Learning to Flock with Policy Gradients

Mehmet Bilgehan Bezcioglu
Department of Computer Science
Stanford University
bilgehan@stanford.edu

***Abstract*—Automating the process of manual hand-tuning of parameters of any swarm robotics mechanism has been an interest of the research community for a long time. In this paper, we show that by constructing efficient reward functions and treating the multi-agent system as a single entity we can effectively learn optimal policies for forming a flock. We provide investigation on the convergence properties of two learning algorithms that belong to the same family of policy gradient methods in reinforcement learning: DDPG and TRPO. This work uses the collective motion dynamics that is based on linear spring-like forces between self-propelled particles in an active crystal. We tuned the damping coefficients of the dynamic model for a swarm population of $N = 100$ using the learned policies by DDPG and TRPO models effectively. We investigate the application of DDPG and TRPO in a centralised multi-agent approach, where we have a global state space matrix that is accessible by all robots in the environment. We show that both methods converge to optimal policies in learning emergent collective flocking behaviours, qualitatively and quantitatively.**

***Index Terms*—Swarm Robotics, Reinforcement Learning, Multi-agent Learning.**

I. Introduction

Swarm robotics is defined as "the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment" according to [1]. There are several swarming behaviours that are important such as aggregation and clustering, collective manipulation, transportation, flocking and collective motion, shepherding some of which are bio-inspired. Swarm systems are known to be robust to single point-of-failures due to their nature of decentralized control schemes. Furthermore, there is an added benefit of flexibility in solving problems such as collective transport or manipulation of objects. For instance, a team of robots can collectively achieve much higher torques and forces than they would individually, due to the weakness of their actuators. However, optimizing swarm systems mostly requires manual hand tuning, which can be a tedious task. With the advancements in deep and reinforcement learning we can enable swarm systems to self-tune their parameter space to their optimal values. In this work, we tune the collective motion dynamic parameters via policy gradient based algorithms. The input to both DDPG and TRPO models are the position and orientation of each robot in a swarm population of $N = 100$. The output is the damping coefficients of AES model,$\{k, \beta, \alpha\}$, that are used in the dot-product computation of the collective behaviour for each robot in the flock. By choosing the optimal values of coefficients the swarm can learn to flock, or if chosen badly it can diverge and scatter around in the environment. Multi-agent reinforcement learning for cooperative swarm intelligence is a difficult task, mainly because each agent has partial access to the global state of the swarm, and also due to the curse of dimensionality for a large number of agents in a swarm. Hüttenrauch et al. [2] explored the incorporation of actor-critic approach, where critic has access to the swarm's global state, but actors are based on the locally observed sensory data. They used a variant of DDPG for the simulation of 2D robots. However, their method was decentralised and did not explore the applicability of a centralised application of reinforcement learning to a biologically inspired dynamic model. In [3], further use of mean embedding distributions was introduced, where each agent is considered a sample. They studied global and local cases with a communication protocol proposed for the local case in pursuit evasion and rendezvous problems in swarm systems. In a followup study [4], they proposed a leader-follower mechanism by using inverse RL to recover unknown reward functions in a flock of birds. Part of this work has been published as our previous work in [5].

II. Dataset

The nature of this work required the data collection to be done online actively whilst the agent is interacting with the environment and therefore collecting data into the replay buffer. We apply certain data pre-processing to the data collected by the agents in the environment to maximise the generalisation capabilities of both methods: DDPG and TRPO. We normalise by centering the data on zero mean and then dividing by the standard deviation to ensure uniform variance at all time steps. Our data is represented in an image-like format where channels in image correspond to R-,G-,B channels and are composed of three channels, where each entry in pixel point is an agent's x,y,θ values. For a swarm population of 100 agents this leads to 10x10x3 format of an image-like representation. Normalization is done channel-wise, meaning we calculate the mean of each x,y,θ values and then obtain three means and three standard deviations to normalise channel-wise. The reason we compute normalisation is so that the relative distance of the centre of the swarm entered into the models remains unaffected by the position of the centre of the swarm in world frame, by ensuring the relative distances between each agent remains similar, even when the swarm has moved in significant distances to another point in the world frame.

III. Methodology

In this paper, we are considering the continuous state-space in a continuous action domain. We train a group of agents to maximise their expected reward based on their proximity and distances to each other via reward signals. We investigate two policy gradient methods: DDPG and TRPO. In this section, we formulate the preliminaries and the problem to be solved. Part of this section has been published in our previous work in [5].

A. Flocking dynamics

This paper uses the Active Elastic Sheet (AES) model originally proposed by [6] that consists of a force for each robot that influences the linear and angular velocities of each robot in the flock. In this model, there are local spring-like interactions that are linear. A more detailed explanation of the mechanics of AES can be found in [6].

Individual linear and rotational velocities are given as,

$$\dot{\mathbf{x}}_\mathbf{i}(t) = \Big(v_0(t) + \alpha[(\mathbf{f}_\mathbf{i}(t) + D_r\hat{\epsilon}_\mathbf{r}(t)).\hat{\mathbf{n}}_\mathbf{i}(t)]\Big)\hat{\mathbf{n}}_\mathbf{i}(t) \ , \quad (1)$$

$$\dot{\theta}_i(t) = \beta\{[\mathbf{f}_\mathbf{i}(t) + D_r\hat{\epsilon}_r(t)]\hat{\mathbf{n}}_\mathbf{i}^{\perp}(t)\} + D_\theta\epsilon_\theta(t) \ , \quad (2)$$

where $\hat{n}_i(t) = \begin{bmatrix} \cos\theta_i(t) \\ \sin\theta_i(t) \end{bmatrix}$ is the heading vector and $\hat{\mathbf{n}}_\mathbf{i}^{\perp}(t)$ is the vector perpendicular to it.

Force vector for the flock can be computed by the summation of the individual force vectors for each individual *i* over its respective neighbours in set *S*, where *j* is a member of set *S* as,

$$\mathbf{f}_\mathbf{i}(t) = \sum_{j\in S} \frac{-k}{l_{ij}}(\|\mathbf{r}_\mathbf{ij}(t)\| - l_{ij})\frac{\mathbf{r}_\mathbf{ij}(t)}{\|\mathbf{r}_\mathbf{ij}(t)\|} \quad (3)$$

where $\mathbf{r}_\mathbf{ij}(t) = \mathbf{x}_\mathbf{i}(t) - \mathbf{x}_\mathbf{j}(t)$ is defined as the relative position vector of neighbour *j* in frame *i*, l_{ij} is defined as the desired equilibrium spring length that the model should maintain. We used the Euler integration method. Summing over each robot in swarm to compute a global force metric, $F_g = \frac{1}{N}\|\sum_{i=1}^N \mathbf{f}_\mathbf{i}\|$ and $\psi = \sum_{n=1}^N \exp i\theta_n$ where $\psi \in (0,1)$ is the degree of alignment of the swarm. When fully aligned (i.e. flocking) $\psi = 1$, when completely misaligned (i.e. not flocking) $\psi \sim 0$.

B. Deep Deterministic Policy Gradients

In this section, we describe the building blocks of DDPG: Q-learning, and policy learning. DDPG iteratively uses the Bellman equation to learn the Q-function, which it then uses the Q-function to learn the policy. It employs the actor-critic learning paradigm and works on continuous action spaces.

1) Q-learning

Q-value is defined as the 'quality' of an action taken by an agent interacting in an environment. Optimal action-value function obeys the Bellman equation, which states that the value of the current state is the immediate reward summed with the recursive value of discounted future states, as shown below

$$Q^*(s,a) = \max_\pi \mathbf{E}\Big[R(\tau)|s_0 = s, a_0 = a\Big] \quad (4)$$

where τ is a trajectory, s_0 is the initial starting state drawn from the state distribution $s_0 \sim p_0(.)$. As our action space is continuous and it is not feasible to use value iteration to solve Q_i as $i \to \infty$, we use function approximation to learn to estimate the optimal action-value function. Once action-value function converges to its optimal solution, we can find the policy, also known as the optimal policy, that maximises the expected return starting from state s and taking action a following the optimal policy thereafter. Optimal action-value is defined as the maximum expected return attained by starting in state s and taking action a, and following the optimal policy forever in the future states. Our policy is a deterministic policy that is parameterised by θ,

$$a^*(s) = \operatorname*{argmax}_a Q^*(s,a). \quad (5)$$

DDPG approximates the optimal Bellman equation solution with an artificial neural network (ANN), also known as the critic, and trains it by minimising the mean-squared Bellman error(MSBE) loss function,

$$L(D) = \mathbf{E}\left[\left(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_\phi(s',a'))\right)^2\right], \quad (6)$$

where, D is transition (ϕ, s, a, r, s', d), Q is the approximator function parameterised by ϕ, r is the reward, γ is the discount factor for future expected rewards, s' is the next state drawn from the probability distribution $P(.|s,a)$.As the target values are also dependent on the network parameters that are being optimised. Therefore, to avoid divergence, the use of time delayed versions of actor and critic networks to ensure the stability of the learning was proposed [7]. These networks are called target networks and basically lag the online actor and critic networks by a time delay of τ that is also a hyperparameter in the algorithm.

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi' \quad (7)$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \quad (8)$$

Hence, our loss function after incorporating the target networks is

$$L(D) = \mathbf{E}\left[\left(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_{\phi'}(s', \mu_{\theta'}(s)))\right)^2\right] \quad (9)$$

We use stochastic gradient descent to optimise the critic network using the following gradient,

$$\nabla_\phi L(D) = \mathbf{E}\Big[Q_\phi(s,a) - \Big(r + \gamma(1-d)\max_{a'} Q_{\phi'}(s',a')\Big)\nabla_\phi Q_\phi(s,a)\Big] \quad (10)$$

2) Policy Learning

Our goal here is to learn a deterministic policy that maximises the expected return starting from state *s*. This is formulated using an objective function, which is also a measure of performance,

$$J(\theta) = \mathbf{E}\Big[Q_\phi(s, \mu_\theta(s))\Big]. \quad (11)$$

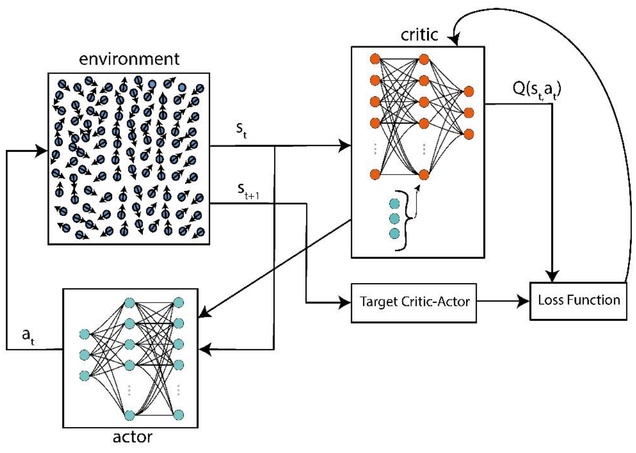

Fig. 1. Architecture overview of DDPG with multi-agent learning. Environment constitutes of *N* robots, actor produces the action a_t, which is observed as s_t. Critic provides the actor with action gradients. Taken from our previous work in [5]

Policy parameters are updated using the policy gradients obtained from the critic network. Since the action space is continuous, we assume that the Q-function is continuously differentiable with respect to action space. We use the chain rule to differentiate the Q-value with respect to actions, and then actions with respect to policy parameters to provide a policy gradient for each parameter in the actor network:

$$\nabla_\theta J(\theta) = \nabla_a Q(s,a) \nabla_\theta \mu(s|\theta). \tag{12}$$

Since we need to maximise the objective function, we negate the policy gradients and optimise the parameters using stochastic gradient descent,

$$\theta \leftarrow \theta + \alpha \left(\nabla_\theta J(\theta) \right). \tag{13}$$

C. Trust Region Policy Optimization

TRPO is a policy gradient algorithm that monotonically improves the local approximation of the optimisation objective with a KL divergence constraint [8]. TRPO uses Minorization-Maximization to improve a lower bound on the local expected reward iteratively and is guaranteed to monotonically improve the policy. The lower bound is bounded at the current evaluation of θ, and maximising the lower-bound leads to non-decreasing value function, hence, monotonic improvements in new policies. Our goal is to find policy parameters that maximise the value function expressed in terms of advantage over current policy can be locally approximated to remove the dependency on the frequencies of the new policy by substituting the frequencies with those from current policy as,

$$L(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{14}$$

where $L(\theta_{old}) = V(\theta_{old})$ and the gradient of L is equal to gradient of V at θ_{old}. The lower bound on π_{new} is guaranteed in the form of,

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\beta^2 \tag{15}$$

which is tight at the $\pi_{new} = \pi_{old}$, and where $\epsilon = \max_s \| \mathbb{E}_{a\sim\pi'(a|s)}[A_\pi(s,a)] \|$

Our goal is to find new policy that maximises the lower bound as the objective function.,

$$\max_\theta L_{\theta old}(\theta_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{max}(\theta_{old}, \theta_{new}) \tag{16}$$

where C is the penalty term. Due to the presence of the KL-penalty term, solving this leads to small step sizes that would be too small in magnitude [8]. Converting this optimisation algorithm to Lagrange duality problem where the constraint is the KL-divergence between new and old policies, we have our objective as to maximise,

$$\theta_{k+1} := \underset{\theta}{\operatorname{argmax}}\, L(\theta_k, \theta) \qquad s.t. \quad \bar{D}_{KL}(\theta_k||\theta) \leq \delta \tag{17}$$

$L(\theta_k, \theta)$ is the surrogate advantage function being optimized, and $\bar{D}_{KL}(\theta_k||\theta)$ is the KL-divergence between old and new policies evaluated at the states that were visited by the old policy.$L(\theta_k, \theta)$ and $\bar{D}_{KL}(\theta_k||\theta)$ are given as,

$$L(\theta_k, \theta) = \underset{s,a\sim\pi_{\theta_k}}{\mathbb{E}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \right] \tag{18}$$

$$\bar{D}_{KL}(\theta_k||\theta) = \underset{s\sim\pi_{\theta_k}}{\mathbb{E}} \left[D_{KL}(\pi_\theta(.|s)||\pi_{\theta_k}(.|s)) \right] \tag{19}$$

L is approximated linearly and constraint, KL, is approximated quadratically as shown

$$L \sim g^T(\theta - \theta_k)$$
$$D_{KL} \sim \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k)$$

which is equivalent to optimising

$$\theta_{k+1} := \underset{\theta}{\operatorname{argmax}}\, g^T(\theta - \theta_k) \tag{20}$$

$$s.t. \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \leq \delta \tag{21}$$

Authors of [8] proposed conjugate-gradient method to solve for the inverse of the Hession matrix, which is $x = H^{-1}g$, hence, solving for $Hx = g$ without explicitly having to compute the full matrix. Solution to new policy then becomes,

$$\theta_{k+1} := \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{22}$$

Line search is done iteratively to find the smallest positive integer j such that the constraint is satisfied, and the surrogate objective is improved. This step ensures the step size is not too large and hence prevents explosion of the solution.

$$\theta_{k+1} := \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{23}$$

where $\alpha in(0,1)$ is the backstepping coefficient and j is the smallest positive integer that satisfies the constraint and optimises the surragate objective.

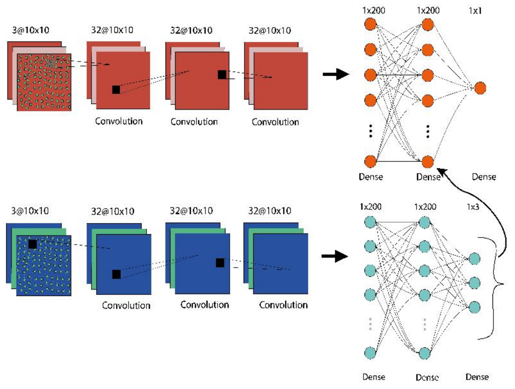

Fig. 2. Actor (bottom) and critic (top) network architectures. Taken from our previous work in [5]

D. Problem formulation

In this section, we define our problem formulation which applies to both learning algorithms. Our state-space is a high dimensional continuous vector space. Thus, our actor and critic networks begin with convolutional layers in their first few layers. This enables us to reduce the higher-dimensional state space to a low-dimensional representation which is useful for combining the actions with the intermediate layers in the critic.

1) State

The state-space of swarm of robots is represented as a combination of three matrices, where each matrix is composed of x, y and θ values of each robot in the swarm. This state-space input can be interpreted as an image with three channels, but with pixel values replaced with pose and orientation of each robot. In our case, the first channel is *x*-, second channel is *y*-, and the last channel is θ values of each robot. We implemented swarm population case $N = 100$ robots. Our states are normalised in channel-wise relation as described in section II section.

2) Action

We designed our action space with two different bounds for DDPG and TRPO models. *Collective* action set involves one set of parameters that is being shared across the swarm in both models. Hence, our action space is composed of 3 output units, corresponding to $\{k, \beta, \alpha\}$ parameters for N robots. Since the linear and rotational velocities of CM cannot be tuned explicitly, as they are the result of the dot product between heading and force vectors, we input $\{k, \beta, \alpha\}$ parameters into models, which in turn tunes the heading and force vectors. The last layer of the actor network is passed through a `sigmoid` layer to ensure symmetry in actions, and to also ensure that the CM parameters never become negative. Actions are then multiplied by the action bound vector associated with each model to ensure that the values are within an acceptable range for fast rate of convergence. Our action space scaling vector is given as $\{1, 4, 0.1\}$ for DDPG, and as $\{2, 5, 0.1\}$ for TRPO.

3) Reward

The reward function in our problem is the heart of the multi-agent learning and ensures that the swarm converges. The two main elements of the reward design is composed of the degree of alignment and the global force for the entire swarm, which is obtained by summing over the individual forces for each robot in the swarm, given in Eq. (3). The degree of alignment needs to be maximised, and the global force is to be minimised. Hence, we formulated a reward function that is the negated global force function with a weighted sum of the degree of alignment and the derivative of force term. We then maximise this reward to ensure: (i) cohesive motion by means of minimising the global force, (ii) ensuring polarisation order converges, (iii) Maximising the rate at which (i) and (ii) happen by means of incorporating swarm order and force derivative term, For DDPG, our reward is

$$R^*(s, a) = -\omega_f F_g + \omega_\psi \psi + \omega_{deriv} \nabla F_g \tag{24}$$

where F_g denotes the global force obtained as $F_g = \frac{1}{N} \| \sum_{i=1}^{N} \mathbf{f_i} \|$, $\psi = \sum_{n=1}^{N} \exp i\theta_n$ denotes the swarm order, ∇F_g denotes the derivative of the force w.r.t. time, and their respective weights. For TRPO, derivative term in reward is replaced by $\omega_a \frac{1}{N} \sum_{k=1}^{A} a_k$, which is the mean of the actions incorporated in the reward to prevent the actions from diminishing to zero during training. The weightings are given as $[\omega_f = -10, \omega_\psi = 3, \omega_{deriv} = 0.5]$, and $[\omega_f = -3, \omega_\psi = 0.1, \omega_a = 0.2]$ for DDPG amd TRPO models respectively.

4) Network Architectures

As shown in Figure 1, the first layers of our actor and critic networks were composed of convolutional layers, output of which were flattened out and fed as input into the three fully connected layers. For both networks, the first fully connected layer was composed of 200 neurons, second layer of 200 neurons, and third layer of a single output for the critic, which is the estimated Q-value, and 3 outputs for the actor. The first five layers had ReLU as activation function, with the sixth layer having a linear activation function. We used a filter size of 3 for the first CONV layer and 2 for the following CONV layers. The final action outputs were then scaled with their respective values.

5) Weight Initialisation

For both the actor and critic networks, the weight initialisation for the first 5 layers were drawn from a uniform distribution with standard deviation of $1/\sqrt{fan_{in}}$ and mean of zero where fan_{in} is the number of input neurons from the previous layer. The last layer in critic was drawn from uniform distribution with standard deviation of $3e^{-4}$ and mean zero. The last layer in actor was drawn from uniform distribution with standard deviation of $3e^{-3}$ and zero mean.

6) Optimisation & Learning Schedule

We used momentum optimiser with SGD for both networks, where at each time step of an episode both the actor and critic network weights. We trained DDPG model for 100 episodes, where each episode had a duration of 1000 episode-length, and TRPO model for 270 episodes. We used a learning rate of 0.001 and a momentum of 0.3 in both actor and critic networks unless stated otherwise. Our minibatch size was 32. We also used batch normalisation in training as the original DDPG

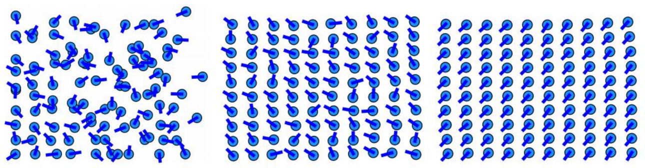

Fig. 3. DDPG (blue) model at inference: (a) t=0s, (b) t=500s, (c) t=1000s

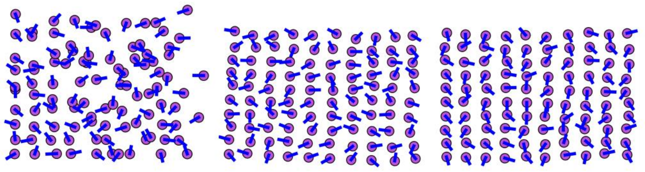

Fig. 4. TRPO (purple) model at inference: (a) t=0s, (b) t=500s, (c) t=1000s

noted that it helps speeding up the training. We used learning rates of 0.001 and momentum of 0.5 for both actor and critic networks in DDPG. Discount factor is 0.99 and soft target weight is given as 0.001. For TRPO, we used a discount factor of 0.995, $\tau = 0.97$, l2-regularization of 0.001 and maximum KL-divergence of 0.01. Our batch-size is reported to be 512 for TRPO.

IV. Results & Discussion

We investigated experiments involving swarm populations of $N = 100$ agents, where each agent is assumed to have access to global state matrix for their local neighbourhoods. Figure 5 shows that both models converge to optimal flocking behaviours and hence learn useful flocking policies effectively. Comparing the two methods, TRPO in figures (a) and (c) takes longer to converge at around 100 iterations to converge whereas DDPG converges around 25 iterations. DDPG is known to be more sample efficient than TRPO. Also notice that the peak overshoot in force plot of TRPO is much higher than that in DDPG as well as the overall curve being much more oscillatory. This could be explained with the fact that DDPG is off-policy and TRPO is on-policy, hence it would lead to higher variance than DDPG, thus, leading to much more oscillatory learning experience. However, TRPO has the benefit of monotonic improvement in policy at each iteration and has lower bias than off-policy methods, such as DDPG. During the tuning process, we encountered that DDPG is very sensitive to hyperparameter changes and random seeds in the experimental setup. This would be sensible as off-policy methods are prone to higher bias than on-policy methods and as a result lead to divergence in some random seeds and large changes in hyperparameter space. We also found that TRPO leads to better exploration strategy in terms of action values suggested by the model, and DDPG was more prone to exploitation of actions. Furthermore, the weightings given in reward design in previous section rely on the optimal values being chosen for effective learning. Increasing one of the weights in relation to others would mean the one being more emphasised and as a result this led to explosion. Our results

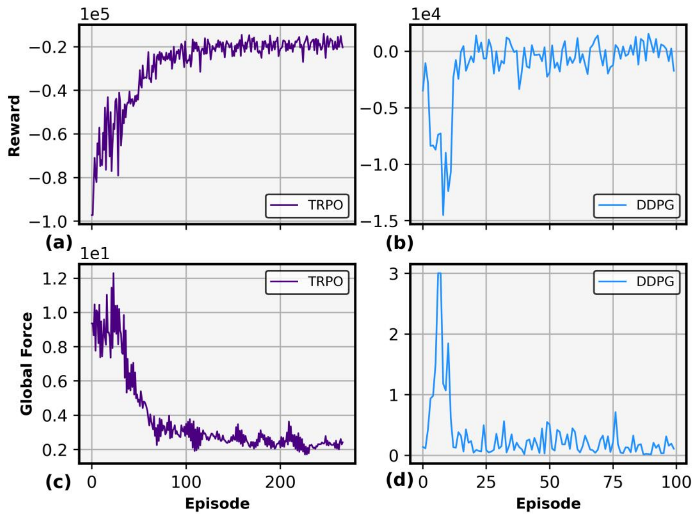

Fig. 5. (a) Reward for TRPO during training over 260 episodes, (b) Reward for DDPG during training over 100 episodes, (c) Convergence of global force, hence, emergence of cohesive collective motion for TRPO during training over 260 episodes, (d) Convergence of global force for DDPG during 100 episodes. Purple represents TRPO, and blue represents DDPG.

are also shown qualitatively at inference experiments, where both models manage to learn to flock, with DDPG resulting in more rapid convergence compared to TRPO. Note that the robots in inference are initialised at the same random positions with Gaussian noise. This initial state is not present in the training dataset. At $t = 0s$, both methods have same initial positions. After 100s of iterations, both models seem to have approximately equal cohesiveness in swarm. At the end of t=1000s, we see that DDPG is converged to optimal policy but TRPO is stuck at a local minima, as the difference between $t = 500s$ and $t = 1000s$ are not very different qualitatively. This shows the generalisation capabilities and the robustness of both methods, but DDPG with a much better convergence.

V. Conclusion

In this paper, we investigated the application two policy gradient methods where one directly parameterised the policy space on (TRPO) and one that fitted a value function to learn the optimal policy. We conclude that our hypothesis that both models are capable of learning emergent flocking behaviours is satisfied as illustrated in figures qualitatively and quantitatively. We overcame the problem of sparse rewards and non-stationarity of the environments in multi-agent studies by composing our reward as if the agent was a single entity and treating the collective mechanism as an agent. By doing so, we can train multi-agent systems to collectively emerge flocking behaviours under any random initial starting positions.

References

[1] M. Schranz, G. A. Di Caro, T. Schmickl, W. Elmenreich, F. Arvin, A. Şekercioğlu, and M. Sende, "Swarm intelligence and cyber-physical systems: Concepts, challenges and future trends," *Swarm and Evolutionary Computation*, vol. 60, p. 100762, 2021.

[2] M. Hüttenrauch, A. Šošić, and G. Neumann, "Guided deep reinforcement learning for swarm systems," *arXiv preprint arXiv:1709.06011*, 2017.

[3] M. Hüttenrauch, S. Adrian, G. Neumann *et al.*, "Deep reinforcement learning for swarm systems," *Journal of Machine Learning Research*, vol. 20, no. 54, pp. 1–31, 2019.

[4] R. Pinsler, M. Maag, O. Arenz, and G. Neumann, "Inverse reinforcement learning of bird flocking behavior," in *ICRA Swarms Workshop*, 2018.

[5] M. B. Bezcioglu, B. Lennox, and F. Arvin, "Self-organised swarm flocking with deep reinforcement learning," in *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. IEEE, 2021, pp. 226–230.

[6] E. Ferrante, A. E. Turgut, M. Dorigo, and C. Huepe, "Collective motion dynamics of active solids and active crystals," *New Journal of Physics*, vol. 15, no. 9, p. 095011, 2013.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.