32$. Velocity $\mathbf{v}$ of the agent is shown by the red arrow.

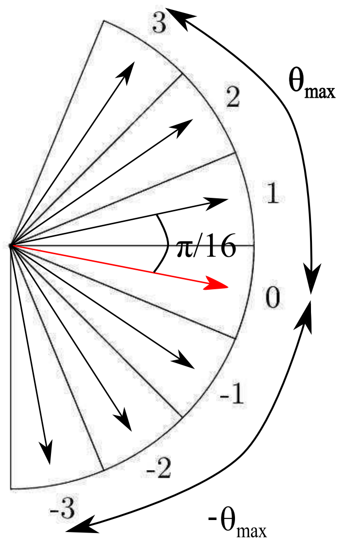


FIGURE A.3: Set of actions $\mathcal{A}$ for $K_a = 7$. Velocity $\mathbf{v}$ of the agent is shown by the red arrow.

Appendix B

On behavior of 'teacher agents'

B.1 Noise-free

In Sec. 2.3 we described a policy followed by the teacher agents in order to form a flock. The policy can crudely be summarized as each teacher agent must align with average direction of its neighbors if a required change in heading direction to do so is within a prescribed limit. If however, the required change in heading direction is more than the permissible limit then the agent must turn by maximum permissible angle. We implemented this policy with discrete turning angles to obtain results reported in chapter 2. We observed that for our choice of parameters in the noise-free case, the teacher agents formed highly polar ordered states ($\psi > 0.995$). To rule out any artifacts of the discrete nature of directions and as a check that indeed agents form ordered state with maximum turning angle $\theta_R < \pi$, we studied a model with restriction on the maximum turn allowed for the agents in a continuous description of directions. For simplicity, we shall refer to this model as 'Restricted angle self-propelling particle (RASPP)' model. The RASPP model is identical to the Vicsek model except for the constraints on the angular velocity of an agent. The rules to update velocity of an agent in RASPP model are depicted in Fig. B.1. Gao et al. [33] studied identical model and other operational details of our simulations are identical to the work of Gao et al.

In Fig. B.2 we show that for any positive value of θ_R, the agents starting from random initial conditions form highly ordered states. Counter-intuitively, we observed emergence of highly ordered states even for very small values of θ_R. However with

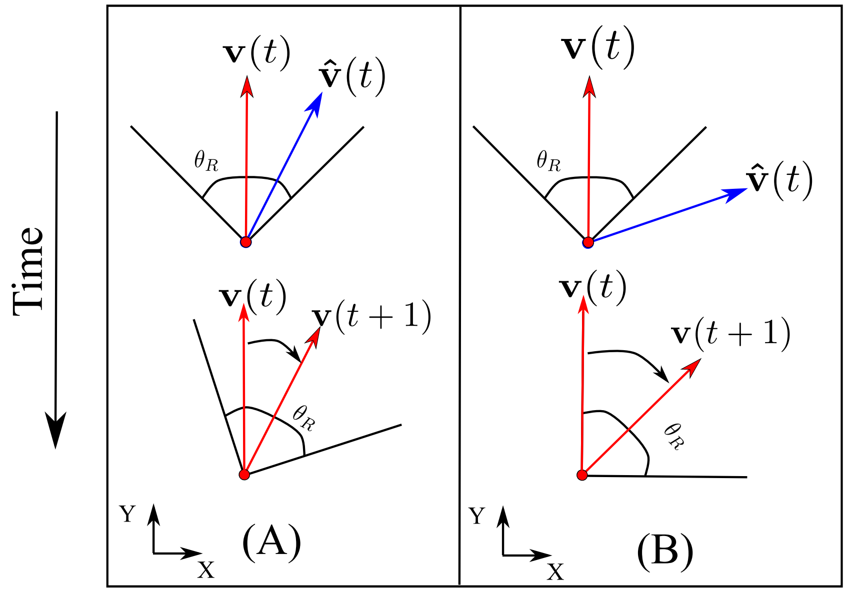


FIGURE B.1: Model with maximum turning angle θ_R in continuous space. (A) Depiction of the update rule when average direction of neighbors $\hat{\mathbf{v}}$ is within permitted turning angle θ_R. (B) Depiction of the update rule when average direction of neighbors $\hat{\mathbf{v}}$ is not within permitted turning angle θ_R.

smaller values of θ_R the transient time is larger (see Fig. B.2B). With this exercise we conclude that there are no undesired artifacts due to discrete nature of directions in formation of an ordered state. Also, it is clear that in the noise-free case, agents form highly ordered states for any non-zero value of maximum turning angle θ_R. For the implementation of the single agent reinforcement learning it is sufficient to have a flock of teacher agents moving synchronously. The details of the model used for simulating the flock are irrelevant. Therefore, we used the model with discrete directions to simulate a flock with teacher agents.

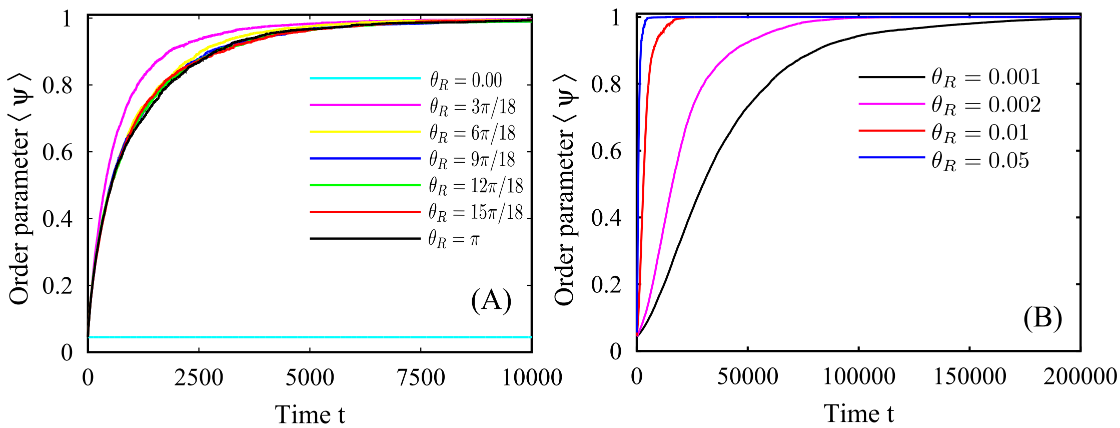


FIGURE B.2: Evolution of order parameter with time in noise-free case. Number of agents $N = 400$, Density of agents $\rho = 1.0$, Radius of interaction $R = 0.3$, speed of agents $v_0 = 0.1$.

B.2 With noise

Since the RASPP model that we used to simulate flocks of teachers is a variant of a well-known Vicsek model, it is imperative to make few comments about the similarities of RASPP model with the Vicsek model. We studied the effect of noise in the RASPP model. To implement noise in the system, we used the customary definition of noise that is used in studies of self-propelled particles [30]. We observed that similar to the Vicsek model, a system undergoes a phase transition from ordered state to disordered state as noise in the system is increased. We show the average value of order parameter ψ as noise η_π is varied for various values of θ_R in Fig. B.3. It is worthwhile to note that the RASPP model reduces to the Vicsek model for $\theta_R = \pi$. In Fig. B.3 we observed that the behavior of a system with RASPP model is identical to the behavior of a system with Vicsek model above a certain value of θ_R. However, for very small values of θ_R the behavior of a system with RASPP model deviates significantly form the behavior of a system with Vicsek model. It is observed that for larger values of θ_R, system undergoes a phase transition form ordered to disordered state. For very small values of θ_R, the system does not undergo a phase transition as noise is varied. Instead, the synchronization in the system is improved and the improvement is more and more significant as noise in the system is increased. This note-worthy observations of the RASPP model constitute for our ongoing work and we limit our comments on the RASPP model to the present state in this thesis.

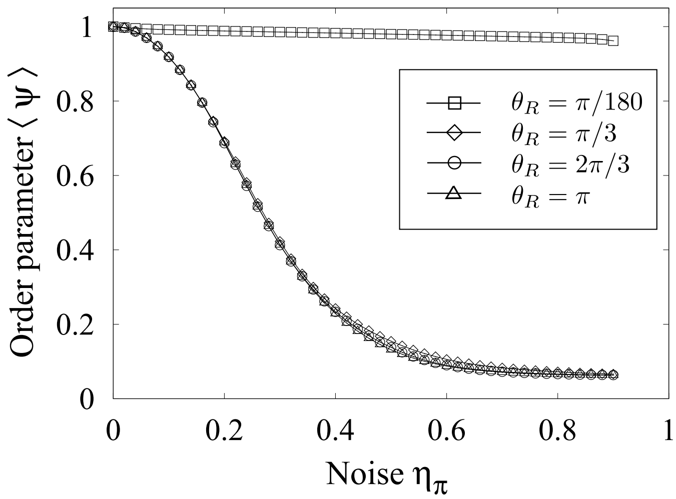


FIGURE B.3: Order parameter ψ as a function of noise η_π for various values of maximum turning angle θ_R. $N = 200$, $\rho = 1.0$, $R = 0.6$, $v_0 = 0.1$.

Appendix C

Reward for alignment

In this appendix, we present results of a single agent learning to flock with teachers with a reward scheme that encourage alignment. Such a reward scheme is a natural choice, since many prominent models for flocking, such as the Vicsek model focus on velocity alignment rules. Interestingly, reward for alignment is also obtained by the methods of inverse reinforcement learning (IRL). IRL techniques can be used to learn local reward function from observed global dynamics of expert systems. In one study [98], researchers implemented the IRL techniques to swarm of agents navigating as per the Vicsek model. They showed that the IRL techniques lead to high reward for high local alignment. Later, by training agents with this reward scheme, they showed that the agents perform as good as the agents with Vicsek model.

We carried out simulations with 200 agents flocking together by following the noise-free Vicsek model. A single naive learner with a goal to maximize the reward R was introduced in the flock. The perception of the learner is described in detail in appendix A. The local reward scheme for the agent is given by;

$$R(t) = ||\mathbf{v}_i(t)|| \; ||\hat{\mathbf{v}}(t)|| \; cos(\theta). \tag{C.1}$$

Here, $\mathbf{v}_i$ is velocity vector of the agent i, $\hat{\mathbf{v}}$ is average velocity vector of neighbors of the agent i and θ is the angle between the two vectors. We implemented RL techniques (see Chapter 2) for various densities of teachers ρ and radius of interaction R_a of the agents.

Fig. C.1 shows the average reward $\langle R \rangle$ earned by the learner as the training progresses. We observe that the learner earns near optimal reward irrespective of the chosen density of teachers ρ and interaction radius R_a. At the end of the training, we observe that the learner learns to align with the teachers and it behaves as the teachers do.

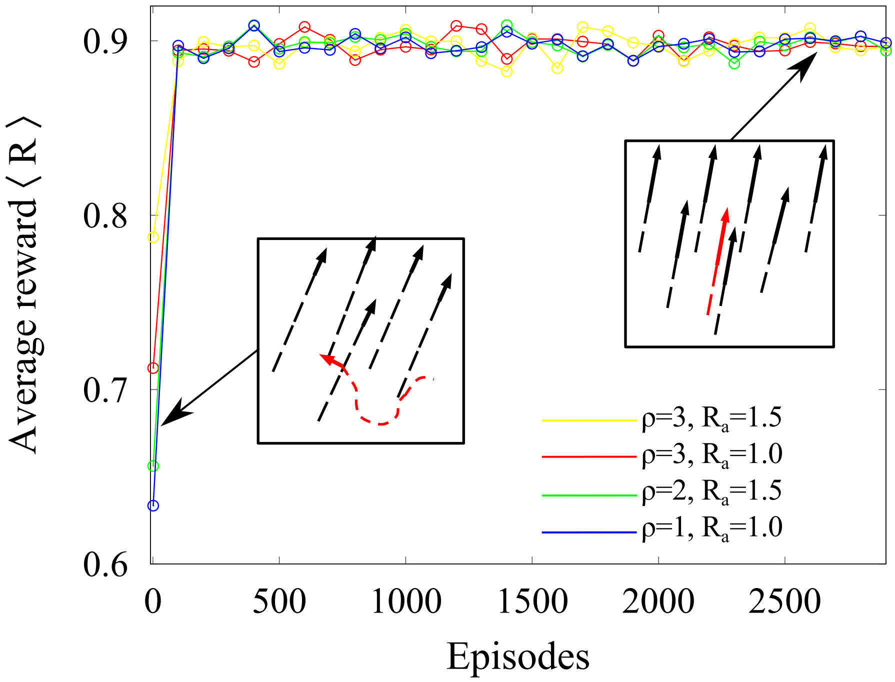


FIGURE C.1: Average reward for alignment earned by the learner. Insets show representative snapshots of the system in different phases of the training.

Fig C.2 shows the plot of Q-matrix of the learner at the end of the training. We observe that policy discovered by the learner is to align with the average direction of its neighbors to maximize the total reward.

With this exercise we show that a single learner can be trained with a reward scheme that encourages alignment with its neighbors to flock with them.

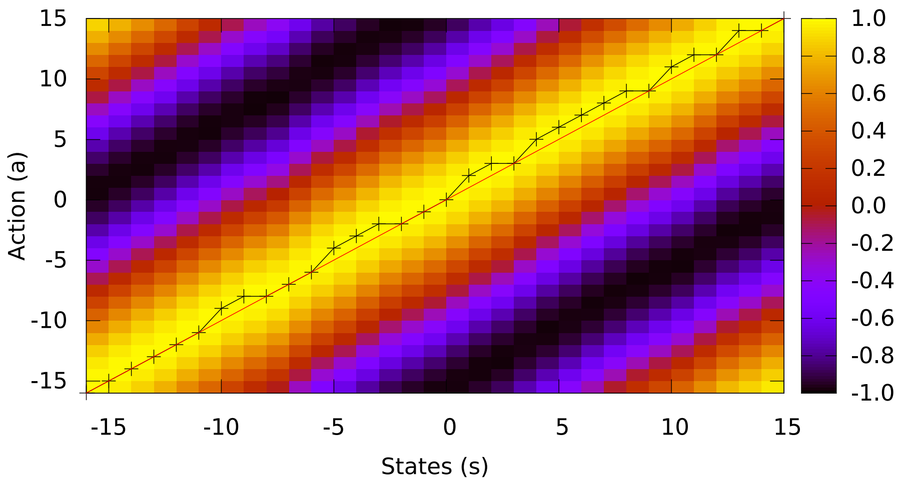


FIGURE C.2: Q-matrix of the learner at the end of the training. The colorbar shows the values in the Q-matrix. The black points indicate the best estimated action to perform in the given state. Red line (a*=s) is a guideline to the eye.

Appendix D

Reward for congregation

In this appendix, we present additional results (not presented in Chapter 2) for single as well as for multi-agent reinforcement learning systems. Here we set a reward scheme to encourage congregation of agents. Another equivalent point of view, which is presented in Chapter 2, is to encourage agents to lose minimum number of neighbors. The reward scheme implemented for results presented in this appendix is as follows.

$$R_i^{t+1} = \begin{cases} 0, & \text{if } n_i^{t+1} < n_i^t, \\ 1, & \text{otherwise} \end{cases} \tag{D.1}$$

where n_i^t is current number of neighbors.

We study and present results with this reward scheme for various choices of parameters and various choices of allowed actions.

D.1 Single agent

We carried out simulations with a flock consisting of 200 teacher agents that follow the Vicsek-like model for flocking. A single naive learner is introduced in the flock. Goal of the learner is to maximize the reward for congregation. The learner perceives a state s as the discretized average direction of its neighbors as described in appendix A. In the following section, we present results of simulations for the full set of actions i.e. $K_a = 32$.

D.1.1 With $K_a = 32$ actions

In this section, we present results of simulations with all possible actions allowed to the agent. Set of actions (32 in number) is shown in appendix. A. Fig D.1 shows average reward accumulated by the learner as training progresses for various initial densities ρ of the teachers and radius of interaction R_a of the agents. The error bars show standard deviation in values of the accumulated reward in 20 simulation experiments.

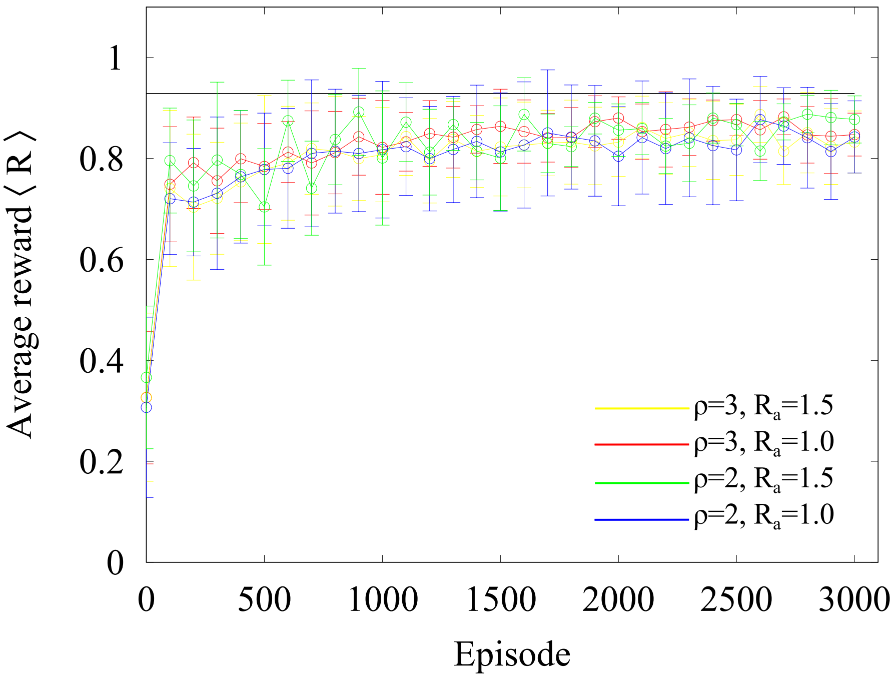


FIGURE D.1: Average reward for congregation earned by the learner for various system parameters. The error bars indicate standard deviation in the values in 20 independent simulations. Black line indicates value of policy evaluation of a model implemented for the teachers to flock.

We observe that, as the training progresses, average reward accumulated by the learner increases from low values and saturates to a higher value. The trained learner performs almost optimally. Optimal reward that the learner could accumulate is shown by the black line. This value is < 1 due to discrete nature of directions used in the percept of the agent and execution of actions. We also observe that the learner performs a random walk in the early phases of training (earning less reward) and learns to align with its neighbors to earn higher rewards. Fig D.2 shows Q-matrix of the agent at the end of the training process. White points indicate best estimated actions to perform in a given state s. It can be easily seen that the best estimated action is to align with average direction of its neighbors (which is same as the state s perceived by the agent).

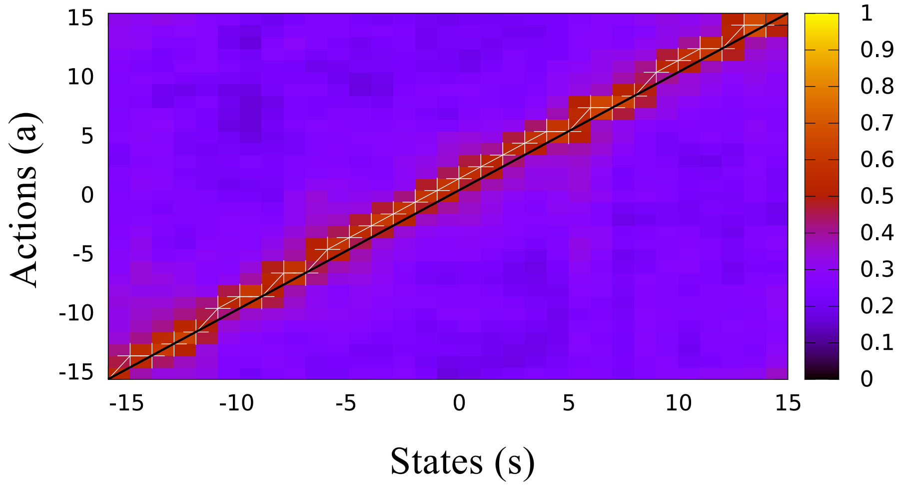


FIGURE D.2: Q-matrix at the end of one of the simulations. The white points show the best estimated action to perform in the given state.

D.2 Multi-agent

D.2.1 With $K_a = 7$ actions

In this section, we present results of simulations with 7 allowed actions with other choices of the parameters other than those chosen in Chapter 2. Fig. D.3 shows average reward earned by the agents for various choices of density ρ and radius of interaction R_a. In inset we show average Q-matrix computed over all Q-matrices of the agents at the end of the training in one of the systems. We observe that the average Q-matrix in other systems is same qualitatively. Policy learned by the agents dictates the agent to minimize the angle between its current velocity and average velocity of its neighbors by turning at an angle $\leq \theta max$.

As the training progresses, we measured direction consensus among the agents by computing polar order parameter ψ. The evolution of the polar order parameter is shown in Fig. D.4. We observe that, in all systems, the agents forms a highly polar order states in a later phase of the training.

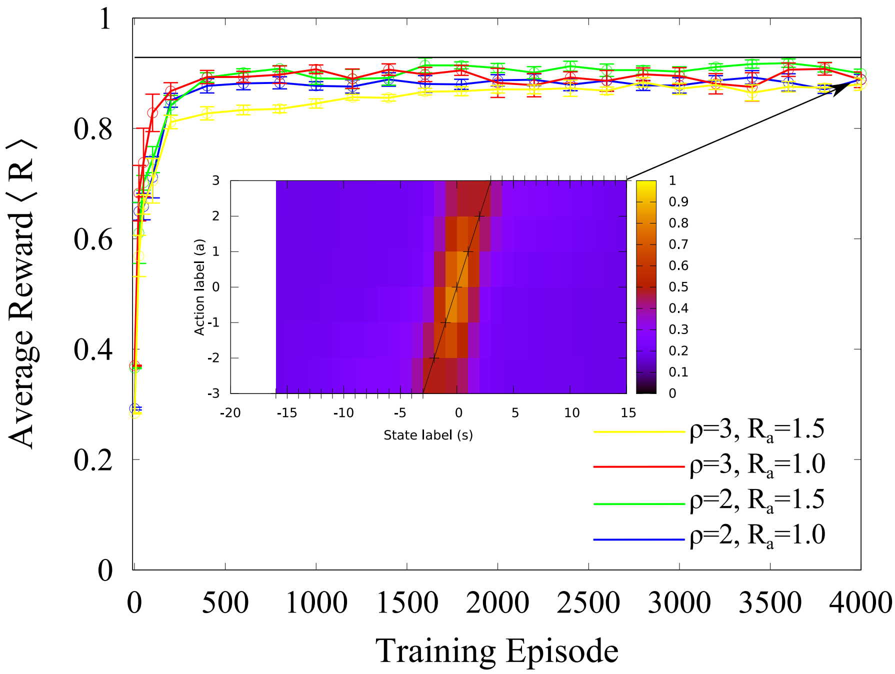


FIGURE D.3: Evolution of the order parameter ψ as the training progresses.

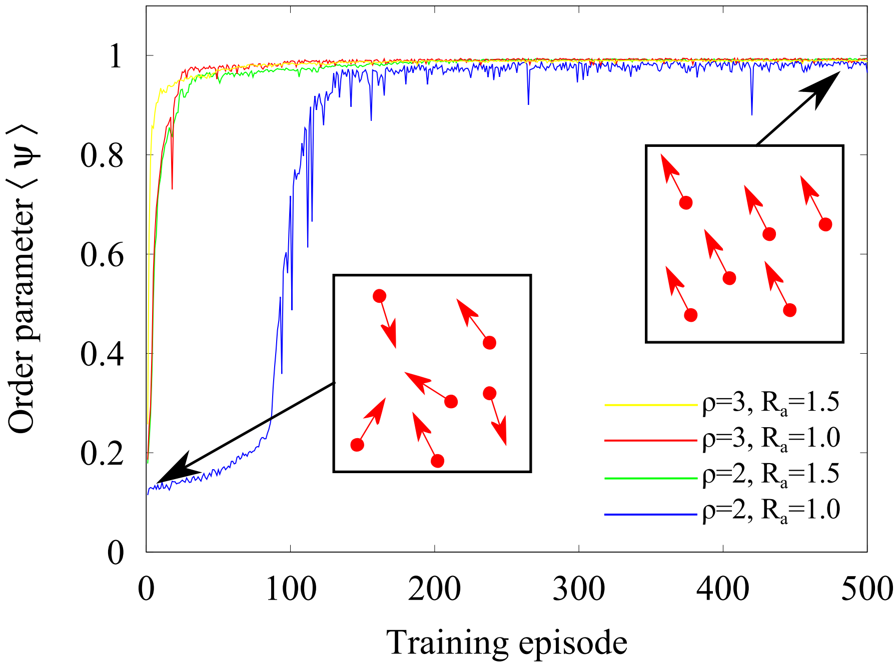


FIGURE D.4: Evolution of the order parameter ψ as the training progresses.

D.2.2 With $K_a = 3, 5, 7, 9$ actions

In Chapter 2, we presented results for $K_a = 7$ allowed actions with θ_{max} as the maximum turn allowed. Here we show results for various choices of the parameters. First, we present results with a variable number of actions allowed. Fig D.5 compares average reward earn by the agents as training progresses with various number of allowed actions. A set of actions consists of K_a elements. By increasing a number of elements in the set of actions, we essentially increased the maximum turning angle θ_{max} allowed to an agent. The other possibility to increase a number of actions by fixing θ_{max} is explored in Chapter 2.

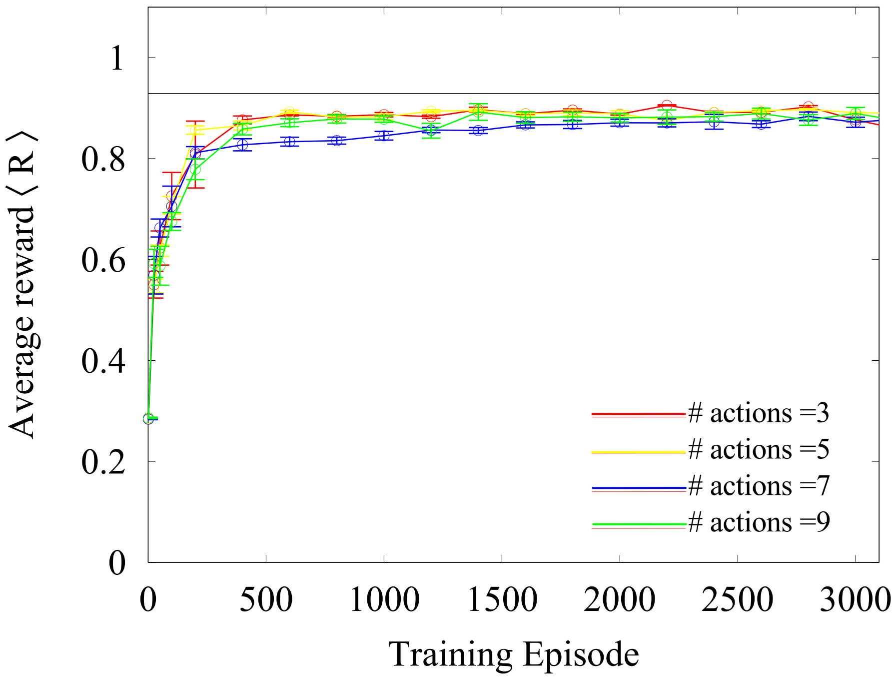


FIGURE D.5: Average reward for congregation earned by the learner for various system parameters. Error bars indicate standard deviation in reward earned by each agent. Black line indicates the maximum reward possible by following the discretized Vicsek model.

There seems to be no strong dependence on average reward that the agents earn with the number of actions. In Fig. D.7, we plot policy discovered by the agents. In this plot, we show best estimated action a^* in the state s. The best estimated action is an action with highest Q-value in the Q-matrix for a given state s. We observe that, regardless of a number of actions allowed to the agents, the policy discovered by the agents is identical. This policy dictates the agent to execute the action that minimizes the angle between velocity of the agent $\mathbf{v}$ and average direction of its neighbors $\hat{\mathbf{v}}$.

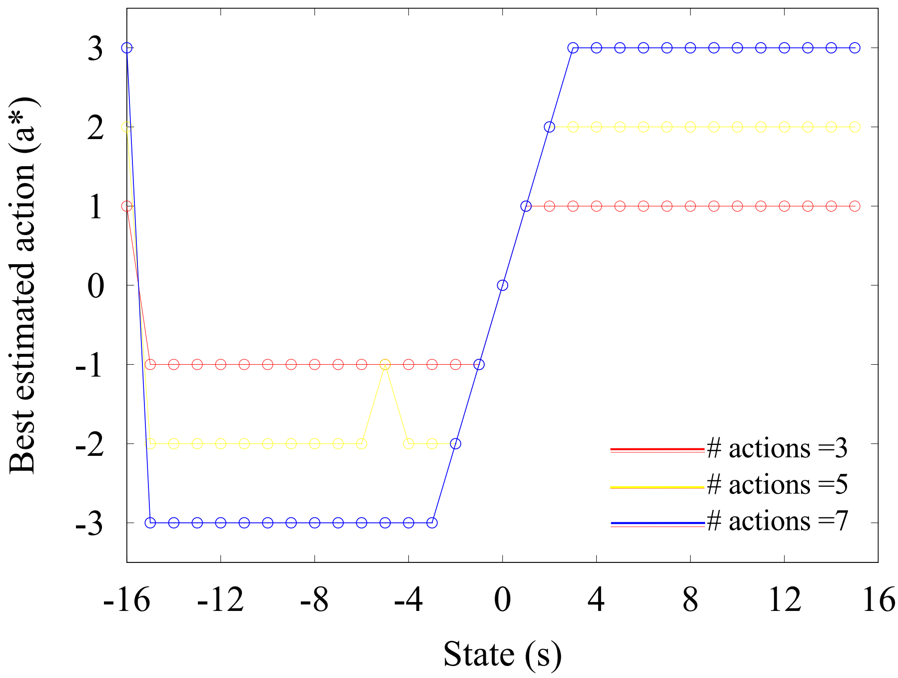


FIGURE D.6: Best estimated action to perform in a given state. The points correspond to the action with highest Q-value for a given state in the Q-matrix.

As the training progresses, we measured direction consensus among the agents by computing polar order parameter ψ. The evolution of the polar order parameter is shown in Fig. D.7. For all these cases, we observed, that the agents form highly ordered states. It might appear counterintuitive to see that the agents could form a polar ordered states even when they can turn only by a few discrete angles. However, Gao et al. [33] showed that such a restrictions on the angular velocity of agents in fact increase the direction consensus.

D.2.3 With $K_a = 32$ actions

We increased number of actions systematically till we reached the full set of actions i.e. $K_a = 32$. This choice restores the rotational symmetry in the set of actions and we observed that, the policy discovered by the agents is not unique in different simulation experiments with random initial conditions. We carried out many independent simulations with $K_a = 32$ with random initial conditions. The other parameters such as number of agents N, density ρ, radius of interaction R_a were held constant. In Fig. D.8, we show results of 4 such representative simulations. We observed that, the agents accumulate roughly the same reward in different simulation experiments but the average Q-matrix at the end of the training is not

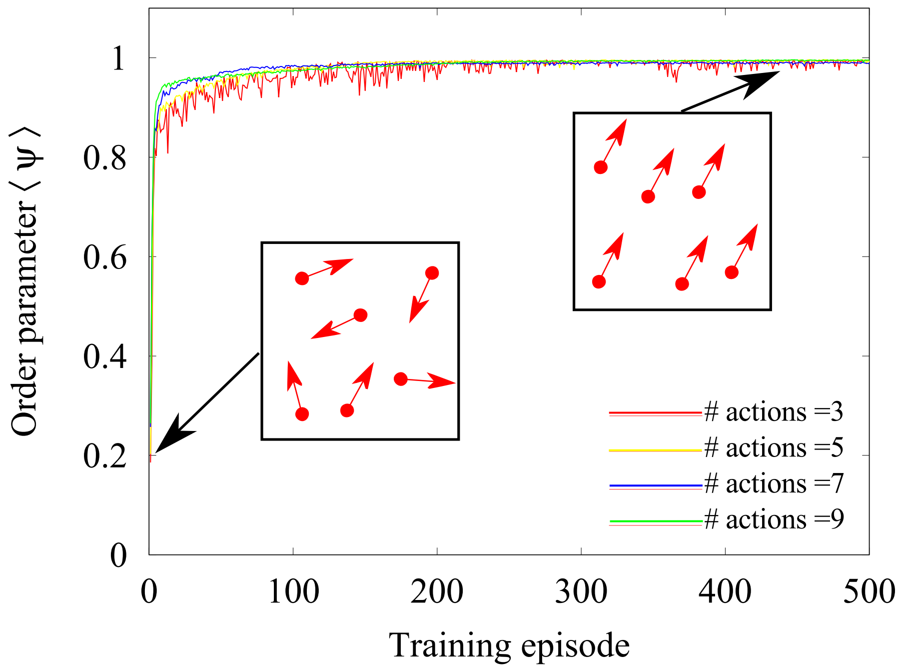


FIGURE D.7: Evolution of the order parameter ψ as the training progresses.

qualitatively same in each of the simulations thereby leading to different policies. One of the policies discovered by the agents is the Vicsek model like policy in which an agent aligns with its neighbors. However, the other discovered policies dictate an agent to move in a direction separated by angle ω' from average direction of its neighbors. We observed that, with these policies, all the agents are aligned in a common direction at a given time but all of them turn by an angle ω' in the next step. Thus, the ensuing state of the agents results in highly polar ordered states as captured by the polar order parameter ψ shown in Fig. D.9. (see movie Appendix1.mp4)

We carried out 1000 simulation experiments and cannot conclude if any particular value of ω is favored by the agents. In Fig. D.10, we show number of times the policy with angular velocity ω was discovered by the agents. To draw a conclusion with high confidence, we shall need to carry out more simulations which is, unfortunately, not possible at this time.

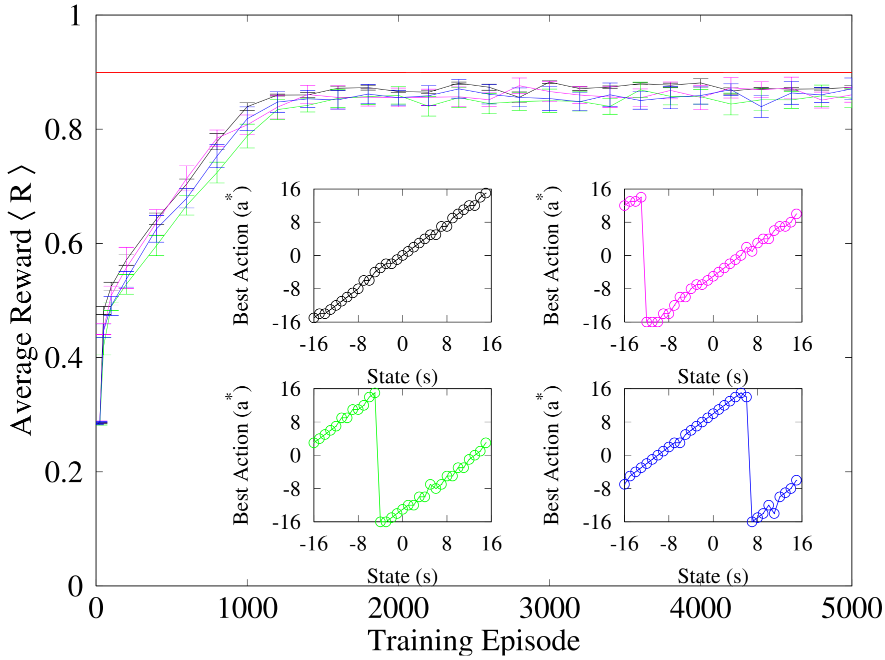


FIGURE D.8: Average reward earned by agents in 4 representative simulations. Agents earn the same reward by discovering different policies shown in the insets.

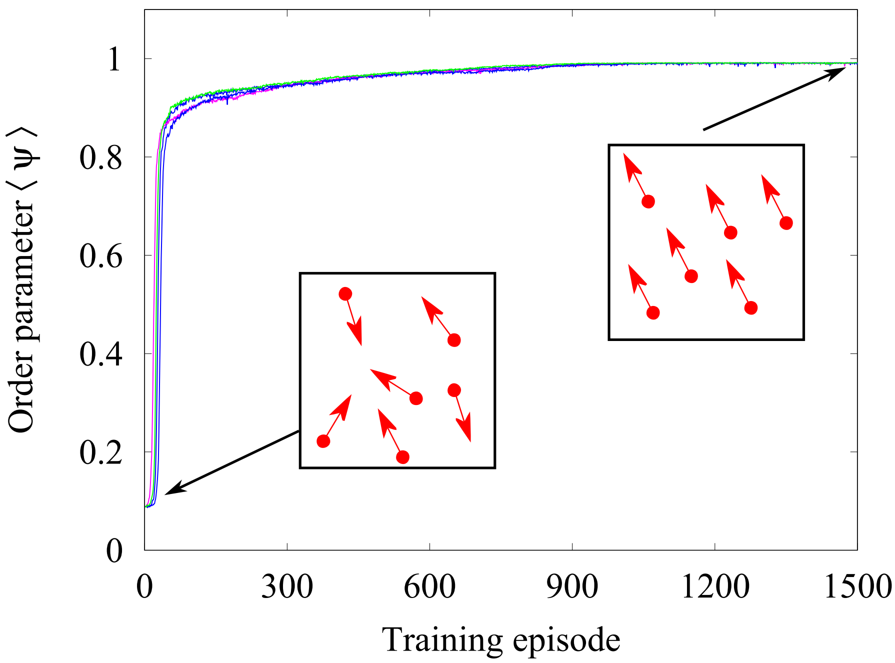


FIGURE D.9: Evolution of order parameter ψ in various simulation experiments.

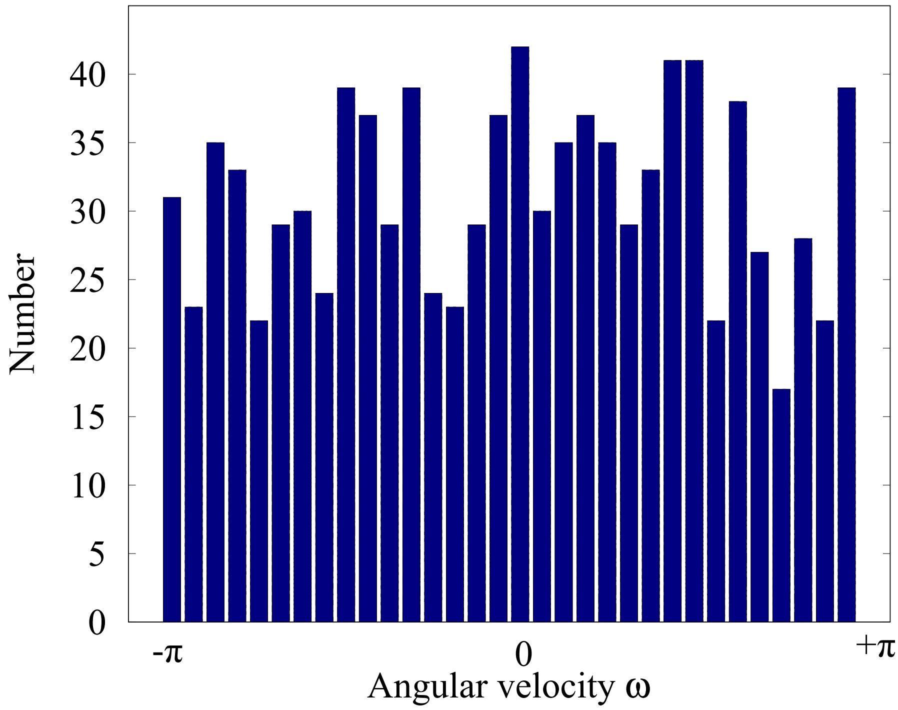


FIGURE D.10: Frequency of a discovered policy with a given angular velocity. The total number of independent simulations carried out were 1000.

Appendix E

MARL with limited field of view and noisy measurements

E.1 Learning to flock with a limited field of view

Biological agents such as bird or fish usually have a limited perception of their surroundings. For example, if one considers visual cues then most of the animals have an anisotropic perception due to their limited field of view. For example, the cyclopean field of view (i.e., the combined field of view of both eyes [25]) of the grey-headed Albatross is about 270° in the horizontal plane [26] and for humans it is 180°. We have then introduced this limited vision for the agents in our simulations by implementing a restricted field of view.

In practice, we define the neighborhood of an agent spanned by its field of view as a sector of a circle with radius R and half opening angle, or “view angle”, ϕ (see Fig E.1). For $\phi < \pi$, the agents interact with anisotropic and non-reciprocal interactions [23]. An agent i can perceive the velocity of other agents which are within its neighborhood.

We have varied the view-angle ϕ in our simulations while keeping the neighborhood area constant by scaling up the radius of interaction R appropriately. As in the case of full view discussed in the main text, agents start from random initial conditions with an optimistic Q-matrix (i.e. all values in the Q-matrix are set to 0). Each agent has its own Q-matrix. At each time-step, each agent processes its sensorial input and computes the perceived state s_t of the environment according

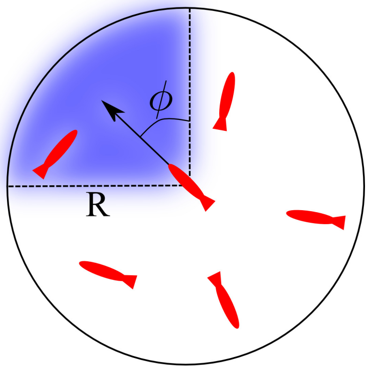


FIGURE E.1: Neighborhood Nb_i(shown by blue shaded region) of an agent i placed at the center of a circle of radius R.

to Eq. (2) of the main text, that is the average velocity of agents in its neighborhood Nb_i. Given the state s_t, an agent performs an action a_t, i.e. it changes its velocity, given by Eq. (3), according to the policy π. An agent updates its policy π according to a (negative) reinforcement signal in the form of a cost c_i^{t+1} for decreasing the number of neighbors. The cost is computed according to Eq. (4). It is important to note here that while calculating the cost for an agent, only the agents in its field of view are taken into consideration as neighbors. To update policy π, an agent modifies values in its Q-matrix for the state-action pair just visited (i.e. $Q(s_t, a_t)$) according to Eq. (5). The updated policy is based upon the modified Q-matrix according to the ϵ-greedy exploration scheme given by Eq. (6). According to this scheme an agent performs the best estimated action (the one that minimizes the total expected cost) with probability $1-\epsilon$ or a random action with probability ϵ. In our simulations, we used the following scheduling scheme for the exploration rate ϵ.

$$\epsilon(E) = \begin{cases} 1 - 0.002(E-1), & \text{if } E < 500 \\ 0, & \text{otherwise} \end{cases} \tag{E.1}$$

Here, E is the index number of an episode. The training phase starts with an initial value of $\epsilon = 1$ which is then linearly reduced to zero. With $\epsilon = 0$ agents always perform the action that is estimated to minimize the expected cost.

In Fig. E.2A we show the average cost, i.e. the rate of loss of neighbors, for N independently learning agents with limited field of view. Agents starting from higher cost learn to reduce the rate of loss of neighbors in few hundred training episodes. The resulting Q-matrix, at the end of the training, averaged over all the agents is shown in Fig. E.2B. The performance of the agents and the discovered policy are very close to the respective ones for agents with full field of view (i.e. $\phi = \pi$) as described in the main text.

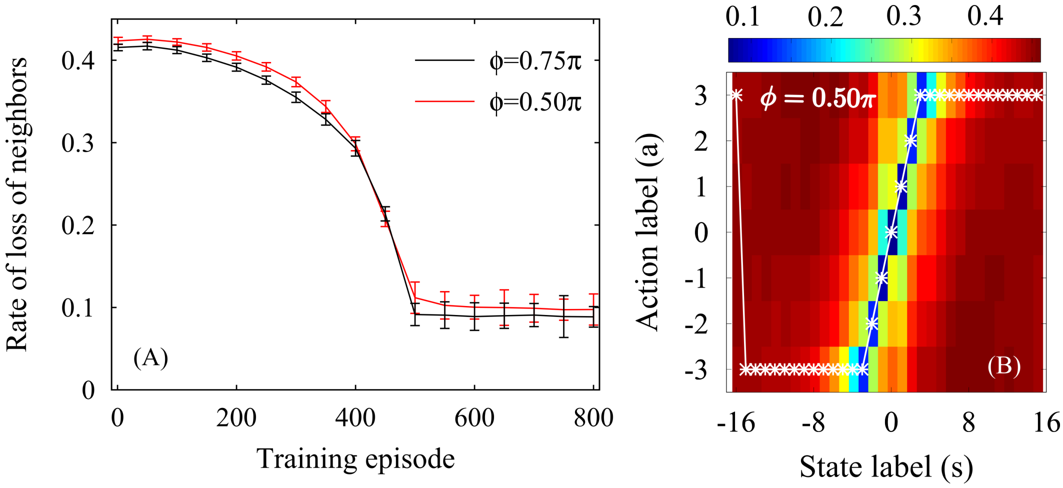


FIGURE E.2: Result for limited angle of view. Each training episode consists of 10000 time-steps. Number of agents $N = 200$, density of agents $\rho = 2$ agents/unit area, area of neighborhood $A = \pi$, $\{Ks, Ka\} = \{32, 7\}$. (A)Performance of multi-agent system as training progresses. Error bars indicate standard deviation in the average values for each agent. (B) Average Q-matrix at the end of the training with $\phi = 0.50\pi$. White points indicate actions with estimated minimum cost for given state. The colors represent values in the Q-matrix.

E.2 Limited field of view and noisy observations

In the main text we considered a simplified model whereby an agent can precisely measure the mean velocity of its neighbors and also their distances in order to define its state. Here we relax these assumptions to account for imperfect measurements by adding an observational noise on the measurement of distance and mean velocity of neighbors.

To begin with, we investigate the effects of observational noise in the measurement of distance between two agents for agents with limited field of view ϕ.

We defined noise in the measurement of distance between the agents as follows. An agent i perceives a distance d'_{ij} to another agent j as; $d'_{ij} = |d_{ij} + \eta_R(0, \sigma)|$ where d_{ij} is the true distance between agents i and j and $\eta_R(0, \sigma)$ is a random number chosen from Gaussian distribution of random numbers with 0 mean and standard deviation σ. An agent i “sees” agent j if it is within the field of view of an agent i and the perceived distance $|d'_{ij}|$ is less than R. In Fig. E.3A we show the performance of agents as the training progresses which is similar to the one for perfect observations. The discovered policy that minimizes the rate of loss of neighbors is shown in Fig. E.3B and is identical to the one obtained in the noise-free case and with full vision.

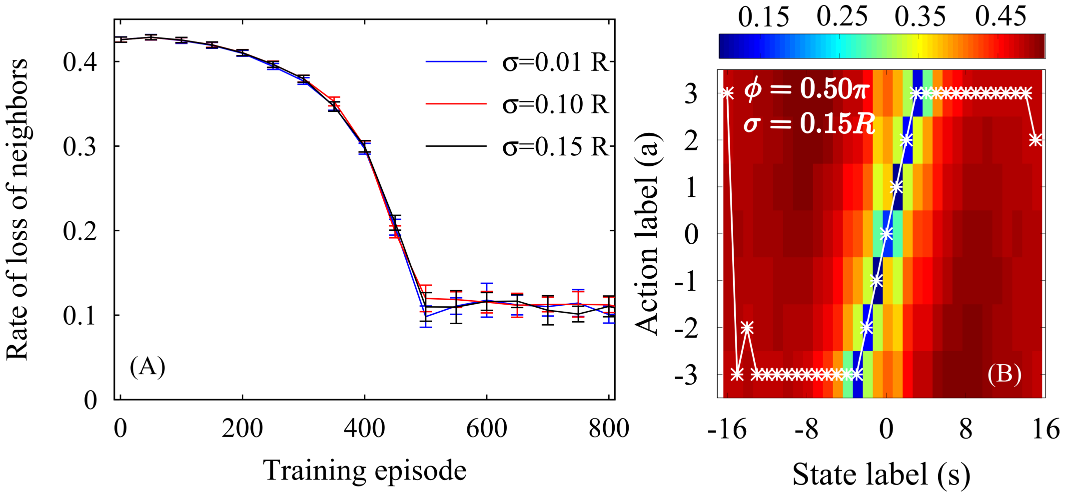


FIGURE E.3: Learning with a limited angle of view and noisy measurements of distance. Each training episode consists of 10000 time-steps. Number of agents $N = 200$, density of agents $\rho = 2$ agents/unit area, radius of interaction $R = 1.41$, view-angle $\phi = 0.50\pi$ $\{Ks, Ka\} = \{32, 7\}$. (A)Performance of multi-agent system as training progresses. Error bars indicate standard deviation in the average values for each agent. (B) Average Q-matrix at the end of the training with $\sigma = 0.15R$. White points indicate actions with estimated minimum cost for given state. The colors represent values in the Q-matrix.

Now, in addition to the limited field of view and observational noise in the measurements of the distances we add another noise on the measurements of the mean velocity of neighbors. The perceived average velocity of neighbors $\mathbf{P'}_i$ by an agent i is given as $\mathbf{P'}_i = \mathcal{R}(\theta)\mathbf{P}_i$. Here, $\mathcal{R}$ is an rotational operator that rotates the vector it acts upon by an angle θ. An angle θ is chosen randomly and uniformly within the range $[-\eta_a\pi, +\eta_a\pi]$. η_a is strength of the observational noise in the range $[0, 1]$. $\mathbf{P}_i = (\sum_{j\in Nb_i} \mathbf{v}_j^t)/n_i$ (n_i is a number of neighbors of an agent i). This definition of noise is essentially equivalent to the noise customarily used in the Vicsek model.

In Fig. E.4A, we show the performance of the multi-agent system with a limited field of view and observational noise bit in position and velocity. We observed that up-to certain strength of the mean velocity noise η_a, the discovered policy by the agents to minimize the cost is identical to the policies discovered in the noise-free and full vision case as described in the main article. However, above a certain level the agents appear to discover policies which are different than the one obtained with small or no error in velocity. This interesting observation probably deserves further analysis, that we do not pursue here, in order to ascertain the causes of this behavior.

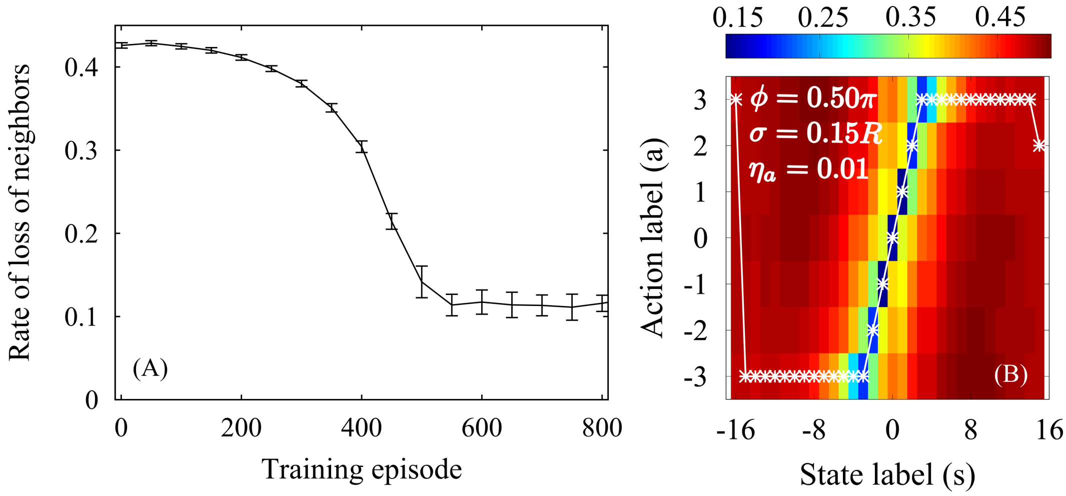


FIGURE E.4: Result for limited angle of view and observational noise on positions and velocity. Each training episode consists of 10000 time-steps. Number of agents $N = 200$, density of agents $\rho = 2$ agents/unit area, radius of interaction $R = 1.41$, noise in the measurement of distance with $\sigma = 0.15R$, noise strength $\eta_a = 0.01$, view-angle $\phi = 0.50\pi$ $\{Ks, Ka\} = \{32, 7\}$. (A)Performance of multi-agent system as training progresses. Error bars indicate the standard deviation in the average values for each agent. (B) Average Q-matrix at the end of the training. White points indicate actions with estimated minimum cost for given state. The colors represent values in the Q-matrix.

So far we have measured performance of the multi-agent system with the cost incurred by agents for losing their neighbors. We observed that with a limited view-angle and up to a certain level of observational noise the agents learn how to minimize the cost. It is then natural to ask about the structure of the swarms that form under the discovered policies. As is customarily done, we have computed polar order parameter ψ to measure alignment in the group. In Fig. E.5 we plot the polar order parameter ψ against the cost during the learning process. We have observed that even with limited vision and noisy observations, the agents form highly polar ordered states in which agents move in a common heading direction at any given instance.

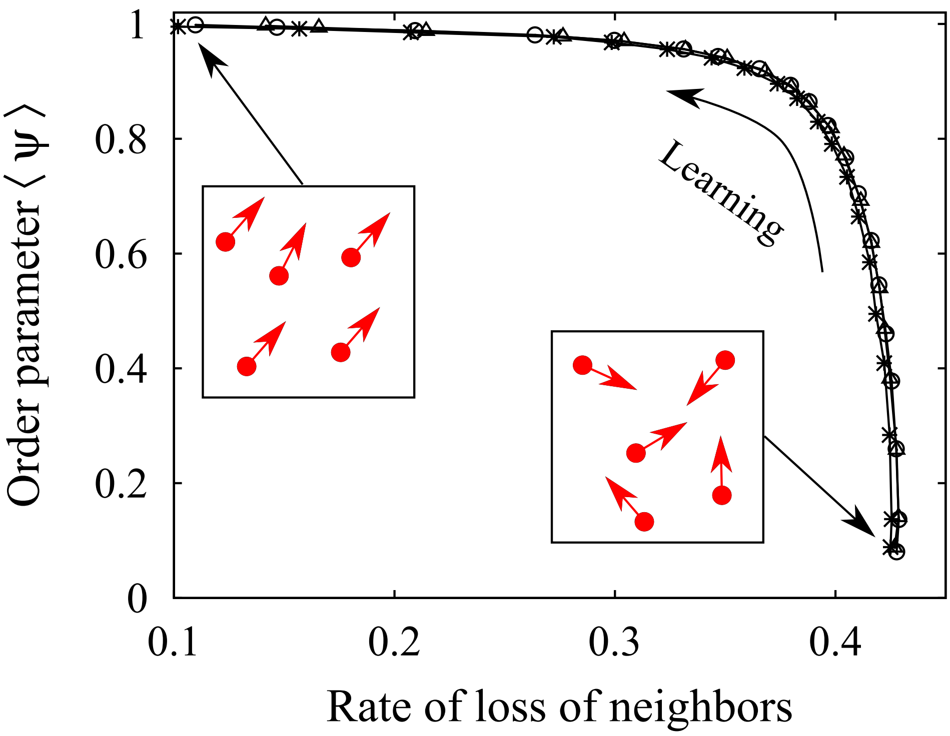


FIGURE E.5: Average polar order parameter $\langle\psi\rangle$ versus the rate of loss of neighbors. Number of agents $N = 200$, density of agents $\rho = 2$ agents/unit area, radius of interaction $R = 1.41$. Asterisk: $(\phi = 0.5\pi, \sigma = 0.0, \eta_a = 0.0)$, circle: $(\phi = 0.5\pi, \sigma = 0.15R, \eta_a = 0.0)$, triangle: $(\phi = 0.5\pi, \sigma = 0.15R, \eta_a = 0.01)$. In the insets we show a snapshot of a subset of naive and trained agents.

Appendix F

Implementation of the turbulent flow

F.1 Details on the implementation of the *cast and surge* algorithm

The cast-and-surge strategy describes the motion of an agent elicited by the private information acquired. In the following, we provide its algorithmic implementation. As discussed in the main text, the strategy consists of two components: the estimate of the mean wind velocity $\hat{\boldsymbol{u}}(t)$ and a behavioral response to the presence or absence of an odor within its olfactory range (circle with radius R_d) at a given time. In particular, we assume that the agent can measure the instantaneous local wind at every discrete times δt, which is the integration step used to advance the odor particles. Using such measurements, the agent can construct the estimate of the mean wind velocity $\hat{\boldsymbol{u}}(t)$ by taking an exponentially discounted running average of the perceived flow velocity $\boldsymbol{u}$, as described in the main text.

Without loss of generality, for the purpose of describing the algorithm, we take a simple case where the agent perfectly estimates the mean wind direction at all times (i.e. $\hat{\boldsymbol{u}}(t) = \boldsymbol{U}$). As explained in the main text, this corresponds to the choice $\lambda = 0$ in the memory kernel. Further, we assume that the agent moves every discrete time t separated by the interval $\Delta t \gg \delta t$, called the decision time. During the time Δt, apart from estimating the mean wind direction every δt, the agent can detect the odor particles within its olfactory range. From a practical perspective, Δt corresponds to the time taken by the agent to make the decision to move by processing the acquired information about the mean wind and the odor detection. Following an extension to continuous space of the cast-and-surge on-lattice algorithm described by Balkovsky et al. [91], we define the behavioral response of the agent as follows (see Fig F.1A):

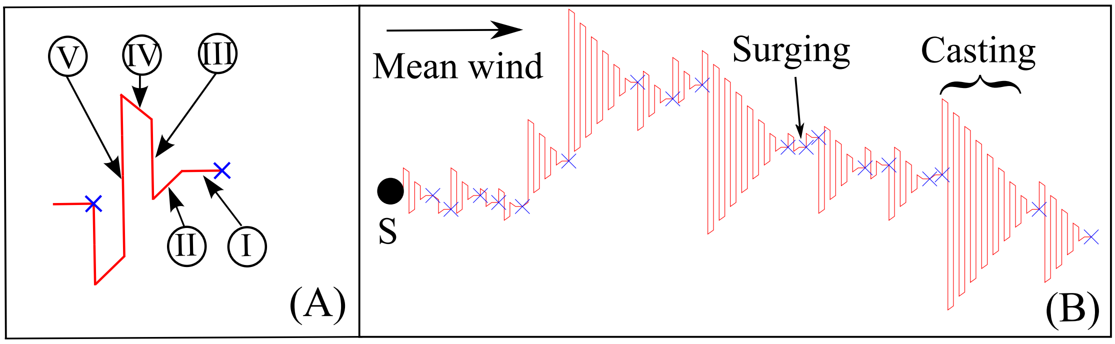


FIGURE F.1: (A) A short trajectory of an agent navigating according to the cast-and-surge algorithm with $\lambda \to 0$. (B) A complete sample trajectory. The black circle is the location of the source (S), while the blue $\times$ correspond to the points where it detected an odor particle within its olfactory range.

step I: If the agent has detected at least one odor particle in the time interval Δt, it moves upwind by $v_0 \Delta t$ units. v_0 being the speed of the agent. This phase is called 'surging'. The agent remains in such phase as long as it detects odor particles within every Δt time. After moving the agent sets $t' = 0$, a number that the agent keeps track of.
step II: In absence of any odors, the agent moves by $v_0 \Delta t$ units in a direction that forms an angle of $+45°$ with respect to the locally estimated upwind direction.
step III: The agent updates the t' as $t' \leftarrow t' + 2\Delta t$ and then moves in the crosswind direction for time t' with speed v_0.
step IV: The agent moves by $v_0 \Delta t$ units in the direction that forms an angle of $-45°$ with respect to the locally estimated upwind direction.
step V: The agent updates the t' as $t' \leftarrow t' + 2\Delta t$ and then moves with speed v_0 in the crosswind direction (opposite to the one taken in step III) for time t', and resumes further from step II.

The steps II-V describe the 'casting' phase. During this phase, if at any time the agent detects the odor, then it terminates the casting phase, sets $t' = 0$ and starts the surging phase (step I) from the next decision time.

In Fig F.1B, we plot a complete sample trajectory of the agent following the cast-and-surge algorithm described above. The ensuing trajectory displays the characteristic zig-zag pattern. Two observations are in order. First, the crosswind excursions increase linearly with time. Second, the length traveled in the upwind direction decreases as the inverse square root of time since the last detection. This reflects the fact that the upwind progression is discouraged in the absence of any cues. In the case presented in the main text (for which $\lambda = 1$) the estimate of the mean wind direction $\hat{\boldsymbol{u}}(t)$ computed by the agent changes with time, as $\lambda > 0$. Thus, in the turbulent environment, where the local wind direction fluctuates, the trajectory of the agent deviates from the one depicted in Fig. F.1B as can be see in Fig. 1C of the main text.

F.2 Description of the flow environment

In our simulations, the flow environment is given by an incompressible, two-dimensional velocity field, $\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{U} + \boldsymbol{v}(\boldsymbol{x}, t)$, with a constant mean $\boldsymbol{U}$, representing the mean wind and superimposed isotropic fluctuations, $\boldsymbol{v}(\boldsymbol{x}, t)$. Odor particles, representing patches of odor with concentration above the threshold value for being detected by the searching agents, are evolved as tracers according to the dynamics $\dot{\boldsymbol{x}} = \boldsymbol{u}(\boldsymbol{x}, t)$. In the following we discuss in detail the two models we considered for the fluctuating component of the velocity field.

F.2.1 Stochastic flow

As a first simplified setting, we model velocity fluctuations by considering a stochastic flow obtained by superimposing a few Fourier modes, each one of them having Gaussian amplitudes, whose real and imaginary part evolve according to independent Ornstein-Uhlenbeck (OU) processes with a specified correlation time τ_f. In this way the resulting flow is spatially smooth and exponentially correlated in time.

Specifically, we consider a flow characterized by a single scale L, obtained by superimposing 8 Fourier modes: $\boldsymbol{k} = (k_x, k_y) \in K = K_1 \cup K_2 = \{(k_s, 0), (0, k_s)\} \cup \{(k_s, \pm k_s)\}$, where $k_s = 2\pi/L$ (notice that we listed only four modes as the other fours are obtained from $\boldsymbol{k} \to -\boldsymbol{k}$, i.e. complex conjugation for maintaining the fields real). The fluctuating velocity is obtained as $\boldsymbol{v}(\boldsymbol{x},t) = \boldsymbol{\nabla}^{\perp}\psi(\boldsymbol{x},t)$ with $\boldsymbol{\nabla}^{\perp} = (-\partial_y, \partial_x)$, and the stream function ψ is computed at each odor particle position by means of the following formula:

$$\psi(\boldsymbol{x},t) = \sum_{\boldsymbol{k}\in K} (A(\boldsymbol{k},t)e^{i\boldsymbol{k}\cdot\boldsymbol{x}} + c.c.) \ , \tag{F.1}$$

where *c.c.* stands for the *complex conjugate*. The amplitudes of the Fourier modes $A(\boldsymbol{k},t)$ are Gaussian random complex variables evolving with the following OU process

$$\partial_t A_\gamma(\boldsymbol{k},t) = -\frac{1}{\tau_f}A_\gamma(\boldsymbol{k},t) + \left(\frac{2\sigma^2(\boldsymbol{k})}{\tau_f}\right)^{\frac{1}{2}}\eta_\gamma(\boldsymbol{k},t) \ , \tag{F.2}$$

where γ labels the real and imaginary part, $\eta_\gamma(\boldsymbol{k},t)$ are zero mean Gaussian variables with correlation $\langle\eta_\gamma(\boldsymbol{k},t)\eta_{\gamma'}\boldsymbol{k}',t')\rangle = \delta_{\gamma,\gamma'}\delta_{\boldsymbol{k},\boldsymbol{k}'}\delta(t-t')$ and so that

$$\langle A_\gamma(\boldsymbol{k},t)A_{\gamma'}(\boldsymbol{k}',t')\rangle = \sigma^2(\boldsymbol{k})\delta_{\gamma,\gamma'}\delta_{\boldsymbol{k},\boldsymbol{k}'}\exp(-|t-t'|/\tau_f) \ .$$

The standard deviations $\sigma(k)$ have been chosen to have an approximately isotropic velocity field with full control on the fluctuations intensity $u_{rms} = \sqrt{\langle(v_x^2+v_y^2)/2\rangle}$. In particular, we take $\sigma(k) = cu_{rms}/(\sqrt{3}k_s)$ with $c=1$ for $\boldsymbol{k}\in K_1$ and $c=1/2$ for $\boldsymbol{k}\in K_2$ so that $\langle v_x^2\rangle = \langle v_y^2\rangle = u_{rms}^2$.
Similar flows have been used for studying, e.g., the statistical dynamics of inertial particles (see, e.g. [99, 100]).
In our simulations, the constant mean wind is fixed to $U=1$, and the fluctuations intensity to $u_{rms} = 0.42U$. For what concerns the fluctuating component, it has one single characteristic scale set to $L=10$ and correlation time of the amplitudes of the Fourier modes equal to $\tau_f = 5$.

Tests about the search conducted by one single agent have been done considering different values of the flow parameters, also introducing more than one scale. Such tests have shown the same qualitative behaviors reported here, provided that u_{rms} remains smaller than U.

F.2.2 Turbulent flow

As said in the main text, in order to test the robustness of the collective odor search algorithm, we also considered the more realistic and more complex case in which the velocity fluctuations are obtained from a direct numerical simulation

(DNS) of the 2D Navier-Stokes equations (NSE) in the inverse cascade regime. In particular, we considered the NSE written for the vorticity field, $\omega = \boldsymbol{\nabla} \times \boldsymbol{u}$, reads

$$\partial_t \omega + \boldsymbol{v} \cdot \boldsymbol{\nabla} \omega = \nu \Delta \omega - \alpha \omega + f \,, \tag{F.3}$$

where $\boldsymbol{v} = \nabla^{\perp} \psi(\boldsymbol{x}, t)$ and the stream function is obtained by inverting $\omega = -\Delta \psi$. DNS of Eq. (F.3) have been carried out using a standard 2/3 dealiased pseudo-spectral solver over a bi-periodic $2\pi \times 2\pi$ box with 2^{nd} order Runge-Kutta time stepping. Energy and enstrophy are injected at rates ϵ and ζ, respectively by the forcing term f which is a zero mean, Gaussian field with correlation $\langle f(\boldsymbol{x}, t) f(\boldsymbol{0}, t') \rangle = \delta(t - t') F(r/\ell_f)$ acting at small scales, $\ell_f \ll 2\pi$, with $F(x) = F_0 \ell_f^2 \exp(-x^2/2)$. With this forcing, an inverse energy cascade sets in at scales $r \gg \ell_f$. In order to establish a statistically steady state the Ekman friction term, $-\alpha\omega$, extracts energy at the large scales, $L_\alpha \approx \epsilon^{1/2} \alpha^{-3/2}$, while the viscous term removes enstrophy at small scales. As a result, we have a velocity field which is non-smooth in the inertial range of scales, $\ell_f \ll r \ll L_\alpha$, and smooth below ℓ_f. In Fig. F.2 we show the mean energy spectrum, $E(k)$, displaying the Kolmogorov, $k^{-5/3}$, scaling behavior, which means that in the inertial range velocity differences over a scale r are approximately Hölder continuous with exponent 1/3.

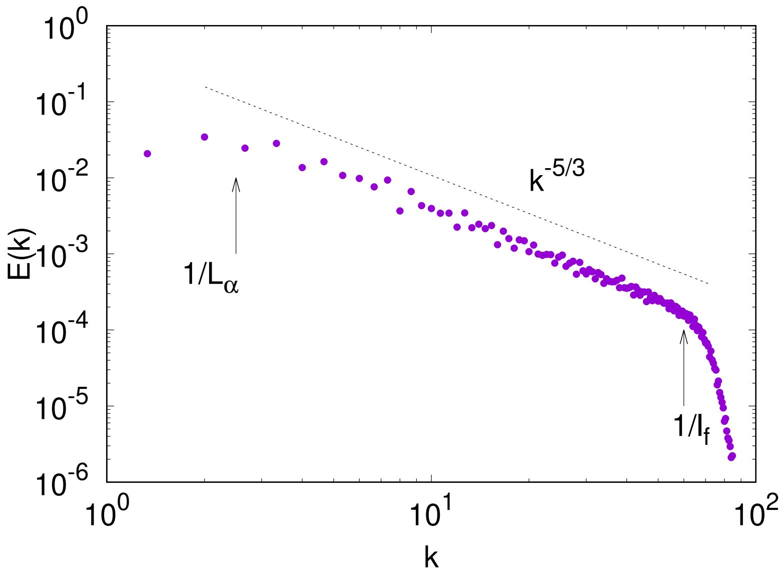


FIGURE F.2: Eneergy spectrum obtained by direct numerical simulations of Eq. (F.3) with 256^2 grid points. Hyperviscous dissipation of order 8 has been used with viscosity $\nu_8 = 1.3\,10^{-29}$, Ekman friction coefficient $\alpha = 0.02$ and time step $dt = 10^{-3}$. The large scale of the flow is about half of the simulation box.

Owing to the necessity to store the entire history of the full velocity field (see below for details), we used a relatively small resolution of 256^2 grid points. Thus to reduce as much as possible the enstrophy cascade range we used an hyperviscous term of order 8 which remove enstrophy very close to the injection scale, this is a customary procedure when interested in simulating the inverse cascade in low resolution DNS (see Refs. [95, 96]).

In order to evolve the odor particles and perform statistics over many episodes of the collective search, we stored the whole evolution of the velocity field for about 10 large-scale time scales, $T_{L_\alpha} \approx 5$. The velocity field history is then cycled in time, so the flow is effectively periodic in time with a period of about $10T_{L_\alpha}$. For each episode we place the source in a different position within the simulation box and define the mean wind direction to be either along the horizontal or vertical direction (this is done to average over different flow regions). We let the source emit the particles at exponentially distributed times with average $\tau = 5$, which corresponds to the time scale associated to the forcing scale, and advect them in the full plane (making use of the spatial periodicity), with a velocity obtained by interpolating the velocity field at the particle position and superimposing the mean wind U. We wait until the statistics of the odor particles becomes stationary in the region of interest and then let the searching agents look for the source.

Then the agents are initially placed at distance L_x downwind from the source (see Fig.1A of main text) and wait for the first detection to start the search. The episode ends when one of the agents reaches the source as described in main text.

F.3 Table of parameters

Description	**Symbol**	**Numerical Value**
Initial distance between the source and the center of mass of the agents	L_x	$250R_d$
Simulation box size factor	b	2.5
Number of agents	N	100
Emission rate of odors from the source	J	1.0 particle/Δt
Initial cluster size of agents	R_b	$25R_d$
Range of agent-agent interaction	R_a	$5.0R_d$
Speed of the agents	v_0	$2.5R_d/\Delta t$
Strength of the noise	η	0.1
Inverse of the memory time	λ	$1.0/\Delta t$
Mean wind intensity	U	1.0

Description	**Stochastic Flow**	**Turbulent flow**
Decision time Δt	1.0	0.2
Olfactory range of the agent R_d	0.2	0.04
Fluctuations intensity u_{rms}	0.42U	0.42U
Characteristic length	10.0	2.0
Characteristic time	5.0	5.0

TABLE F.1: Top table shows the values used for the parameters. Apart from the dimensionless quantities, they are written in terms of the radius of detection of one agent R_d and its decision time Δt. Bottom table shows the values used for the latter quantities in each flow configuration as well as the flow parameters. It is worth pointing out that the difference between the first two numerical values comes from the fact that the two flows implemented in our simulations have different characteristic length and time scales. Therefore, in order to study the olfactory search in comparable regimes, we had to rescale all the quantities accordingly, maintaining at the same time identical ratios among them.

Appendix G

Simulation codes

For both studies presented in Chapter 2 and 3, we developed in-house simulation codes written in Fortran. In this appendix, we provide codes that we developed to carry out simulations.

G.1 Code : Simulation of multi-agent reinforcement learning

Mihir Durve and Fernando Peruani

```
! Input parameters are imported from input.in file.
! Sample of input file "input.in" :

!seed    n    tstart   tend     rho  v0   r    eta  phi
!10551   200  0        10000    2.0  5.0  1.0  0.0  1.0
!nconfig      tskip    lbird    Read_flag    Action_max
!1000         500  1   0        7


! Seed : Seed for random number generator
! n : number of teacher agents
! tstart : Start time of simulation (typically = 0)
! tend : Number of time steps in single episode
! rho : Density of agents (teachers + learners)
! v0 : Speed of the agents
! r : Agent-agent interaction radius
```

```
! eta : Noise strength (range : 0 to 1)
! phi : View-angle of agents (range 0 to 1)
! nconfig : Number of training episodes
! tskip : Number of time steps to discard before system reach steady state
! lbird : Number of learners (l stands for leraner)
! Read_flag : Flag to enable policy evaluation, by reading Q-values from another
!           input file
! Action_max : Number of allowed actions

! Flow of the code :

! Initialisation of parameters, variables
! Read input
! Calculate required parameters, initialize required parameters

! DO loop : from episode=1, to nconfig

  ! Do loop : from time=tstart to tend

    ! Do loop : from agent 1, nr
      ! Compute state of an agent
      ! Compute action of the agent
      ! Execute action
      ! update position and velocity of agent, with PBCs
      ! Compute reward based on current and previous neighbors
      ! For learners, update their Q-matrix
    ! Do loop for agents end

  ! Compute and save quantities of interest
  ! Do loop for time end

! Compute and save quantities of interes
! Do loop for episodes end

! Saving data files. Stop.


program learn

implicit none
 character(len=100) :: fn,fn1,fn2,fn3,fn4,fn5

real*8, parameter :: zero = 0.0_8, half = 0.5_8, one = 1.0_8
real*8, parameter :: pi = 2.0_8*asin(one)
```

```

integer*8 :: n,ln,tstart,tend,tskip,state_max=32,action_max, nconfig,p
integer*8 :: ltheta_old_int,index_Q1,lnbi_temp,a_pointer,s_pointer,nr
integer*8 :: sd(1), nbi,state_ln,horizon,E,T,rnd_int,d1,d,flag_md=0

integer*8, allocatable, dimension(:,:) :: lnbi
integer*8, allocatable, dimension(:) :: s,a,s_record,a_record,same,choose


real*8 :: box, v0, phi, cosphi, halfbox, r, twopi,r2,rnd_gauss,eps=0.09
real*8 :: rho, eta, psi, rnd, delTheta, gamma_b=1.0,gamma_lb=1.0,psi_teacher
real*8 :: beta,reward_temp,eps_old,eps_fix,alpha_fix
real*8 :: t1,t2,sum_op,sine_ij,theta_dot,mod_new,theta_mean
real*8 :: dt=0.1_8,eta1=0.3,alpha=0.02,prob=0.0
real*8 :: meanVx,meanVy,reward_theta,angle,theta
real*8 :: cosdth,sindth,newvx,newvy,vx,vy,sumvx
real*8 :: sumvy,theta_max,del_theta,temp_angle


real*8, allocatable, dimension(:) :: x, y, x_new, y_new, theta_old, theta_new
real*8, allocatable, dimension(:) :: lx, ly, lx_new, ly_new,
ltheta_new,ltheta_old
real*8, allocatable,dimension(:) :: mean_angle,vx_theta,vy_theta,psi_avg
real*8, allocatable,dimension(:) :: avg_reward,psi_avg_teacher
real*8, allocatable, dimension(:,:) :: reward

real*8, allocatable,dimension(:,:,:) :: Q,Q_old
Integer*8, allocatable,dimension(:,:,:) :: n_sa

integer*8 :: i,j,k,diff_int,max_temp,temp_int,read_flag,a_noise
real*8 :: diff,dist,xji,yji,temp,z0,z1,u1,u2,reward_factor=1.0,dummy
logical :: testx, testy, testxy

open(unit=1, file='input.in', status='old', action='read')

call cpu_time(t1)

read(unit = 1, fmt = *)
read(unit = 1, fmt = *) sd(1), n, tstart, tend, rho, v0, r, eta, phi
read(unit = 1, fmt = *)
read(unit = 1, fmt = *) nconfig, tskip, ln,read_flag,action_max
 close(unit=1)

nr = n + ln ! #Total agents = #Learning + #teachers
```

```
alpha_fix = alpha
eps_fix = eps


if(n>0) Allocate (x(n), y(n), x_new(n), y_new(n), theta_old(n), theta_new(n) )
Allocate (lx(nr), ly(nr), lx_new(nr), ly_new(nr), ltheta_old(nr))
allocate (ltheta_new(nr),mean_angle(nr))
Allocate (Q(nr,0:state_max,0:action_max),n_sa(nr,0:state_max,0:action_max))
allocate (lnbi(0:tend+1,nr),reward(nr,0:tend))
allocate(s(0:state_max+1),a(0:action_max+1))
allocate(s_record(nr),a_record(nr),avg_reward(nr),choose(0:action_max-2))
allocate(vx_theta(nr),vy_theta(nr),psi_avg(0:tend))
allocate(same(0:action_max),psi_avg_teacher(0:tend))

open(unit=3,file='spp.stat', status='replace', action='write')

open(unit=5,file='formovie.txt', status='replace', action='write')
open(unit=55,file='lnbi.txt', status='replace', action='write')
open(unit=555,file='lformovie.txt', status='replace', action='write')
open(unit=11,file='lstate.txt', status='replace', action='write')
open(unit=12,file='Q.txt', status='replace', action='write')

open(unit=14,file='Q_final.txt', status='replace', action='write')
open(unit=16,file='Q_initial.txt', status='old', action='read')
open(unit=17,file='op_avg.txt', status='replace', action='write')

open(unit=21,file='SA.txt', status='replace', action='write')
open(unit=22,file='Plot.gnu', status='replace', action='write')
open(unit=23,file='Reward_Episode.txt', status='replace', action='write')


open(unit=24,file='Orientation.txt', status='replace', action='write')
open(unit=200,file='OP_progress.txt', status='replace', action='write')
open(unit=201,file='OP_progress_teacher.txt', status='replace', action='write')

box = sqrt(dble(n+ln)/rho)

write(unit = 3, fmt = *) "Input Values******************************************"
write(unit = 3, fmt = *) "seed: ", sd(1)
write(unit = 3, fmt = *) "n: ", n
write(unit = 3, fmt = *) "tstart: ", tstart
write(unit = 3, fmt = *) "tend: ", tend
write(unit = 3, fmt = *) "rho: ", rho
write(unit = 3, fmt = *) "v0: ", v0
```

```
write(unit = 3, fmt = *) "r: ", r
write(unit = 3, fmt = *) "eta: ", eta
write(unit = 3, fmt = *) "phi: ", phi
write(unit = 3, fmt = *) "nconfig :", nconfig
write(unit = 3, fmt = *) "tskip :", tskip
write(unit = 3, fmt = *) "Learning birds : ", ln
write(unit = 3, fmt = *) "Read_flag : ", read_flag
write(unit = 3, fmt = *) "box: ", box
write(unit = 3, fmt = *) "*****************************************************"

call random_seed(put = sd)

twopi = 2.0_8*pi

halfbox = half*box
r2 = r**2
psi_avg = 0.0_8
psi_avg_teacher = 0.0_8
eps_old = eps

if(action_max>1) then
  del_theta = 2.0*pi/real(state_max)
else
  del_theta = 0 ; theta_max=0.0
endif

theta_max = del_theta*(0.5)*(real(action_max-1))
theta_max = abs(theta_max)

print*,theta_max*(180.0/3.1415),del_theta*(180.0/3.1415)

if(ln>9999) then
  print*," More than 9999 Birds. Need to modify file units. STOPPING"
  stop
endif

do i=1,ln
  ! build filename -- i.dat
  write(fn2,fmt='(i0,a)'), i, 'RE.txt'

  ! open it with a fixed unit number
  open(unit=20000+i,file=fn2, form='formatted',status='replace',action='write')

enddo
```

```
print*,"Simulation Start."

n_sa =0

open(unit=101, file='current.txt', status='unknown', action='write')


do E=0,action_max-1

  if(mod((action_max),2)==0) then
    beta = -theta_max + E*del_theta - (del_theta/2.0)
  else
    beta = -theta_max + E*del_theta
  endif
  print*,E,beta*(180.0/pi)
enddo

eps_fix=eps

open(unit=876,file='op_1.txt', status='replace', action='write')

do E=1,nconfig  !nconfig is max number of episodes.

avg_reward = 0.0_8
psi_avg = 0.0_8  !Attention, Psi is avg in all episodes.
psi_avg_teacher = 0.0_8
open(unit=18,file='op.txt', status='replace', action='write')

  do i=1, nr  ! Loading random position and velocity for learning bird.

    call random_number(rnd)
    lx(i) = box*rnd
    call random_number(rnd)
    ly(i) = box*rnd
    call random_number(rnd)
    ltheta_old(i) = (rnd-half)*twopi
  end do
  ! Loading done************************************************************
```

```
!Loading Q-value for first learning episodes****************************
if(E==1) then
  if(n>0) then
    do i=ln+1,ln+1
      do j=0,state_max-1
        do k=0,action_max-1
          read(16,*),dummy,dummy,dummy,Q(i,j,k)
        enddo
      enddo
      enddo

      if(n>1) then
      do i=ln+2,nr
        do j=0,state_max-1
          do k=0,action_max-1
            Q(i,j,k) = Q(ln+1,j,k)
          enddo
        enddo
      enddo
      endif

    do i=1,ln
      do j=0,state_max-1
        do k=0,action_max-1
          Q(i,j,k) = reward_factor
        enddo
      enddo
    enddo
  else
    do i=1,ln
      do j=0,state_max-1
        do k=0,action_max-1
          Q(i,j,k) = reward_factor
          !if(i==bird_index) write(*,887) i,j,k,Q(i,j,k)
        enddo
      enddo
    enddo

  endif
endif
! Loading Qvalue done*******************************************************


!******************* Saving the initial configuration*****************
```

```
888 Format(4F10.5)
if(E==nconfig) then

  do i=1,ln        ! Saving data for movie
  write(555,888) lx(i),ly(i),v0*cos(ltheta_old(i)),v0*sin(ltheta_old(i))
  enddo

  do i=ln+1,nr        ! Saving data for movie
  write(5,888) lx(i),ly(i),v0*cos(ltheta_old(i)),v0*sin(ltheta_old(i))
  enddo


endif
!**********************************************************************

do T=1,tend

  ! Computing order parameters for learnes and teachers.
  sumvx = 0.0 ; sumvy=0.0
  do j=1,ln
  vx_theta(j) = cos(ltheta_old(j)) ; vy_theta(j) = sin(ltheta_old(j))
  sumvx = sumvx + (vx_theta(j)); sumvy = sumvy + (vy_theta(j))
  enddo
  psi = sqrt(sumvx**2 + sumvy**2)/(v0*dble(ln))
  psi_avg(T) = psi_avg(T) + psi

  sumvx = 0.0 ; sumvy=0.0
  if(n>0) then
    do j=ln+1,nr
      vx_theta(j) = v0*cos(ltheta_old(j))
  vy_theta(j) = v0*sin(ltheta_old(j))
      sumvx = sumvx + (vx_theta(j)); sumvy = sumvy + (vy_theta(j))
    enddo

    psi_teacher = sqrt(sumvx**2 + sumvy**2)/(v0*dble(nr-ln))
    psi_avg_teacher(T) = psi_avg_teacher(T) + psi_teacher
    if(E==1) write(876,*) t,psi_teacher
  endif
  ! Order parameters computed.

  if(T>=0) then

  do i=1,nr ! For Learning bird

```

```
    ! Calculating state in which the bird i is ********************************
    ! For state, first we compute average direction of neighbors of the bird i
      meanvx = 0.0 ; meanvy =0.0

      lnbi(T,i) = 0

      do j = 1, nr

        xji = lx(j) - lx(i)
        yji = ly(j) - ly(i)

        !Apply minimum image separation condition
        if(xji > halfbox) then
          xji = xji - box
        else if(xji <= -halfbox) then
          xji = xji + box
        end if
        if(yji > halfbox) then
          yji = yji - box
        else if(yji <= -halfbox) then
          yji = yji + box
        end if

        !Check neighbourhood
        dist = sqrt(xji**2 + yji**2)
        if(dist > r) cycle

        lnbi(T,i) = lnbi(T,i) + 1

        meanvx = meanvx + cos(ltheta_old(j)-ltheta_old(i))
        meanvy = meanvy + sin(ltheta_old(j)-ltheta_old(i))
      end do

      if(lnbi(T,i)>0) then
    ! Calculating mean average direction of neighbours
        mean_angle(i) = atan2(meanvy,meanvx)
        temp_angle = mean_angle(i)
      endif
    ! Average direction of neighbors is calculated


    ! Calculating state label of bird i using average direction of neighbors

      if(lnbi(T,i)>0) then
```

```

        theta = -pi + pi/real(state_max)

        do j=0,state_max
          if(mean_angle(i) < (theta)) then
            if(j<=state_max-1) then
              s_pointer =j
              exit
            else
              s_pointer = 0
              exit
            endif
          endif

          if(j<=state_max-2) then
            theta = theta + 2.0*pi/real(state_max)
            else
            theta = theta + pi/real(state_max)
          endif

        enddo

      s_record(i) = s_pointer   ! This is a state label of the bird
      endif

      if(lnbi(T,i)>0) then

        ! Select best action for this state with Epsilon greedy****************

        call random_number(rnd)

        if(i>ln) then
          eps=0.0
        else
          eps=eps_fix
        endif

    ! Check if arg max Q is to be executed
    ! or any random action is to be executed

        if(rnd>eps) then
    ! Calculating arg max Q(s,a')

        k=0
```

```
            temp = Q(i,s_record(i),0) ; max_temp = 0
            do j=1,action_max-1

              if (Q(i,s_record(i),j)>temp) then
              temp = Q(i,s_record(i),j)
              max_temp = j

              endif

            enddo
            k=0 ; same(1)= max_temp
            do j=0,action_max-1
              if (Q(i,s_record(i),j)==temp)then
                k=k+1
                same(k) = j
        ! Actions with same Q values are stored in this array
              endif
            enddo

           if(k>1) then
         ! If there is a degeneracy then execute random action amonst them
         call random_number(rnd)
             rnd_int = rnd*real(k)  ! Action is randomly chosen from this array.
             max_temp = same(rnd_int+1)
           endif

        else
      ! Executing a random action
          call random_number(rnd)
          rnd_int = rnd*real(action_max)
          max_temp = rnd_int

        endif

        a_record(i) = max_temp ! Action label to be executed

    ! Computing and executing turn with the chosen action label
        if(mod((action_max),2)==0) then
        beta = -theta_max + a_record(i)*del_theta - (del_theta/2.0)
        else
        beta = -theta_max + a_record(i)*del_theta
        endif
```

```
          ltheta_new(i) = beta + ltheta_old(i)

          n_sa(i,s_record(i),a_record(i)) = n_sa(i,s_record(i),a_record(i)) + 1

        else

     ! For teachers, their heading direction is modified by noise
          call random_number(u1)
          call random_number(u2)

          z0 = sqrt(-2.0*log(u1))*cos(2.0*twopi*u2)
          z1 = sqrt(-2.0*log(u1))*sin(2.0*twopi*u2)

          rnd_gauss = z0

          ltheta_new(i) = ltheta_old(i) + (eta1*sqrt(dt)*rnd_gauss)

        endif
      enddo ! birds loop ended
      endif ! IF (T >Tskip)

      if(mod(T,100)==0) then
        write(*,900),T,"Of",tend,E,"of",nconfig
        write(101,*) T,"Of",tend,E,"of",nconfig

      endif

      900 Format (I10,A6,I10,I6,A6,I6)

      ! Overriding the previous values till tskip
      if(t<=tskip) then
        do i=1,ln
          call random_number(u1)
          call random_number(u2)

          z0 = sqrt(-2.0*log(u1))*cos(2.0*twopi*u2)
          z1 = sqrt(-2.0*log(u1))*sin(2.0*twopi*u2)

          rnd_gauss = z0

          ltheta_new(i) = ltheta_old(i) + (eta1*sqrt(dt)*rnd_gauss)
        enddo
      endif
```

```

!updating agents positions

do i=1,nr
  lx(i) = lx(i) + v0*cos(ltheta_new(i))*dt     ! Updating postions of
  ly(i) = ly(i) + v0*sin(ltheta_new(i))*dt     ! Learning birds
enddo


!! Applying PBC for birds
if(T>=0) then
do i = 1, nr
  if(lx(i) < zero) then
    lx(i) = lx(i) + box
  else if(lx(i) > box) then
    lx(i) = lx(i) - box
  end if

  if(ly(i) < zero) then
    ly(i) = ly(i) + box
  else if(ly(i) > box) then
    ly(i) = ly(i) - box
  end if

  testx = (lx(i) < zero).or.(lx(i) > box)
  testy = (ly(i) < zero).or.(ly(i) > box)
  testxy = testx.or.testy
  if(testxy) then
    write(unit=3, fmt = *) "Learning particle outside the box; stopping."
  stop
  end if

end do
endif
!! Applying PBC for birds done.

  do i=1,ln

  lnbi(T+1,i) = 0


  do j = 1, nr

    xji = lx(j) - lx(i)
```

```
        yji = ly(j) - ly(i)

        !Apply minimum image separation condition
        if(xji > halfbox) then
          xji = xji - box
        else if(xji <= -halfbox) then
          xji = xji + box
        end if
        if(yji > halfbox) then
          yji = yji - box
        else if(yji <= -halfbox) then
          yji = yji + box
        end if

        !Check neighbourhood
        dist = sqrt(xji**2 + yji**2)
        if(dist > r) cycle

        lnbi(T+1,i) = lnbi(T+1,i) + 1


      end do


      ! Switching on Q-learning after the flock has reached steady state
      if(T>tskip) then

          if(lnbi(T,i)>1) then

          diff_int = lnbi(T+1,i) - lnbi(T,i)
          s_pointer = s_record(i) ; a_pointer = a_record(i)

          if(i>ln) then
            alpha=0.0

          else
            alpha=alpha_fix
        ! Updating Q-matrices of learners
            if(diff_int >=0) then
            reward(i,T) = 1.0*reward_factor
            Q(i,s_pointer,a_pointer) = (1.0-alpha)*Q(i,s_pointer,a_pointer) +
(alpha)*reward(i,T)
            else
            reward(i,T) = -1.0*reward_factor
```

```
              Q(i,s_pointer,a_pointer) = (1.0-alpha)*Q(i,s_pointer,a_pointer)+
(alpha)*reward(i,T)
              endif
            endif
            endif
          avg_reward(i) = avg_reward(i) + (reward(i,T))

        endif
        enddo


        ltheta_old = ltheta_new  ! Updating theta for learning birds

      if(E==nconfig) then

      write(55,*) T,lnbi(T,1)
      if(t>tskip) then
        if(mod(T,1)==0) then
          do i=1,ln        ! Saving data for movie
          write(555,888) lx(i),ly(i),v0*cos(ltheta_old(i)),v0*sin(ltheta_old(i))
          enddo

          do i=ln+1,nr        ! Saving data for movie
          write(5,888) lx(i),ly(i),v0*cos(ltheta_old(i)),v0*sin(ltheta_old(i))
          enddo


        endif
      endif
      endif


    enddo !T=1,tend loop

psi=0.0_8
psi_teacher=0.0_8
do i=tskip+1,tend
  psi = psi + psi_avg(i)
  psi_teacher = psi_teacher + psi_avg_teacher(i)
enddo


avg_reward = avg_reward /(real(tend-tskip))

```

```

do i=1,ln
  write(20000+i,*) E,(1.0-(avg_reward(i)/reward_factor))/2.0,
(avg_reward(i)/reward_factor)
enddo


write(200,*)E,psi/(real(tend-tskip))
write(201,*)E,psi_teacher/(real(tend-tskip))



do i=1,ln
        ! build filename -- i.dat
  write(fn1,fmt='(i0,a)') i, 'Q.txt'

  ! open it with a fixed unit number
  open(unit=10000+i,file=fn1, form='formatted',status='replace',action='write')

enddo

do i=1,ln
  do j=0,state_max-1

    do k=0,action_max-1
      write(10000+i,887) i,j,k,Q(i,j,k)/reward_factor
    enddo
  write(10000+i,*) ""
  enddo
enddo

do i=1,ln
 close(unit=i+10000)
enddo


do i=1,ln
        ! build filename -- i.dat
  write(fn4,fmt='(i0,a)') i, 'SA.txt'

  ! open it with a fixed unit number
  open(unit=40000+i,file=fn4, form='formatted',status='replace',action='write')

enddo
```

```
do i=1,ln
  do j=0,state_max-1

    do k=0,action_max-1
      write(40000+i,865) j,k,n_sa(i,j,k)
    enddo
   write(40000+i,*) ""
  enddo
enddo

do i=1,ln
 close(unit=i+40000)
enddo


865 Format(2I10,I20)


do i=1,ln
        ! build filename -- i.dat
   write(fn3,fmt='(i0,a)') i, 'maximum_Q.txt'

        ! open it with a fixed unit number
        open(unit=30000+i,file=fn3,status='replace',action='write')

enddo

do i=1,ln
  do j=0,state_max-1
    temp = Q(i,j,0) ; temp_int=0
    do k=0,action_max-1
      if(Q(i,j,k)>temp) then
        temp=Q(i,j,k)
        temp_int = k
      endif
    enddo
   write(30000+i,*)j, temp_int
  enddo
enddo

 close(unit=18)

do i=1,ln
```

```fortran
   close(unit=10000+i)
   close(unit=30000+i)
enddo

vx_theta = cos(ltheta_old) ; vy_theta = sin(ltheta_old)
sumvx = sum(vx_theta); sumvy = sum(vy_theta)
write(24,*) E, atan2(sumvy/real(ln), sumvx/real(ln))

enddo !E=1,nconfig loop **************************************************

 close(unit=24)

887 Format(3I10,F10.5)
883 format(2I6,I12)


psi_avg = psi_avg !/(real(nconfig))

psi = 0.0_8

do i=tskip,tend
   psi = psi + psi_avg(i)
enddo

write(17,*) psi/(real(tend-tskip))



do i=1,ln
 close(unit=10000+i)
 close(unit=20000+i)
 close(unit=30000+i)
enddo

call cpu_time(t2)
write(unit = 3, fmt = *) "Computation Time =",(t2-t1)
write(unit = 3, fmt = *) "Job completed"

 close (unit=1)
 close (unit=5)
 close (unit=555)
 close (unit=55)
 close (unit=11)
 close (unit=3)
```

```
 close (unit=14)
 close (unit=15)
 close (unit=16)
 close (unit=17)
 close (unit=200)

 close (unit=21)
print*,"Simulation End."

STOP
END PROGRAM

!******************************************************************************
```

G.2 Code : Multi-agent olfactory serach

Lorenzo Piro and Mihir Durve

G.2.1 Main code

```
! Ref BC + Noise in VM + stop at 1st arrival.
! Seems fine
! 28 June 2019
! Odor particles diffusing with diffusion constant 'ed'
! Define ed, initial_r , mean_wind, d_t
! Odor particle are not resetted once detected by an agent.


program aroma
use flow_mod
use inout
implicit none

 character(len=50) :: fn1

integer :: i,j,k,tstart,tend,t,n,detect_flag_temp=0,reflect,is
integer :: lorenzo,inverse_dt,int_dt,fails,piro
integer :: detected =0,odd_count=0,global_detect_flag_temp=0
```

```
integer :: global_detect_flag=0,detect_time=1
integer  :: sd(1),nconfig,E,E_count=1,reach_count=0,total_reach_count=0
integer :: first_reach_flag=0,first_detect_time
integer,allocatable,dimension(:) :: detect,reach_flag,agent_count,detect_flag
integer,allocatable,dimension(:) :: clock,turn_time,sign_flag,odor_flag
integer,allocatable,dimension(:,:) :: num_detect
real*8 :: pxy,D,pi=3.1415,half=0.5_8,rnd,theta_i,beta,two=2.0
real*8 :: pL,pR,theta,Lx,rho,ra,rd,v0,vo,t_real
real*8,allocatable,dimension(:) :: x,y,vx,vy,x_odor,y_odor,vx_odor
real*8,allocatable,dimension(:) :: vy_odor,vx_old,vy_old
real*8 :: vx_temp_j,vy_temp_j,dist,vx_temp_i,vy_temp_i
real*8 :: p_detect, norm,y_temp,vx_odor_avg,vy_odor_avg
real*8 :: vx_cs,vy_cs,vx_vicsek,vy_vicsek,cpu1,cpu2
real*8 :: x_low,x_max,y_low,y_max,rb
real*8 :: delTheta,cosdth,sindth,temp_vx,temp_vy,eta,box_factor
real*8 :: ed=0.2,d_t=0.010,u1,u2,z0,initial_r,d_t_v=1.0
real*8 :: vx_mean_wind=1.0,vy_mean_wind=0.0,inital_r
real*8 :: gauss1,gauss2,r1,r2,box_size,lambda,flow_rate2

real*8 :: flow_rate=5 !emission rate is inverse of flowrate.

real*8,allocatable,dimension(:) :: vx_last,vy_last,vx_last_firm,vy_last_firm
real*8,allocatable,dimension(:) :: vy_current,vx_current,theta_estimate
real*8,allocatable,dimension(:) :: x_new,y_new,vx_new,vy_new,vx_reflect
real*8,allocatable,dimension(:) :: vy_reflect,vy_estimate,vx_estimate
real*8,allocatable,dimension(:,:) :: x_dump,y_dump

integer :: winner_index,winner_detection,rb_cnt
real*8 :: average_detection,psi,psi_avg,sumvx,sumvy,omega_avg,omega,rg



call cpu_time(cpu1)
open(unit=1,file='input.in',status='old',action='read')
open(unit=2,file='data.txt',status='replace',action='write')
open(unit=4,file='odor.txt',status='replace',action='write')
open(unit=5,file='odor_count.txt',status='replace',action='write')
open(unit=6,file='for_time_distribution.txt',status='replace',action='write')
open(unit=7,file='reach_count.txt',status='replace',action='write')
open(unit=8,file='reach_time.txt',status='replace',action='write')

open(unit=10,file='run_stat.txt',status='replace',action='write')
open(unit=12,file='formovie.txt',status='replace',action='write')
open(unit=14,file='NP_t.txt',status='replace',action='write')
```

```
open(unit=15,file='Position_detected_odor.txt',status='replace',action='write')
open(unit=16,file='Estimate.txt',status='replace',action='write')
open(unit=17,file='Detections.txt',status='replace',action='write')
open(unit=18,file='Less_than_Rb.txt',status='replace',action='write')
open(unit=19,file='Vicsek_order_parameter.txt',status='replace',action='write')
open(unit=20,file='Massimo_order_parameter.txt',status='replace',action='write')
open(unit=21,file='Distance_square.txt',status='replace',action='write')
read(1,*)
read(1,*) sd(1),n,tstart,tend,Lx,ra,rd
read(1,*)
read(1,*) nconfig,beta,eta,v0,box_factor,lambda

Lx = Lx*rd


x_low=(-box_factor*Lx) ; x_max=(box_factor*Lx)
y_low=(-box_factor*Lx) ; y_max=(box_factor*Lx)

box_size = box_factor*Lx

NP = 2.0*box_size/(vx_mean_wind*flow_rate)

print*,"NP =",NP

call input()

allocate(x(n),y(n),vx(n),vy(n),detect(n),x_odor(NP),y_odor(NP),sign_flag(n))
allocate(vx_odor(0:tend+1),vy_odor(0:tend+1),vx_old(n),vy_old(n))
allocate(reach_flag(n),clock(n),turn_time(n),vx_last(n))
allocate(vy_last(n),agent_count(0:tend))
allocate(x_new(n),y_new(n),vx_new(n),vy_new(n),detect_flag(n))
allocate(vx_last_firm(n),vy_last_firm(n))
allocate(vx_reflect(n),vy_reflect(n),num_detect(1:n,1:nconfig))
allocate(odor_flag(NP),vy_current(n),vx_current(n)
allocate(vx_estimate(n),vy_estimate(n),theta_estimate(n))
allocate(x_dump(n,0:tend),y_dump(n,0:tend))

fails=0
vy_current=0.0_8
vx_current=0.0_8

vx_estimate=0.0
vy_estimate=0.0
```

```

call random_seed(put = sd)

do i=1,5000
call random_number(rnd)
enddo

!**********************************************************Lorenzo
     allocate(Ak_im(NS,NK),Ak_re(NS,NK),kx(NS,NK),ky(NS,NK),sig(NS,NK))

     !Initialise modes and amplitudes
     call initfluid_1(uf,L)

     do is = 1,NS
        do k = 1,NK
           call random_number(r1)
           call random_number(r2)
           gauss1=sqrt(-2.d0*log(r1))*cos(2.d0*pi*r2)
           gauss2=sqrt(-2.d0*log(r1))*sin(2.d0*pi*r2)
           Ak_im(is,k) = sig(is,k)*gauss1
           Ak_re(is,k) = sig(is,k)*gauss2
        end do
     end do
!*****************************************************************




inverse_dt = 1.0/d_t
print*,"inverse dt=",inverse_dt

do i=1,n
        ! build filename -- i.dat
  write(fn1,fmt='(i0,a)') i, 'data.txt'

        ! open it with a fixed unit number
        open(unit=i+100,file=fn1,status='replace',action='write')
enddo

!odd_count=0
num_detect=0.0


odor_flag=0
```

```fortran

do i=1,NP
  call random_number(u1)
  call random_number(u2)
  x_odor(i)=0.2*u1-0.1 ; y_odor(i)=0.2*u2-0.1
enddo

! ************************************************************************
do i=1,60000

  if(mod(i,int(flow_rate*inverse_dt))==0) then
  ! every 'flow_rate' time steps we generate a odor particle
    !odd_count = odd_count + 1  ! count of odor particles at time t
    do lorenzo=1,NP
      if(odor_flag(lorenzo)==0) then
        odor_flag(lorenzo)=1
        exit
      endif
    enddo
  endif

  t_real = real(i)
  Call rk2_1(t_real,x_odor,y_odor,d_t,NP,Ak_re,Ak_im,tau_f,box_size,odor_flag)

enddo



!*************************************************************************


ra = ra*rd
rb = (25.0)*rd
v0 = v0*rd
vo =v0
Do E=1, nconfig

print*,"Episode =",E

global_detect_flag=0
global_detect_flag_temp=0
reach_count=0

detect = 0
```

```
turn_time=2
!x_odor  =0.0
!y_odor=0.0
 clock=0
reach_flag=0
detect_flag=0
sign_flag=1
first_reach_flag=0
first_detect_time=0
psi_avg = 0.0_8
!vo=1.1*v0
omega_avg = 0.0_8

vy_current=0.0_8
vx_current=0.0_8

vx_estimate=1.0
vy_estimate=0.0


!******* initial condition agents***********
do i=1,n
   call random_number(rnd)
   rho=rnd*rb
   call random_number(rnd)
            rnd=two*(rnd-half)*pi
            x(i) = Lx + rho*cos(rnd)
            y(i) = rho*sin(rnd)

            call random_number(rnd)
            rnd = two*(rnd-half)*pi
            vx(i) = cos(rnd) ; vy(i)=sin(rnd)

!print*,i,vx(i),vy(i)
enddo

      !do i=1,n
      !   write(12,999) x(i),y(i),vx(i),vy(i)
      !enddo
!*****************************************


  do t=tstart,tend

```

```
  if(mod(t,1000)==0) print*,"Episode =",E,"Time =",t


    if(flow_rate>=1.0) then

      if(mod(t,int(flow_rate))==0) then
    ! every 'flow_rate' time steps we generate a odor particle
        !odd_count = odd_count + 1  ! count of odor particles at time t
        do lorenzo=1,NP
          if(odor_flag(lorenzo)==0) then
            odor_flag(lorenzo)=1
            exit
          endif
        enddo

      endif

    endif


    flow_rate2 = flow_rate - int(flow_rate)

    !write(5,*) t,odd_count  ! Writing number of odor particles with time.
    do lorenzo=1,inverse_dt
      t_real = real(t)
      if(flow_rate2>0.0_8) then
        if(mod(lorenzo,int(flow_rate2*inverse_dt))==0) then
          do piro=1,NP
            if(odor_flag(piro)==0) then
              odor_flag(piro)=1
              exit
            endif
          enddo
        endif
      endif

      Call
rk2_1(t_real,x_odor,y_odor,d_t,NP,Ak_re,Ak_im,tau_f,box_size,odor_flag)

      int_dt = t_real
      global_detect_flag = global_detect_flag_temp

      do i=1,n
        call
```

```
deriv_agent_1(x(i),y(i),Ak_re,Ak_im,theta,vx_current(i),vy_current(i))
         vx_estimate(i) = vx_estimate(i) +
lambda*(vx_current(i)-vx_estimate(i))*d_t
         vy_estimate(i) = vy_estimate(i) +
lambda*(vy_current(i)-vy_estimate(i))*d_t
         theta_estimate(i) = atan2(vy_estimate(i),vx_estimate(i))

         887 format(2F20.5)
         if(detect(i)==1) cycle

         detect(i)=0
         if(mod(lorenzo,detect_time)==0) then

           do j=1,NP
       ! This loop finds the odor particles that are at
           ! distance less than rd from the seracher
           ! r is the radius of the searcher's zone of detection of odor
particles
             if(odor_flag(j)==1) then
             dist = sqrt((x(i)-x_odor(j))**2+(y(i)-y_odor(j))**2)


             if (dist<=rd) then
          ! counting number of detections made
               num_detect(i,E) = num_detect(i,E)+1
               if(E==nconfig) then
                 write(15,9986) i,x(i),y(i)
               endif
               9986 format(I5,2F15.5)

                detect(i) =1 ; detect_flag(i)=1
                if(global_detect_flag_temp==0) first_detect_time = t
               !print*,"first_detect_time=",first_detect_time
                  global_detect_flag_temp=1
                 global_detect_flag=1
                 vx_odor_avg = 1.0 !vx_odor_avg + vx_odor(j)
                 vy_odor_avg = 0.0 !vy_odor_avg + vy_odor(j)

               endif
             endif
           enddo
         endif
       enddo !(i=1,n)
```

```
    enddo

    do i=1,n
      if(reach_flag(i)==0.and.global_detect_flag==1) then
      !print*,i,detect(i)
      !******** First compute vicsek model **************************
        vx_vicsek = 0.0 ; vy_vicsek=0.0

        do j=1,n
          !if (j==i) cycle
          if(reach_flag(j)==0) then
            dist = sqrt( (x(i)-x(j))**2 + (y(i)-y(j))**2 )
            if(dist<ra) then
              vx_vicsek = vx_vicsek + vx(j)
              vy_vicsek = vy_vicsek + vy(j)

            endif
          endif
        !print*,global_detect_flag,vx_vicsek,vy_vicsek
        enddo

      call random_number(rnd)
      delTheta = (rnd-half)*two*pi*eta
      cosdth = cos(delTheta)
      sindth = sin(delTheta)
      temp_vx = (vx_vicsek*cosdth - vy_vicsek*sindth)
      temp_vy = (vx_vicsek*sindth + vy_vicsek*cosdth)
      !print*,vy_vicsek,temp_vy
      vx_vicsek = temp_vx ; vy_vicsek= temp_vy

      !*********Vicsek model complete********************************




      !*********Compute cast and surge*******************************


        !print*,"clock=",clock(i)
        if(detect(i)==1) then
          vx_temp_i = -vx_mean_wind ; vy_temp_i = -vy_mean_wind
          ! It takes opposite direction of mean wind velocity
          clock(i)=0 ; turn_time(i)=0; vx_last(i) = -vx_mean_wind
      vy_last(i)=-vy_mean_wind
```

```
            vx_last_firm(i)=vx_last(i) ; vy_last_firm(i) =vy_last(i); detect(i)=0

            ! clock of the searcher is reset

          else

            if(detect_flag(i)<1) then ! possibility 1
              vx_temp_i = 0.0 ; vy_temp_i= 0.0
            endif


            if(detect_flag(i)==1) then

              if(clock(i)>0.and.clock(i)<turn_time(i)) then
                vx_temp_i = 0.0_8 ; vy_temp_i= vy_last(i)
                !print*, i,vx_temp_i,vy_temp_i,"2"
              endif

              if(clock(i)==turn_time(i)) then
                vy_last(i) = sign_flag(i)*cos(3.1415/4.0)
                sign_flag(i) = -1*sign_flag(i)
                vx_temp_i= vx_last_firm(i); vy_temp_i = -vy_last(i)
                !print*, i,vx_temp_i,vy_temp_i,"3"

              endif


              if(clock(i)>turn_time(i)) then

                vy_temp_i =  vy_last(i)
                vx_temp_i =  0.0_8
                clock(i) = 0
                turn_time(i) = turn_time(i) + 2
                !print*, i,vx_temp_i,vy_temp_i,"4"
              endif
              clock(i)=clock(i) + 1

            endif
          endif !if(detect(i)==1) then

          !vx_cs = vx_temp_i ; vy_cs = vy_temp_i
          vx_cs = (vx_temp_i*cos(theta_estimate(i)) -
vy_temp_i*sin(theta_estimate(i)))
          vy_cs = (vx_temp_i*sin(theta_estimate(i)) +
```

```
vy_temp_i*cos(theta_estimate(i)))


      !************************************************************

        norm = sqrt(vx_cs**2 + vy_cs**2)
        if(norm/=0) then
          vx_cs = vx_cs/ norm
          vy_cs = vy_cs / norm
        else
          vx_cs = 0.0
          vy_cs = 0.0
        endif

        norm = sqrt(vx_vicsek**2 + vy_vicsek**2)
        if(norm/=0) then
          vx_vicsek = vx_vicsek / norm
          vy_vicsek = vy_vicsek / norm
        else
          vx_vicsek = 0.0
          vy_vicsek = 0.0
        endif

        vx_new(i) = beta*(vx_cs) + (1.0-beta)*(vx_vicsek)
        vy_new(i) = beta*(vy_cs) + (1.0-beta)*(vy_vicsek)

        norm = sqrt(vx_new(i)**2 + vy_new(i)**2)
        if(norm/=0) then
          vx_new(i) = vx_new(i) / norm
          vy_new(i) = vy_new(i) / norm
        else
          vx_new(i) = 0.0
          vy_new(i) = 0.0
        endif

        x_new(i) = x(i) + vx_new(i)*v0*d_t_v
        y_new(i) = y(i) + vy_new(i)*v0*d_t_v

        reflect=0
        !********** Implementing Reflecting boundary ************
        if(x_new(i) >= x_max) then
                          vx_new(i) = -vx(i) ; reflect=1
                      endif

```

```
                    if(x_new(i) <= x_low) then
                          vx_new(i) = -vx(i) ;reflect=1
                      endif

                    if(y_new(i) >= y_max) then
                          vy_new(i) = -vy(i) ; reflect=1
                      endif

                    if(y_new(i) <= y_low) then
                         vy_new(i) = -vy(i) ; reflect=1
                    endif

                    if(reflect==1) then
          vx_last_firm(i) = -vx_last_firm(i)
          x_new(i) = x(i) + vx_new(i)*v0*d_t_v
                         y_new(i) = y(i) + vy_new(i)*v0*d_t_v
                    endif
                !********************** Reflecting BC Done ****************
        999 format(4F15.5)


        x(i) = x_new(i); y(i)=y_new(i)
        vx(i) = vx_new(i); vy=vy_new(i)



      else
    ! Do nothing
      endif !if(reach_flag(i)==0.and.global_detect_flag==1)) then

      if(t>0) then
        norm = sqrt(vx(i)**2 + vy(i)**2)
        if(norm/=0) then
          vx(i) = vx(i) / norm
          vy(i) = vy(i) / norm
        else
          vx(i) = 0.0
          vy(i) = 0.0
        endif
      endif

      if(E==nconfig) then
        write(i+100,999) x(i),y(i),vx(i),vy(i)
      endif
```

```
      x_dump(i,t) = x(1) ; y_dump(i,t) = y(1)

      9991 format(I5,4F15.4)


   enddo ! do i=1,n

   !*************** Computing the Vicsek order parameter*********
   if(global_detect_flag==1) then
   sumvx = sum(vx_new); sumvy = sum(vy_new)
   psi = sqrt(sumvx**2 + sumvy**2)/(real(n))
   psi_avg = psi_avg + psi
   endif
   !*********************************************************

   !*************** Computing the Massimo order parameter********
   sumvx = 0.0_8; sumvy=0.0_8 ; omega=0.0_8
   if(global_detect_flag==1) then
   do j=1,n
     omega = omega + sqrt( ((vx_new(j) + vx_mean_wind))**2 + ( (vy_new(j) +
vy_mean_wind))**2 )
   enddo
   omega = omega/(real(n))
   omega_avg = omega_avg + omega
   endif
   !*********************************************************

   do j=1,n
     if(reach_flag(j)==0) then
       dist = sqrt(x(j)**2 + y(j)**2)
       if(dist <= ra) then
             reach_count = reach_count+1
             reach_flag(j) = 1
                         total_reach_count = total_reach_count +1

           if(first_reach_flag<1) then
             winner_index =j
             winner_detection = num_detect(j,E)
             average_detection = 0
             rb_cnt = 0
             rg=0.0_8
             do k=1,n
               if(k/=j) average_detection = average_detection + num_detect(k,E)
```

```
                    dist = sqrt(x(k)**2 + y(k)**2)
                    rg = rg + dist
                    if(dist<=rb) rb_cnt = rb_cnt + 1 ! Finding agents within Rb
                 enddo
                 average_detection = average_detection /real(n-1)

                 9777 format (3I10,F15.5)
                 write(17,9777) E, winner_index,winner_detection,average_detection
                 write(18,*) E, rb_cnt  ! Less_than_Rb.txt
                 write(21,*) E, rg/ real(n)
              endif
                                  first_reach_flag =1
              write(8,*) j, abs(t-first_detect_time) ! reach_time.txt

           endif
        endif
     enddo

     global_detect_flag = global_detect_flag_temp
     if(first_reach_flag==1)exit
  enddo ! t=tstart,tend

  if(first_reach_flag<1.and.fails<2) then
     do j=1,n
        ! build filename -- i.dat
        write(fn1,fmt='(i0,a)') j, 'fail_data.txt'
        open(unit=20,file=fn1,status='replace',action='write')

        do i=tstart,tend
           write(20,*) x_dump(j,i),y_dump(j,i)
        enddo

        close(unit=20)
     enddo
     fails = fails +1

  !if(fails>5) exit
  endif

  write(7,998) E,reach_count
  998 format (2I10,F10.5)
!if(first_reach_flag==1)exit
```

```
open(unit=13,file="number_of_detections.txt",status='replace',action='write')
do i=1,n
  do j=1,E
    write(13,*) i, num_detect(i,j)

  enddo
enddo
 close(unit=13)

write(19,*) E, psi_avg/real(t-first_detect_time)
write(20,*) E, omega_avg/real(t-first_detect_time)

enddo !E=1,nconfig

open(unit=9,file="total_reach_count.txt",status='replace',action='write')
write(9,*) real(total_reach_count)/ real(nconfig)
 close(unit=9)

call cpu_time(cpu2)

write(10,*) "Job complete, Run Time=",abs(cpu2-cpu1)
do i=1,20
 close(unit=i)
enddo
END
```

G.2.2 Code to simulate turbulent flow

```
module flow_mod
 use inout
 implicit none

 !Number of particles released and flux label
 integer, save :: NP
 !Stochastic flow parameters and variables
 integer, parameter :: NK=8
 real*8, save :: u0x=1.d0
 real*8,allocatable,dimension(:,:), save :: Ak_im,Ak_re
 real*8,allocatable,dimension(:,:), save :: kx,ky,sig
 real*8 :: diff

```

```fortran
contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Initialise amplitudes and modes !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine initfluid_1(uf,L)
    implicit none
    real*8, allocatable,dimension(:) :: ks
    real*8, dimension(:), intent(in) :: uf,L
    real*8 :: pi2=6.283185d0,scra
    integer :: k,is

    allocate(ks(NS))

    do is = 1,NS
       ks(is)=pi2/L(is)

       kx(is,1) = 1.d0*ks(is) ; kx(is,2) = 1.d0*ks(is)
       kx(is,3) = 0.d0*ks(is) ; kx(is,4) = -1.d0*ks(is)
       kx(is,5) = -1.d0*ks(is); kx(is,6) = -1.d0*ks(is)
       kx(is,7) = 0.d0*ks(is) ; kx(is,8) = 1.d0*ks(is)

       ky(is,1) = 0.d0*ks(is) ; ky(is,2) = 1.d0*ks(is)
       ky(is,3) = 1.d0*ks(is) ; ky(is,4) = 1.d0*ks(is)
       ky(is,5) = 0.d0*ks(is) ; ky(is,6) = -1.d0*ks(is)
       ky(is,7) = -1.d0*ks(is); ky(is,8) = -1.d0*ks(is)

       sig(is,1)=1.0d0  ; sig(is,2)=0.5d0
       sig(is,3)=1.0d0  ; sig(is,4)=0.5d0
       sig(is,5)=1.0d0  ; sig(is,6)=0.5d0
       sig(is,7)=1.0d0  ; sig(is,8)=0.5d0

       scra=0.d0
       do k=1,NK
          scra=scra+sig(is,k)
       end do

       do k=1,NK
          sig(is,k)=uf(is)*sig(is,k)*sqrt(2.d0)/sqrt(scra)/ks(is)
       end do
    end do

  end subroutine initfluid_1

```

```fortran
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Compute velocity at given position/time !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 subroutine deriv_1(xp,yp,uxp,uyp,ip,odor_flag)
    implicit none
    integer, intent(in) :: ip
    Real*8, dimension (:),intent(inout) :: xp,yp
    Real*8, dimension (:),intent(inout) :: uxp,uyp
    Real*8 :: xx,yy,arg
    integer :: i,k,is
    integer,dimension(:),intent(in) :: odor_flag

    do i = 1,ip
  if(odor_flag(i)==1) then
       uxp(i)=u0x;uyp(i)=0.d0
       xx=xp(i);yy=yp(i)
       do is=1,NS
          do k=1,NK
             arg=kx(is,k)*xx+ky(is,k)*yy
             uxp(i)= uxp(i) -
Ak_im(is,k)*ky(is,k)*cos(arg)-Ak_re(is,k)*ky(is,k)*sin(arg)
             uyp(i)= uyp(i) +
Ak_re(is,k)*kx(is,k)*sin(arg)+Ak_im(is,k)*kx(is,k)*cos(arg)
          end do
       end do
  endif
    end do

  end subroutine deriv_1

!!!!!!!!!!!!!!!!!!!!!!!
!!! Integrator RK2  !!!
!!!!!!!!!!!!!!!!!!!!!!!
  subroutine rk2_1(t,xp,yp,dt,ip,Ak_re,Ak_im,tau_f,box_size,odor_flag)
    implicit none
    integer,intent(inout) :: ip
    real*8, dimension(:), intent(inout) :: xp,yp,tau_f
    real*8, allocatable,dimension (:) :: xp1,dxp,dxp1,invtau_f
    real*8, allocatable,dimension (:) :: yp1,dyp,dyp1
    real*8, allocatable,dimension (:,:), intent(inout) :: Ak_re,Ak_im
    real*8 :: dt2,r1,r2,gauss1,gauss2,pi=3.14159265358979323844d0,u1,u2
    real*8, intent(in) :: dt,t,box_size
    integer :: k,i,j,is
    integer,dimension(:),intent(inout) :: odor_flag
```

```

    allocate(invtau_f(NS))
    allocate(xp1(ip),dxp(ip))
    allocate(yp1(ip),dyp(ip))
    allocate(dxp1(ip),dyp1(ip))

    dt2=0.5d0*dt
    do is=1,NS
       invtau_f(is)=1.d0/tau_f(is)
    end do

    do is=1,NS
       do k=1,NK
          call random_number(r1)
          call random_number(r2)
          gauss1=sqrt(-2.d0*log(r1))*cos(2.d0*pi*r2)
          gauss2=sqrt(-2.d0*log(r1))*sin(2.d0*pi*r2)
          Ak_re(is,k) = Ak_re(is,k) - invtau_f(is)*Ak_re(is,k)*dt +
sig(is,k)*sqrt(2.d0*dt*invtau_f(is))*gauss1
          Ak_im(is,k) = Ak_im(is,k) - invtau_f(is)*Ak_im(is,k)*dt +
sig(is,k)*sqrt(2.d0*dt*invtau_f(is))*gauss2
       end do
    end do

    do i=1,ip
  if(xp(i)>box_size) then
    odor_flag(i)=0
    call random_number(u1)
    call random_number(u2)
    xp(i)=0.2*u1-0.1 ; yp(i)=0.2*u2-0.1
  endif
    enddo


    call deriv_1(xp,yp,dxp,dyp,ip,odor_flag)

    do k=1,ip
  if(odor_flag(k)==1) then
            xp1(k)=xp(k)+dt2*dxp(k)
            yp1(k)=yp(k)+dt2*dyp(k)
  endif
    end do

    call deriv_1(xp1,yp1,dxp1,dyp1,ip,odor_flag)
```

```
    do k=1,ip
  if(odor_flag(k)==1) then
          xp(k)=xp(k)+dt*dxp1(k)
          yp(k)=yp(k)+dt*dyp1(k)
  endif
   end do

    deallocate(xp1,dxp)
    deallocate(yp1,dyp)
    deallocate(dxp1,dyp1)
    deallocate(invtau_f)
    return
  end subroutine rk2_1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Subroutine for the agent to know the local wind direction (theta) !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  subroutine deriv_agent_1(x,y,Ak_re,Ak_im,theta,vx,vy)
    implicit none

    real*8, intent(inout) :: theta,vx,vy
    real*8, dimension (:,:),intent(inout) :: Ak_re,Ak_im
    real*8, intent(in) :: x,y
    real*8 :: arg,r1,r2,gauss1,gauss2,pi=3.1415926535d0,invtau_f,xnew,ynew
    integer :: k,is

    !dt2=0.5d0*dt

    vx=u0x ; vy=0.d0
    do is=1,NS
       do k=1,NK
          arg=kx(is,k)*x+ky(is,k)*y
          vx = vx - Ak_im(is,k)*ky(is,k)*cos(arg)-Ak_re(is,k)*ky(is,k)*sin(arg)
          vy = vy + Ak_re(is,k)*kx(is,k)*sin(arg)+Ak_im(is,k)*kx(is,k)*cos(arg)
       end do
    end do

    theta=atan(vy/vx)

  end subroutine deriv_agent_1

!!!!!!!!!!!!!!!!!!!!!!!
!!! Diffusive term !!!
```

```
!!!!!!!!!!!!!!!!!!!!!!!
  subroutine diffusion(xp,yp,ip)
    implicit none
    Real*8, dimension (:) :: xp,yp
    Real*8 :: r1,r2,gauss1,gauss2,pi=3.14159265358979323844d0
    integer,intent(in) :: ip
    integer :: k
    !call init_random_seed()

    do k=1,ip
       call random_number(r1)
       call random_number(r2)
       gauss1=sqrt(-2.d0*log(r1))*cos(2.d0*pi*r2)
       gauss2=sqrt(-2.d0*log(r1))*sin(2.d0*pi*r2)
       xp(k)=xp(k)+diff*gauss1
       yp(k)=yp(k)+diff*gauss2
    end do
    return
  end subroutine diffusion

end module flow_mod
```

Bibliography

[1] A. Procaccini, A. Orlandi, A. Cavagna, I. Giardina, F. Zoratto, D. Santucci, F. Chiarotti, C. Hemelrijk, E. Alleva, G. Parisi, and C. Carere. *Animal Behaviour*, 82(4):759 – 765, 2011. ISSN 0003-3472.

[2] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic. *PNAS*, 105:1232–1237, 2008.

[3] A. Attanasi, A. Cavagna, and L. Del Castello et al. *Nature Phys*, 10: 691–696, 2014.

[4] A. Goodenough, N. Little, W. Carpenter, and A. Hart. *PLoS ONE*, 12(6): e0179277, 2007.

[5] Ch. Becco, N. Vandewalle, J Delcourt, and P. Poncin. *Phys A*, 367:487, 2006.

[6] N. Makris, P. Ratilal, S. Jagannathan, Z. Gong, M. Andrews, I. Bertsatos, O. Godø, R. Nero, and J. Jech. *Science*, 323:1734–1737, 2009.

[7] T. Pitcher. *Animal Behaviour*, 31:611–613, 1883.

[8] H. P. Zhang, A. Be'er, R. S. Smith, E.-L. Florin, and H. L. Swinney. *Euro Phys Lett*, 87:48011, 2009.

[9] N. C. Darnton, L. Tuner, S. Rojevsky, and H. C. Berg. *Biophys. J*, 98: 2082, 2010.

[10] A. Czirók, E. Ben-Jacob, I. Cohen, and T. Vicsek. *Phys Rev E*, 54:1791, 1996.

[11] M. Milinski and R. Heller. *Nature*, 275:642–644, 1978.

[12] T. Pitcher, A. Magurran, and I Winfield. *Behav Ecol Sociobiol*, 10:149–151, 1982.

[13] Julia Parrish and William Hamner. *Animal Groups in Three Dimentions.* Cambridge University Press, 1997.

[14] I. Aoki. *Bulletin of the Japanese Society of Scientific Fisheries*, 48(8): 1081–1088, 1982.

[15] C. Reynolds. *SIGGRAPH 1987 Conference Proceedings*, 21(4):25, 1987.

[16] A. Kudrolli, G. Lumay, D. Volfson, and L. S. Tsimring. *Phys Rev Lett*, 100: 058001, 2008.

[17] V. Narayan, N. Menon, and S. Ramaswamy. *J. Stat. Mech.*, 2006:P01005, 2006.

[18] I. Buttinoni, J. Bialké, F. Kümmel, H. Löwen, C. Bechinger, and T. Speck. *Phys Rev Lett*, 110:238301, 2013.

[19] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and Ofer Shochet. *Phys. Rev. Lett*, 75:1226–1229, 1995.

[20] G. Baglietto and E. V. Albano. *Phys Rev E*, 80:050103(R), 2009.

[21] H. Chaté, F. Ginelli, G. Grégoire, and F. Raynaud. *Phys. Rev. E*, 77: 046113, 2008.

[22] C. Huepe and M. Aldana. *Physica A*, 387:2809–2822, 2008.

[23] L. Barberis and F. Peruani. *Phys Rev Lett*, 117:248001, 2016.

[24] F. Ginelli, F. Peruani, M.-H. Pillot, H. Chaté, G. Theraulaz, and R. Bon. *Proceedings of the National Academy of Sciences*, 112(41):12729–12734, 2015.

[25] D. McComb and S. Kajiura. *J. Exp. Biol.*, 211:482–490, 2008.

[26] G. Martin and G. Katzir. *Brain Behav Evol*, 53:55–66, 1999.

[27] M. Durve, A. Saha, and A. Sayeed. *Eur Phys J E*, 41:49, 2018.

[28] C. Huepe and M. Aldana. *Physica A*, 387:2809–2822, 2008.

[29] G. Baglietto and E. V. Albano. *Comp. Phys. Comm.*, 180:527, 2009.

[30] M. Durve and A. Sayeed. *Phys Rev E*, 93:052115, 2016.

[31] Andrea Cavagna, Irene Giardina, Asja Jelic, Stefania Melillo, Leonardo Parisi, Edmondo Silvestri, and Massimiliano Viale. *Phys. Rev. Lett.*, 118: 138003, 2017.

[32] B-M. Tian, H.-X. Yang, W. Li, W-X. Wang, B-H. Wang, and T. Zhou. *Phys. Rev. E*, 79:052102, 2009.

[33] J. Gao, S. Havlin, X. Xu, and H. E. Stanley. *Phys Rev E*, 84:046115, 2011.

[34] A Costanzo and C K Hemelrijk. *J. Phys. D: Appl. Phys.*, 51:134004, 2018.

[35] I. Couzin, J. Krause, R. James, G. Ruxton, and N. Franks. *Journal of Theoretical Biology*, 218(1):1–11, 2002.

[36] D. Grossman, I. Aranson, and E. Jacob. *New J. Phys.*, 10:023036, 2008.

[37] J. Shäfer, S. Dippel, and D. E. Wolf. *J. Phys. I France*, 6(1):5–20, 1996.

[38] A Cavagna, L. Del Castello, I. Giardina, T. Grigera, A. Jelic, S. Melillo, T. Mora, L. Parisi, E. Silvestri, and M. Viale A. Walczak. *J. Stat. Phys.*, 158:601–627, 2005.

[39] P. Rahmani, F. Peruani, and P. Romanczuk. *arXiv:1907.11691*, 2019.

[40] J. Herbert-Read, A. Perna, R. Mann, T. Schaerf, D. Sumpter, and A. Ward. *Proceedings of the National Academy of Sciences*, 108(46): 18726–18731, 2011.

[41] Y. Katz, K. Tunstrøm, C. Ioannou, C. Huepe, and I. Couzin. *Proceedings of the National Academy of Sciences*, 108(46):18720–18725, 2011.

[42] R. Lukeman, Y-X. Li, and L. Edelstein-Keshet. *Proceedings of the National Academy of Sciences*, 107(28):12576–12580, 2010.

[43] C. Loudon. In Gary Blomquist and Richard Vogt, editors, *Insect Pheromone Biochemistry and Molecular Biology*, pages 609 – 630. Academic Press, San Diego, 2003.

[44] J. Murlis and C. Jones. *Physiol. Entomol*, 6:71–86, 1981.

[45] T. Dekker, M. Geier, and R. Cardé. *J. Exp. Biol.*, 208:2963–2972, 2005.

[46] R. Cardé. *Current Biol.*, 25:R793, 2015.

[47] M. Geier, O. Bosch, and J. Boeckh. *Exp. Biol.*, 202:1648, 1998.

[48] J. Elkinton, C. Schal, T. Ono, and R. Cardé. *Physiol. Entomol.*, 12: 399–406, 1987.

[49] J. Kennedy, A. Ludlow, and C. Sanders. *Nature*, 288:475, 1980.

[50] L. Kuenem and R. Cardé. *Physiological Entomology*, 19:15–29, 1994.

[51] J. Kennedy. *Physiol. Entomol.*, 8:109–120, 1983.

[52] T. Baker and K. Haynes. *Physiological Entomology*, 12:263–279, 1987.

[53] M. Willis and E. Arbas. *Manduca sexta. J. Comp. Physiol. A.*, 178: 699–706, 1991.

[54] N. Vickers and T. Baker. *J. Insect Behav.*, 5:669–687, 1992.

[55] B. Szabó, G. Szöllösi, B. Gönci, Zs. Jurányi, D. Selmeczi, and T. Vicsek. *Phys Rev E*, 74:061908, 2006.

[56] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[57] C. Watkins. *Machine Learning*, 8:279, 1992.

[58] L. Panait and S. Luke. *Autonomous Agents and Multi-Agent Systems*, 11: 387, 2005.

[59] L. Buşoniu, R. Babŭska, and B. Schutter. *IEEE transactions on systems, man, and cybernetics-part C: application and reviews*, 38, 2008.

[60] T. Vicsek and A. Zafeiris. *Phys. Rep.*, 517:71, 2012.

[61] J. Buhl, D. Sumpter, I. Couzin, , J. Hale, E. Despland, E. Miller, and S. Simpson. *Science*, 312:1402, 2006.

[62] J. Parrish, S. Viscido, and D. Grünbaum. *Biol. Bull*, 202:296–305, 2002.

[63] J. Puckett, D. Kelley, and N. Ouellette. *Scientific Reports*, 4(4766), 2014.

[64] Francesco Ginelli, Fernando Peruani, Marie-Helène Pillot, Hugues Chaté, Guy Theraulaz, and Richard Bon. Intermittent collective dynamics emerge from conflicting imperatives in sheep herds. *Proceedings of the National Academy of Sciences*, 112(41):12729–12734, 2015.

[65] A. Cavagna, A. Cimarelli, I. Giardina, G. Parisi, R. Santagati, F. Stefanini, and R. Tavarone. *Mathematical Models and Methods in Applied Sciences*, 20:1491–1510, 2010.

[66] W. Bialek, A. Cavagna, I. Giardina, T. Mora, E. Silvestri, M. Viale, and A. Walczak. Statistical mechanics for natural flocks of birds. *Proceedings of the National Academy of Sciences*, 109(13):4786–4791, 2012.

[67] I. D. Couzin and J. Krause. *Adv Study Behav*, 32:1, 2003.

[68] H. Hildenbrandt, C. Carere, and C. Hemelrijk. *Behavioral Ecology*, 21(6): 1349–1359, 2010.

[69] Daniel JG Pearce, Adam M Miller, George Rowlands, and Matthew S Turner. Role of projection in the control of bird flocks. *Proceedings of the National Academy of Sciences*, 111(29):10422–10426, 2014.

[70] C. David, J. Kennedy, and A. Ludlow. *Nature*, 303:804–806, 1983.

[71] J. Murlis and C. Jones. *Physiol. Entomol.*, 6:71–86, 1981.

[72] A. Celani, E. Villermaux, and M. Vergassola. *Phys Rev X*, 4:041015, 2014.

[73] A. Lilienthal, D. Reimann, and A. Zell. *Autonome Mobile Systeme 2003*, pages 150–160, 2003.

[74] G. Ferri, E. Caselli, V. Mattoli, A. Mondini, B. Mazzolai, and P. Dario. *IEEE/RAS-EMBS Conf. Proc. BioRob-2006*, pages 573–578, 2006.

[75] H. Ishida, H. Tanaka, H. Taniguchi, and T. Moriizumi. *Autonomous Robots*, 20:231–238, 2006.

[76] F. Grasso and J. Atema. *J. Environmental Fluid Mechanics*, 2:95, 2002.

[77] T. Lochmatter and A. Martinoli. *Experimental Robotics. Springer Tracts in Advanced Robotics*, 54:473–482, 2009.

[78] Eric Bonabeau, G. Theraulaz, and M Dorigo. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, USA, 1999.

[79] C. Torney, Z. Neufeld, and I. Couzin. *Proceedings of the National Academy of Sciences*, 106(52):22055–22060, 2009.

[80] C. Ioannou, M. Singh, and I. Couzin. *The American Naturalist*, 186: 000–000, 06 2015.

[81] A. Berdahl, C. Torney, C. Ioannou, J. Faria, and I. Couzin. *Science*, 339 (6119):574–576, 2013.

[82] N. Miller, S. Garnier, A. Hartnett, and I. Couzin. *Proceedings of the National Academy of Sciences*, 110(13):5263–5268, 2013.

[83] M. Vergassola, E. Villermaux, and B. Shraiman. *nature*, 445:406–409, 2007.

[84] E. Karpas, A. Shklarsh, and E. Schneidman. *Proc. Nat. Acad. Sci.*, 111: 5589–5594, 2017.

[85] J-B. Masson, M. Bailly, and M. Vergassola. *J. Phys. A: Math. Theor.*, 42: 434009, 2009.

[86] C. Song, Y. He, B. Ristic, and X. Lei. *Robotics and Autonomous Systems*, 125:103414, 2020.

[87] C. Song, Y. He, B. Ristic, L. Li, and X. Lei. *J. Phys. A: Math. Theor.*, 52: 485202, 2019.

[88] S. Zhang, D. Martinez, and J-B. Masson. *Frontiers Robot*, 2:12, 2015.

[89] B. Ristic, D. Angley, B. Moran, and J-L. Palmer. *Sensors*, 17:918, 2017.

[90] Hadi Hajieghrary, M. Ani Hsieh, and Ira B. Schwartz. *Physics Letters A*, 380(20):1698 – 1705, 2016.

[91] E. Balkovsky and B. Shraiman. *Proceedings of the National Academy of Sciences*, 99(20):12589 –12593, 2002.

[92] T. Baker, M. Willis, and P. Phelan. *Physiological Entomology*, 9:365–376, 1984.

[93] T. Vicsek and A. Zafeiris. *Phys. Rep.*, 517:71, 2012.

[94] F. Ginelli. The physics of the vicsek model. *Europ. Phys. J. Special Topics*, 225(11-12):2099–2117, 2016.

[95] G. Boffetta, A. Celani, and M. Vergassola. Inverse energy cascade in two-dimensional turbulence: Deviations from gaussian behavior. *Phys. Rev. E*, 61(1):R29, 2000.

[96] G. Boffetta and R. E. Ecke. Two-dimensional turbulence. *Annu. Rev. Fluid Mech.*, 44:427–451, 2012.

[97] A. Alexakis and L. Biferale. *Phys. Rep.*, 767-769:1–101, 2018.

[98] Adrian ŠoŠić, Wasiur R. KhudaBukhsh, Abdelhak M. Zoubir, and Heinz Koeppl. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1413–1421, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[99] J. Bec. *Phys. Fluids.*, 15:L81–L84, 2003.

[100] J. Bec, A. Celani, M. Cencini, and S. Musacchio. *Phys. Fluids.*, 17:073301, 2005.