Apprenticeship Learning for Continuous State Spaces and Actions in a Swarm-Guidance Shepherding Task

Hung The Nguyen*, Matthew Garratt*, Lam Thu Bui†, Hussein Abbass*

*School of Engineering and Information Technology, UNSW-Canberra, University of New South Wales
Canberra, Australia
Email: hung.nguyen@student.adfa.edu.au; m.garratt@adfa.edu.au; h.abbass@adfa.edu.au
†Department of Information Technology, Le Quy Don Technical University Hanoi, Vietnam
Email: lam.bui07@gmail.com

***Abstract*— Apprenticeship learning (AL) is a learning scheme using demonstrations collected from human operators. Apprenticeship learning via inverse reinforcement learning (AL via IRL) has been used as one of the primary candidate approaches to obtain a near optimal policy that is as good as that of the human policy. The algorithm works by attempting to recover and approximate the human reward function from the demonstrations. This approach assists in overcoming limitations such as the sensitivity associated with the variance in the quality of human data and the short sighted decision time that does not consider future states. However, addressing the problem of continuous action and state spaces has still been challenging in the AL via IRL algorithm. In this paper, we propose a new AL via IRL approach that is able to work with continuous action and state spaces. Our approach is used to train an artificial intelligence (AI) agent acting as a shepherd of artificial sheep-inspired swarm agents in a complex and dynamic environment. The results show that the performance of our approach is as good as that of the human operator, and particularly, the agent's movements seem to be smoother and more effective.**

***Index Terms*—Apprenticeship learning, Inverse reinforcement learning, Swarm Guidance, Shepherding.**

I. Introduction

The problem of controlling bio-inspired swarms has generated a significant amount of novel research in the fields of both control theory and intelligent computation [1]. Bio-inspired approaches are particularly valuable when developing multiple agent systems or robotic swarms. In these robotic swarms, studies focus on addressing the question of how to collect or guide groups of individual agents through natural interactions between agents and the environment, and between agents [2]–[4]. Challenges in addressing this question include how to design flexible geometric constraints among agents, and then maintain them to successfully obtain a defined formation or communication network connections while performing a specific mission [5]–[8].

As described in [9], [10], there are two major approaches in controlling a group of individual agents or swarm: manually programming prescribed rules and learning-based algorithms. The first approach defines a set of rules or equations to compute the behavior of such multiple agent systems through global states or local groups of single agents in the entire swarm [11]–[13]. Although the rule-based approach normally gets designed and implemented fast, and scale up as the number of agents in the swarm increases, it often lacks adaptive and generative abilities when faced with novel scenarios. For the machine learning approach, the need to deeply understand the dynamic model of the swarm is unnecessary; therefore, this type of models provides flexibility. Additionally, the learning-based approach enables agents to adjust within different contexts. Various learning-based methods using reinforcement learning [14], [15] or deep reinforcement learning [16], [17] have been proposed to find out decentralized policies for swarm control. However, these methods do not scale up when the number of individual agents in the swarm increases. In fact, training a large number of agents simultaneously will need significant computational resources.

One possible solution to the scalability problem is to use a shepherd as an agent that influences either the swarm as a whole, or a subset of the swarm. The shepherding problem is inspired by sheepdogs herding sheep in agriculture. In a recent effort of Strömbom et al. [18], the problem is modeled heuristically to explain how one intelligent agent interacts with a group of other agents acting as fixed rules entities. It can be seen that this heuristic model is a promising answer for addressing the lack of scalibility in the learning-based approaches when just one learning individual might influence or control a large group of rule-based agents. The shepherding problem is valuable in the context of human-swarm interaction (HSI) [19] due to the fact that it is simpler for the human operator when he or she just interacts with one intelligent agent to monitor the entire swarm. However, the approach raises another challenge, that the shepherd needs to look after a more complex context.

Therefore, according to the learning-based approach, the problem of swarm control will simply focus on training one smart agent to control a group of others. To train the smart agent in these swarm systems, the literature as described in a recent survey [20] makes a mention of three possible methods: supervised learning (SL), reinforcement learning, and inverse reinforcement learning. The former shows challenges to effectively develop intelligent or autonomous agents when the collected data is sup-optimal because of human factors like control skills or workload. Besides, the data might be insufficient when it is difficult to expect from the human operator to repeat the task for a considerable number of times in order to

have a large enough data-set. Finally, supervised learning just maps an active action with a given state without considering the future states. In multiple-agent systems or swarms, the problem of considering the future states is important because actions in some steps might not be optimal despite that its immediate effects could seem to be the best achievable in the short run.

The second method is reinforcement learning (RL) [21], which is able to address the challenges of supervised learning techniques. RL is an interactive approach in which an agent learns by interacting with the environment through a number of trials and errors. In its interactions, the agent can receive a *reward* or punishment, which is predefined by the expert of the task. This method helps an agent to explore a sufficient state space, and gain an optimal or a reasonably high cumulative total reward. In classic reinforcement learning approaches like Q-learning or SARSA [21], the problem of continuous state and action spaces for swarm control is challenging. Recently, with significant advances in deep learning, deep reinforcement learning agents surpass human experts in various tasks such as playing Atari games [22] or multi-task control of autonomous robots [23]. An increasing number of autonomous systems using deep reinforcement learning methods are being developed in order to address various practical applications. However, when applying deep reinforcement learning methods to address swarm control as in the shepherding problem, it has still been a considerable challenge because of the complexity of this task and the dynamic nature of the environment. The challenge comes from the question of how to successfully design a reward function in order to guide the searching process towards optimal policies.

In contrast to the RL approach, inverse reinforcement learning (IRL) [20] allows a reward function to be learned from human-generated policies instead of manually designing reward functions. In terms of the IRL approach, an underlying assumption is that the humans creating the demonstrations are experts; thus, the resultant policies are somehow optimal. The machine-induced reward function is then applied to an RL algorithm in order to obtain a near-optimal policy being closest to the expert's policy. By using human demonstrations, Abbeel et al. [24] proposed the apprenticeship learning via inverse reinforcement learning (AL via IRL) to obtain an approximate policy that is close enough to the observed one. The advantage of this algorithm is to necessarily avoid the need to recover the human reward function explicitly. Instead, the reward of each state is represented by the sum of its linearly weighted features.

In this paper, we propose a hybrid AL via IRL approach which addresses the problem of continuous action and state spaces in the shepherding problem. Our proposed approach applies K-mean [25] and Radial basis function (Rbf) [21] for handling the problem of continuous state space, and Deep Deterministic Policy Gradients (DDPG) [26] to address the challenge of continuous action spaces.

The organization of this paper is as follows. Section II formulates the shepherding problem. Section III covers the background on reinforcement learning, deep reinforcement learning, and apprenticeship learning via inverse reinforcement learning. Section IV focuses on explaining our research methodology. Section V introduces our simulation environment, experimental setup, and the set of metrics we use to assess the performance of our approach. We then discuss our results and corresponding analysis in Section VI and conclude the paper in Section VII.

II. The Shepherding Problem: A Formulation for Effective Swarm Control

The shepherding problem is known as a promising approach of one agent driving another group of, sometimes willing while in some other situations unwilling, agents in the same direction towards a goal. This problem can be applied for numerous applications such as civil crowd control when planning exit requirements for public events, oil spill recovery, sheep herding for agricultural purposes, helping airplanes avoid birds in transportation safety, interactive games in entertainment, and gathering or driving an exploring swarm [18], [19].

In this paper, we describe the Strömbom et al. [18] model by presenting the notations and mechanism that are utilized later in our experimental design. The shepherding environment is a squared paddock with length of L. There are two types of agents initialized in this environment belonging to a swarm of sheep (influenced agents) $\Pi = \{\pi_1, \ldots, \pi_i, \ldots, \pi_N\}$, or a number of shepherds (influencing agents) $B = \{\beta_1, \ldots, \beta_j, \ldots, \beta_M\}$. Primary behaviours of shepherds and sheep are described as follows:

a) Behaviours of Shepherds: The shepherd has three behaviours: driving and collecting, at a time step t. Its driving behaviour is triggered when all the sheep have been collected in a cluster. Equation 1 represents the threshold to know if the sheep are completely clustered or not. If all distances of the sheep to their center of mass is lower than this threshold, the shepherd moves to a driving point located behind the cluster to dive the sheep.

$$f(N) = R_{\pi\pi} N^{\frac{2}{3}} \tag{1}$$

The collecting behaviour will be triggered when there is any outer sheep whose distance to the center of sheep mass is greater than the threshold $f(N)$. The shepherd moves towards a collecting point located behind this sheep to force it towards the mass to form the cluster.

b) Behaviours of Sheep: Each sheep at a time step t includes four behaviours: escaping, collision avoidance, grouping, and jittering. The escaping behaviour represents a repulsive force $F^t_{\pi_i\beta_j}$ if the distance between sheep π_i at position $P^t_{\pi_i}$ and shepherd β_j at position $P^t_{\beta_j}$ is less than $R_{\pi\beta}$; that is,

$$\|P^t_{\pi_i} - P^t_{\beta_j}\| \leq R_{\pi\beta} \tag{2}$$

The second represents the repulsion of sheep π_i from other sheep $\pi_{k\neq i}$. The condition for the existence of a force between

a pair of sheep is that the distance between them is less than $R_{\pi\pi}$; that is,

$$\exists k,\ such\ that\ \|P^t_{\pi_i} - P^t_{\pi_k}\| \leq R_{\pi\pi} \tag{3}$$

We then denote $F^t_{\pi_i\pi_{-i}}$ to represent the summed force vectors from all other sheep within the threshold distance applied onto sheep π_i.

For the grouping behaviour, it represents the attraction of the sheep to the center of sheep mass of its neighbors $\Lambda^t_{\pi_i}$ by a force $F^t_{\pi_i\Lambda^t_{\pi_i}}$. Lastly, the jittering behaviour aims to avoid moving impasse so a random noise, which is presented $F^t_{\pi_i\epsilon}$ with weight $W_{e\pi_i}$, is summed into the total force.

Sheep π_i total force behaviour σ_9 is represented by the total force $F^t_{\pi_i}$ which is a weighted sum of force vectors $F^t_{\pi_i\beta_j}$, $F^t_{\pi_i\pi_{-i}}$, $F^t_{\pi_i\Lambda^t_{\pi_i}}$, and $F^t_{\pi_i\epsilon}$; that is,

$$F^t_{\pi_i} = W_{\pi_v}F^{t-1}_{\pi_i} + W_{\pi\Lambda}F^t_{\pi_i\Lambda^t_{\pi_i}} + W_{\pi\beta}F^t_{\pi_i\beta_j} + W_{\pi\pi}F^t_{\pi_i\pi_{-i}} + W_{e\pi_i}F^t_{\pi_i\epsilon} \tag{4}$$

The positions of shepherds and sheep are updated according to Equations 5 and 6 given $S^t_{\beta_j}$ and $S^t_{\pi_i}$ be the speed of β_j and the speed of π_i at time t. However, in original Strömbom model, the speeds of agents are constant.

$$P^{t+1}_{\beta_j} = P^t_{\beta_j} + S^t_{\beta_j}F^t_{\beta_j} \tag{5}$$

$$P^{t+1}_{\pi_i} = P^t_{\pi_i} + S^t_{\pi_i}F^t_{\pi_i} \tag{6}$$

In this paper, the shepherding environment includes one shepherd and a swarm of sheep, and the underlying assumption is that the shepherd is aware of all the sheep.

III. Related Work

A. Reinforcement Learning/RL

A Markov decision process (MDP) [21] represents a sequential decision-making problem for which an agent must select sequential actions to maximize optimization criterion based reward values with the finite MDP formally presented as a tuple $\Xi = \{S, A, T, w, \gamma\}$ where:

- $S = \{s_1, s_2, ..., s_n\}$ is a finite set of n states describing the dynamic world.
- $A = \{a_1, a_2, ..., a_m\}$ a set of m actions executed by the agent.
- $Trans : S \times A \times S \mapsto [0, 1]$ a transition function or transition model, where $Trans(s, a, s')$ is the transition probability when the agent takes action ($a \in A$) in a state ($s \in S$) which leads to the next state $s' \in S$, i.e, $Trans(s, a, s') = \rho(s'|s, a)$.
- $w : S \times A \mapsto \mathbb{W}$ a reward function, where $w(s, a)$ is the immediate reward received by the agent when it takes action ($a \in A$) in state ($s \in S$).
- $\gamma \in [0, 1]$ a discount parameter.

A control policy is a mapping ($\tau : S \times A \to [0, 1]$), where $\tau(s, a)$ is the probability of selecting action ($a \in A$) in state($s \in S$). When the agent follows a τ policy, the value function V^τ and the action-value function Q^τ are defined respectively, by:

$$V^\tau(s) = \mathbb{E}_\tau\left[\sum_{i=0}^{\infty}\gamma^t w(s_t, a_t)|s_0 = s\right]. \tag{7}$$

$$Q^\tau(s, a) = \mathbb{E}_\tau\left[\sum_{i=0}^{\infty}\gamma^t w(s_t, a_t)|s_0 = s, a_0 = a\right] \tag{8}$$

where (s_t, a_t) is a state-action pair generated under policy τ in time step t, the value function $V^\tau : S \to \mathbb{W}$, and action-value one $Q^\tau : S \times A \to \mathbb{W}$.

The aim of the agent is to discover an optimal control policy (τ^*) which maximises the total discounted reward accumulated over all states. The objective of learning is to maximize the accumulated return over time:

$$W_t = \sum_{t'=t}^{T}\gamma^{t'-t}w_{t'} \tag{9}$$

where T is the total number of time steps, and γ is the discount factor.

The optimal value function denotes $V^*(s)$, and $V^*(s) = sup_\tau V^\tau(s)$. Similarly, the optimal action-value function denotes $Q^*(s, a)$, and $Q^*(s, a) = sup_\tau Q^\tau(s, a)$ with τ is called an optimal policy in the MDP framework if and only if:

$$\pi(s) \in arg\max_{a\in A} Q^\tau(s, a), \forall s \in S. \tag{10}$$

Based on the Bellman equations [8], $V^*(s)$ and $Q^*(s, a)$ are defined. For all $a \in A$ and $s \in S$, respectively, as:

$$V^*(s) = \max_{a\in A} Q^*(s, a). \tag{11}$$

$$Q^*(s, a) = w(s, a) + \gamma\sum_{s'\in S}\rho(s'|s, a)V^*(s'). \tag{12}$$

B. Deep Q-learning/DQN

In real-world scenarios, Q-Learning [21] is impractical because of its limited state and action spaces, and inability to be generalized to unseen states. Therefore, in practical Q-learning systems, function approximators, generally linear ones, are used to approximate the Q-function as:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{13}$$

However, these functions are limited to manually crafted features and low-dimensional data. Recently, deep neural networks (DNNs) [27] have been shown to be capable of solving raw and high dimensional data in real-word scenarios. These advances lead to a breakthrough of deep Q-learning [28] when it uses DNNs to effectively approximate representations over a continuous and large state space in the context of reinforcement learning. It is appealing to apply an RL paradigm for a real world environment. This algorithm has been a game-changer that surpassed top human-level performance in various tasks [28]. The main core of DQN is that the authors used an experience replay approach while training DNNs.

In this technique, experiences are stored in a sufficiently large memory over episodes and then a certain number, called

a mini-batch, gets randomly chosen from the memory, and updated to the DNN in every step. The advantages of this technique enable the learning to be stable and allows for converge. These advantages have been analyzed in [28]. The algorithm assumes a discrete action space. For continuous action spaces, a family of deep deterministic policy gradients (DDPG) algorithms is introduced below.

C. Deep Deterministic Policy Gradients/DDPG

Deep deterministic policy gradients [26] address the limitations of the Deep Q-learning approaches by producing continuous actions. DDPG has an actor-critic architecture including two separate networks: the first known as the actor network (with weights θ^{π}) is used to approximate the output action; and the latter known as the critic network (with weights θ^{Q}) is used to estimate the Q-value of the action produced by the actor network. Both have their own target networks (with weights $\theta^{\pi'}$ and $\theta^{Q'}$ respectively) to maintain the stability in learning. The loss function of the critic network is represented as follows:

$$L(\theta^{Q}) = \left(Q(s_t, a_t|\theta^{Q}) - y_t\right)^2 \tag{14}$$

where

$$y_t = w_t + \gamma Q(s_{t+1}, \tau(s_{t+1}|\theta^{\tau\prime})|\theta^{Q\prime}) \tag{15}$$

The weights of the deep critic network can be updated by

$$\theta^{Q} \leftarrow \theta^{Q} - \alpha_Q \nabla_{\theta^{Q}} L(\theta^{Q}) \tag{16}$$

where α_Q is the learning rate of the critic network. The actor network is then updated by using the action gradient estimated with the critic network.

$$\theta^{\tau} \leftarrow \theta^{\tau} - \alpha_\tau \nabla_a Q(s_t, \tau(s_t|\theta^{\tau})|\theta^{Q}) \nabla_{\theta^{\tau}} \tau(s_t|\theta^{\tau}) \tag{17}$$

where α_τ is the learning rate of the actor network. Experience replay can also be used with DDPG for reducing bias introduced from temporally correlated transitions data and more learning efficiency.

D. Apprenticeship learning via inverse reinforcement learning/AL via IRL

In the reinforcement learning approach, the RL agent needs to discover an optimal control policy which maximizes a reward-based criterion. However, in real-world scenarios, defining the reward function is challenging. Inverse Reinforcement Learning (IRL) [20] recovers/approximates a reward function by learning it from human demonstrations.

ABS via IRL [24] assumes that the reward function, $W(s)$, is a linear function represented as a weighted, θ, sum of a state feature vector $\phi(s) = \{\phi(s_1), \ldots, \phi(s_i), \ldots, \phi(s_K)\}$:

$$R(s) = \theta^T \cdot \phi(s) = \sum_{k=1}^{K} \theta_k \phi_k(s), \forall s \in S \tag{18}$$

In terms of the IRL approach, the underlying assumption is that the human expert is following an optimal policy; thus, there is an optimal weight vector θ^* such that the optimal reward $W^*(s)$ is

$$W^*(s) = \theta^* \cdot \phi(s), \forall s \in S \tag{19}$$

AL via IRL uses human demonstrations to then find the weight vector θ that approximates θ^*. For a fixed policy, its value is defined as follows:

$$V^{\tau}(s_0) = \theta \cdot E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \tau\right] \tag{20}$$

where γ is the discount factor.

Meanwhile, the feature expectations vector under a policy π is defined as:

$$\mu(\tau) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \tau\right] \tag{21}$$

If the dynamic model is unknown, the feature expectations vector $\mu(\tau)$ can be seen in Equation 22:

$$\mu(\tau) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{22}$$

where m is the number of trajectories. When the feature mapping function, ϕ, and demonstrations are given, a reinforcement learning algorithm is used to discover $\tilde{\tau}$ in order to make $\mu(\tilde{\tau})$ as close as possible to $\mu(\tau_E)$ with τ_E being the policy of the human expert.

In the next section, we introduce a novel hybrid AL via IRL approach that produces an effective feature mapping function to address the problem of continuous state space, and uses the deep deterministic policy gradients (DDPG) in the RL step in order to output continuous actions.

IV. A Novel Hybrid Apprenticeship Learning via Inverse Reinforcement Learning Approach

In the original apprenticeship learning via inverse reinforcement learning method [24], the reward of a state is represented as a sum of its linearly weighted features as described in Equation 18. Therefore, the learning success of the AL via IRL depends significantly on the feature mapping function, ϕ. Particularly, in a complex and dynamic environment such as that present in swarm control, the state space is continuous so that effectively designing the the feature mapping function is not simple. Additionally, the problem of the continuous action space is not mentioned in the original algorithm yet this is highly necessary and practical in the context of swarm control or the shepherding problem. In this paper, we propose a novel approach aiming to address these challenges in the original AL via IRL algorithm.

A. The Problem of Continuous State Space

In term of the IRL approach, the reward function is expressible as a linear sum of known features. Hence, the feature mapping function is needed to transfer a state into a vector or array of values between 0 and 1. In simple contexts where the state space are discrete, this space can directly be transferred

into a vector of binary features of 0 or 1. Regarding the problem of the continuous state space, these binary features are not appropriate when the feature can just represent on or off.

To handle the problem of the continuous state space, a radial basis function (RBF) [21] is used as a feature mapping function, ϕ, to transfer a state into an array of real values $\in [0, 1]$. These real values indicate the range in which a feature is turning on or off, and then the state representation is more sufficient and closer to the visited state space of the expert. As a result, the recovered reward function will be represented precisely so that the obtained policy is closest to that of the expert. Equation 23 shows an RBF feature of the Gaussian type in which the value of the RBF feature replies on the distance between a given state, s, and a center, c_i, representing the feature i.

$$\phi(s) = e^{-\frac{||s-c_i||^2}{2\sigma_i^2}} \tag{23}$$

where σ is a constant parameter.

In Equation 23, it can be seen that the centers are needed to be predefined. The problem of how to find these centers appropriately is necessary and important so that the RBF function is effectively able to transfer the state space into suitable real-value vectors. In this paper, we use the K-mean algorithm [25] to cluster the state space into a number of centers that are able to sufficiently represent the entire state space. This approach enables us to automatically find the centers in the state space instead of defining them manually. The combination between the RBF function and K-mean creates a robust feature mapping function in our proposed AL via IRL approach when it can transfer the state space into a vector of real values being more precise and closer to the visited state space of the human.

B. The Problem of Continuous Action Space

Recently, with significant advances in deep reinforcement learning [22], it is more effective when applying deep reinforcement learning algorithms for the RL step of the IRL approaches like AL via IRL. In the shepherding problem, continuous actions are required in order to achieve effective control policies. Deep deterministic policy gradients (DDPGs) introduced in [26] address this problem by producing continuous actions.

In this paper, the DDPGs algorithm is used as an RL step in which the input is the continuous state space, and the output is a real value representing continuous actions. Our proposed AL via IRL algorithm is shown in Algorithm 1. The main contributions center around using the proposed feature mapping function and the DDPGs algorithm in the RL step. Our contributions are shown in italics, while the remainder of the algorithm is identical to the one presented in [24].

Algorithm 1 Our Apprenticeship Learning via Inverse Reinforcement Learning Algorithm For Continuous State and Action Spaces; an improvement version of the algorithm presented in [24] with the additional contributions shown in italics.

Input: : *A RBF feature mapping function* ϕ, m human trajectories, $T = \{T_1, T_2, ..., T_m\}$ where $T_i = \{(s_1, a_1), (s_2, a_2), ..., (s_n, a_n)\}$ in which $S = \{s_1, ..., s_{n\times m}\}$ and $A = \{a_1, ..., a_{n\times m}\}$ are continuous state and action spaces, respectively, and the stop threshold ϵ.

Output: : A number of policies $\{\tau^{(i)} : i = 0..n\}$.

Randomly choose policy $\tau^{(0)}$, estimate $\mu^{(0)} = \mu(\tau^{(0)})$ via Monte Carlo, and set $i = 1$.
Initializing the DDPGs parameters [26].
Calculating C centers by K-mean algorithm [25].
Calculating the total expert feature expectations vector, μ_E as described in Equation 22.
Set $\theta^{(1)} = \mu_E - \mu^{(0)}$ and $\bar{\mu}^{(0)} = \mu^{(0)}$.
Set $t^{(1)} = \|\mu_E - \mu^{(0)}\|_2$.
if $t^{(i)} \leq \epsilon$ **then**
    terminate
end if
while $t^{(i)} > \epsilon$ **do**
    Using DDPG producing continuous actions to compute the optimal policy $\tau^{(i)}$ with $W = (\theta^{(i)})^T \phi$.
    Compute $\mu^{(i)} = \mu(\tau^{(i)})$ and set $i = i + 1$.
    Set $a = \mu^{(i-1)} - \bar{\mu}^{(i-2)}$.
    Set $b = \mu_E - \bar{\mu}^{(i-2)}$.
    Set $\mu^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{x^T y}{x^T x} x$.
    Set $\theta^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$.
    Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$.
end while

V. Experiments

In this section, we describe the designed shepherding environment, experimental setups along with their parameters; and finally the evaluation metrics and the parameters of training the deep neural network are clarified.

A. The Shepherding Environment

The shepherding simulation environment is designed using the Python language in Ubuntu 18.04. The size of the shepherding environment is 150×150 being identical to that of Strömbom et al. [18]. Similarly, the parameters of shepherding problem are no different from that of Strömbom except the number of sheep (10 sheep in this paper). Initially, our aim is to evaluate the proposed approach being able to produce an agent acting as well as a human in a not too complex environment. Thus, the set number of sheep is not significant in order to reduce the complexity of the task, and then the human demonstrations collected can generally avoid unexpected noise in the underlying policy followed by the human resultant from sub-optimal actions. This allows us to have a more exact view of the learning success of our proposed approach.

In each episode, the positions of the shepherd and sheep are randomly initialized within a constant area located in the top right corner of the environment. While a target is set at the bottom left corner, the shepherd is required to herd the entire herd of sheep towards the target.

B. Experimental Setups

In this paper, we conduct three experimental setups: human demonstrations, supervised learning using a deep neural network (DNN), and our proposed AL via IRL algorithm (Hybrid AL via IRL), as shown in Table I.

Table I: The Experimental Setups

Experiment ID	Description
Human	A human operator controls the shepherd to gather a data-set
DNN	Training a deep neural network by the human data-set
Hybrid AL-IRL	Our proposed algorithm

Regarding the human demonstrations, the human subject was asked to control the shepherd to herd sheep towards the target for 20 episodes. The set of the human demonstrations include 4352 samples. For our proposed approach - Hybrid AL-IRL, the stop threshold ϵ is set to 2, the set number of centers, K, produced from the K-mean algorithm is 100, and σ in Equation 23 is calculated in [29].

C. Evaluation Metrics

Due to the dynamics of the environment and initialization of training scenarios, we employ the ratio $\frac{total-reward}{total-number-of-steps}$ in each episode to see if the learned policy is optimized or not. The $total-reward$ is a sum of all individual rewards of each step in the episode. The individual reward is calculated in Algorithm 2. The agent is presented with a small positive reward if the sheep global centre of mass moves closer to the target and a small negative reward if this point is further from the target. One episode of training is a success when the target is reached. We also visualize the footprint of the shepherd and the sheep center of mass to show the policy exploitation.

Algorithm 2 Reward Function

Input: Sheep Global Centre of Mass (GCM) $\Gamma^t_{\pi_i}$; Position of Target P^G; Task Success Radius R_{TS};
Output: reward value w^t

$w^t_1 = 0$; $w^t_2 = 0$;
Calculate distance $d^t_{P^G-GCM} = ||P^G - \Gamma^t_{\pi_i}||$
if $d^t_{P^G-GCM} < d^{t-1}_{P^G-GCM}$ **then**
    w^t_1 = 0.1
else
    w^t_1 = -0.1
end if
if $d^t_{P^G-GCM} < R_{TS}$ **then**
    w^t_2 = 10
end if
$w^t = w^t_1 + w^t_2$

D. Training Deep Neural Network

In the two experimental setups for DNN and Hybrid AL-IRL, we use an identical deep neural network (DNN) consisting of an input layer, two fully-connected hidden layers, and a fully-connected output layer. The number of nodes of the two hidden layers are 32 and 16, respectively. Tensorflow and Keras libraries [30] were used to design the network. The optimizer method is the stochastic gradient descent. The DDPG algorithm is trained with a replay memory size of 100,000 state-action pairs, the discount factor $\gamma = 0.99$, mini-batch size 64, and the learning rate 0.0001. During training, the $\epsilon-greedy$ is decreased linearly from 1 to 0. The epsilon decay is 0.995 for every episode. The maximum number of steps is 1000. The deep network was trained on an NVIDIA GeForce GTX 1080 GPU. Figure 1 illustrates the DNN structure used in this paper.

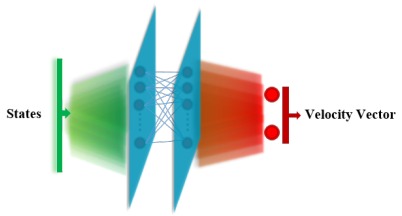


Figure 1: The Illustration of Deep Neural Network Structure

The states space presented in Table II are used as inputs, while the outputs are a vector of two continuous values of the shepherd.

Table II: The States Space

State ID	State Name	State Description
1-2	$V_{GCM_D}(x, y)$	Vector from GCM to shepherd
3-4	$V_{FS_D}(x, y)$	Vector from the furthest sheep to shepherd
5-6	V_{T_GCM}	Vector from target to GCM

VI. Results and Discussion

In this paper, we conducted three experimental setups as described in Table I. In this section, we provide the learning curves and explanation in training of the later two setups, and then compare the performance among them in various testing cases.

A. Training

a) DNN Setup: We fully train the deep neural network by the human data-set. Figure 2 shows the loss in training. It is clear that the training loss decreases significantly in the first 4000 epochs, and then maintains stable performance at approximately 0.25 in the remaining epochs.

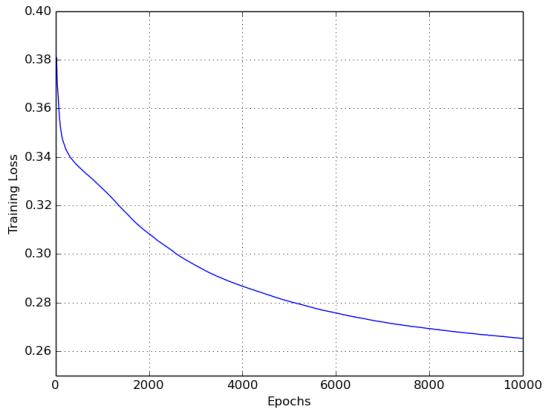


Figure 2: Training loss of the deep neural network.

b) Hybrid AL via IRL Setup: We use the same human data-set used in the DNN setup to train our proposed AL via IRL approach. To evaluate the learning success of the AL via IRL approach, it is worth to illustrate the distance - t between the feature expectation of the training agent and that of human. Figure 3 shows this distance in training. It can be seen that the Euclidean distance between these two feature expectations decreases significantly from more than 40 to 5 in the first 5 inverse reinforcement learning iterations, and then maintains stable values at just slightly over two units of distance in the remaining iterations. This trend demonstrates the successful learning of the agent trained by our approach. We should not expect that this gap should reach a zero value due to the possible inconsistencies in human actions.

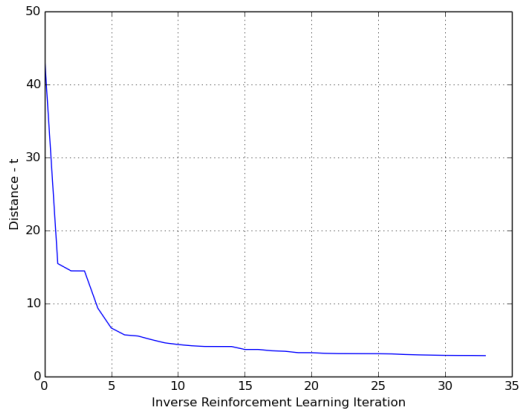


Figure 3: Euclidean distance to human's feature expectation.

B. Testing

In testing, we performed 50 testing episodes for the trained DNN and Hybrid AL via IRL models and used 20 episodes of the human data-set, and then calculated the averages and standard deviations of cumulative total reward per action and the percentage of task success. Table III provides these results of three setups: Human, DNN, and Hybrid AL via IRL.

Table III: Averages and Standard Deviations of Reward Per Action and Number of Steps, and Success Rates of the Three Setups in the Testing Cases.

Experimental ID	Action Reward	Steps Number	Success Rate (%)
Human	0.144 ± 0.004	217.6 ± 10.9	100
DNN	0.111 ± 0.055	399.7 ± 312.2	80
Hybrid-AL-IRL	$\mathbf{0.147} \pm 0.002$	$\mathbf{207.8} \pm 8.7$	**100**

The results show that the trained Hybrid AL via IRL agent performs more precisely than a human when its own and the corresponding human's averages of reward per action and the number of steps are $(0.147, 207.8)$ and $(0.144, 217.6)$, respectively, and overpasses this value of the trained DNN agent being just 0.111. The percentages of task success of both are 100% compared to just 80% of the trained DNN agent. These results demonstrate that our proposed algorithm is able to perform better than the traditional supervised learning-DNN with the same human data-set, and reach the level of human performance even though the Hybrid AL via IRL performance is more effective than that of the human's one. To have clear view of this efficiency, we provide some general observations in Figure 4.

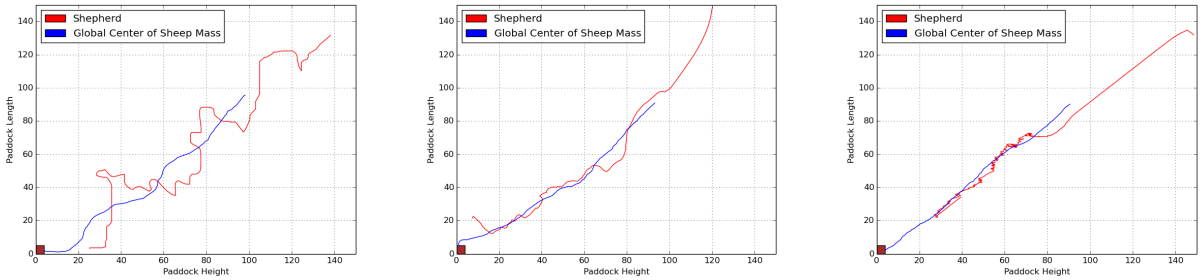
Firstly, sub-figure (a) illustrates the human performance in herding sheep towards the target. The movement of the human shepherd is flexible in a quite wide range. When applying the DNN shepherd for this mission, the sub-figure (b) shows that the movement of the shepherd is narrowed in a smaller range than that of the human shepherd. Particularly, the Hybrid AL via IRL shepherd moves highly precisely in a shortest path which avoids some additionally unexpected steps in human movement.

VII. Conclusion and Future Work

This paper proposes the Hybrid AL via IRL approach in addressing the problem of continuous action and state spaces in the shepherding problem wherein the shepherd drives the swarm of sheep towards the target. Our approach integrates the radial basis function with the K-mean algorithm to produce the effective feature mapping function in the inverse reinforcement learning approach. This function is able to address the problem of continuous state space in the human demonstrations. Additionally, with applying the deep deterministic policy gradients (DDPG) algorithm for the reinforcement learning step, our approach also allows to handle the problem of continuous action space when it outputs a real vector.

In our experiments, we conducted the primary two setups: the first is a supervised learning approach by training the deep neural network (DNN) on the human data-set, and the latter is our proposed approach trained on the same human data-set. The results demonstrates that the trained Hybrid AL via IRL agent overpasses the trained DNN agent, and even though perform preciser than human when it can find shorter paths reducing unexpected actions of the human.

Future studies will look deeply at the impact of parameters of the proposed feature mapping function like the number of



(a) Human (b) DNN (c) Hybrid AL via IRL

Figure 4: Trajectories of the entire swarm in three different testing cases of the setups. The X and Y axes represent the size of the environment (150x150). The agents are randomly initialized at the top right corner of the environment, and the target is located at the bottom left corner.

centers on the quality of the performance, and also investigate the structure of the DDPG algorithm in order to output more effective policies. From that, we aim to transfer this model to the physical environment that is highly complex and dynamic.

REFERENCES

[1] S. Martinez, J. Cortes, and F. Bullo, "Motion coordination with distributed information," *IEEE Control Systems Magazine*, vol. 27, no. 4, pp. 75–88, 2007.

[2] J. P. Desai, J. P. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, 2001.

[3] R. Carelli, C. De la Cruz, and F. Roberti, "Centralized formation control of non-holonomic mobile robots," *Latin American applied research*, vol. 36, no. 2, pp. 63–69, 2006.

[4] H. Oh, A. R. Shirazi, C. Sun, and Y. Jin, "Bio-inspired self-organising multi-robot pattern formation: A review," *Robotics and Autonomous Systems*, vol. 91, pp. 83–100, 2017.

[5] D. J. Stilwell and B. E. Bishop, "Platoons of underwater vehicles," *IEEE control systems*, vol. 20, no. 6, pp. 45–52, 2000.

[6] D. P. Scharf, F. Y. Hadaegh, and S. R. Ploen, "A survey of spacecraft formation flying guidance and control. part ii: control," in *American Control Conference, 2004. Proceedings of the 2004*, vol. 4. IEEE, 2004, pp. 2976–2985.

[7] A. Serrani, "Robust coordinated control of satellite formations subject to gravity perturbations," in *American Control Conference, 2003. Proceedings of the 2003*, vol. 1. IEEE, 2003, pp. 302–307.

[8] A. Robertson, T. Corazzini, and J. P. How, "Formation sensing and control technologies for a separated spacecraft interferometer," in *American Control Conference, 1998. Proceedings of the 1998*, vol. 3. IEEE, 1998, pp. 1574–1579.

[9] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[10] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on robotics and automation*, vol. 14, no. 6, pp. 926–939, 1998.

[11] D. Xu, X. Zhang, Z. Zhu, C. Chen, and P. Yang, "Behavior-based formation control of swarm robots," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[12] A. Sen, S. R. Sahoo, and M. Kothari, "Cooperative formation control strategy in heterogeneous network with bounded acceleration," in *Control Conference (ICC), 2017 Indian*. IEEE, 2017, pp. 344–349.

[13] A. Guillet, R. Lenain, B. Thuilot, and V. Rousseau, "Formation control of agricultural mobile robots: A bidirectional weighted constraints approach," *Journal of Field Robotics*, 2017.

[14] H. Iima and Y. Kuroe, "Swarm reinforcement learning method for a multi-robot formation problem," in *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2298–2303.

[15] T. Nguyen, H. Nguyen, E. Debie, K. Kasmarik, M. Garratt, and H. Abbass, "Swarm q-learning with knowledge sharing within environments for formation control," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[16] C. Speck and D. J. Bucci, "Distributed uav swarm formation control via object-focused, multi-objective sarsa," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 6596–6601.

[17] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 443–451.

[18] D. Strömbom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. Sumpter, and A. J. King, "Solving the shepherding problem: heuristics for herding autonomous, interacting agents," *Journal of the royal society interface*, vol. 11, no. 100, p. 20140719, 2014.

[19] J.-M. Lien and E. Pratt, "Interactive planning for shepherd motion." in *AAAI Spring Symposium: Agents that Learn from Human Teachers*, 2009, pp. 95–102.

[20] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *arXiv preprint arXiv:1806.06877*, 2018.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[23] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE transactions on neural networks and learning systems*, no. 99, pp. 1–11, 2018.

[24] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[25] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[26] 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.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[29] A. Meyer-Baese and V. J. Schmid, *Pattern recognition and signal analysis in medical imaging*. Elsevier, 2014.

[30] F. Chollet, "Keras: Theano-based deep learning library," *Code: https://github. com/fchollet. Documentation: http://keras. io*, 2015.