Apprenticeship Bootstrapping

Hung Nguyen
School of Engineering & IT
UNSW-Canberra
Canberra, Australia
Hung.Nguyen@student.adfa.edu.au

Matthew Garratt
School of Engineering & IT
UNSW-Canberra
Canberra, Australia
m.garratt@adfa.edu.au

Hussein Abbass
School of Engineering & IT
UNSW-Canberra
Canberra, Australia
h.abbass@adfa.edu.au

***Abstract*— Apprenticeship learning is a learning scheme based on the direct imitation of humans. Inverse reinforcement learning is used to learn a reward function from human data. Coupling Inverse reinforcement learning with reinforcement learning has demonstrated production of human-competitive policies. However, obtaining human subjects with the right level of skills for complex tasks can be a challenge. We propose a new learning scheme called Apprenticeship Bootstrapping to learn a composite task using human demonstrations on sub-tasks. The scenario is a ground-air interaction task with an Unmanned Aerial Vehicle that needs to maintain 3 autonomous Unmanned Ground Vehicles within range of an imaging sensor. For validation, we show that the bootstrapped policy performs as good as a policy learnt from a human performing the composite task. The method offers a clear advantage when skilled humans are available for simpler tasks that form the building blocks for a more complex task, where availability of experts is limited.**

***Index Terms*— Inverse Reinforcement Learning, Learning by Imitation, UAVs, UGVs, Ground-Air Interaction**

INTRODUCTION

In reinforcement learning (RL) [1], an agent executes actions and receives feedback signals in the form of a *reward*, indicating the degree of a successful action. The agent attempts to learn a policy (a state-action map) that maximizes its own accumulated reward function over time. The success of a classical RL agent depends on a successful hand-design of the reward function that can guide the search process towards optimal policies.

Inverse Reinforcement Learning (IRL) replaces hand-designed reward functions by learning the reward function from human-generated policies. The underlying assumption is that the human-generated policies are generated from experts; thus, are somehow optimal. The machine-induced reward function is then used by an RL algorithm to find a near-optimal machine-policy that is closest to the expert's policy.

By using demonstrations from humans, Ng and Russell [2] developed three algorithms for IRL that vary in the state space representation and sample size: tabulated reward functions over a finite state space, linear approximation of the reward function over infinite state space, and a partial sample of observed trajectories. They used linear programming to maximally discriminate between sub-optimal policies and the observed policy. The key finding of their study is a demonstration of the existence of multiple reward functions that can lead to the same optimal policy.

In [3], Abbeel and Ng proposed apprenticeship learning to find an approximate policy that is close enough to the observed one. This approach avoids the need to recover the reward function explicitly. Instead, a state's reward is represented by a sum of its linearly weighted features.

In 2015, the Google's DeepMind group proposed the Deep Q-Network (DQN) algorithm [4], [5]. The work demonstrated that DQN agents are able to successfully learn from complex state spaces, with the performance of DQN outperforming all classical RL algorithms. The DQN agent achieved a professional skill-level when tested on 49 Atari games.

While in the general case DQN does not necessarily need human demonstrations, in the example above, the success of DQN was pertinent to the existence of a large collection of human-played Atari games, which allowed the research team to replace the classic human-designed reward function to train an RL algorithm with an automatically derived reward function from human demonstrations. Unfortunately, this is not a simple endeavor in some real-world applications such as the control of autonomous systems. When designing a new task for an autonomous system, there is no guarantee that a human expert is available, let alone exists, to perform these machine tasks.

Research in [6], [7], [8], [9], [10], [11] has shown significant applications of RL for successful development of autonomous systems. Apprenticeship learning in conjunction with IRL was then successfully applied to design a controller for a helicopter [12].

Recently, there has been an increasing amount of research focusing on coordinating UAVs and UGVs to execute complex tasks, such as in surveillance [13], [14], [15]. This non-trivial ground-air interaction problem comes in different forms [16], [17], but one important common aim in many applications is to design an autonomous controller for the UAV to support the UGVs in their mission's objective. This controller can either replace or augment a human controlling the UAV, extending the horizon to supervisory control of the UAV, or to fully autonomous operations. The asymmetry in the behavioral characteristics and dynamics of UAVs and UGVs makes the problem non-trivial. The UAV is far faster and has a broader view than UGVs [18]. It is suitable to sense and supply a larger map of the environment [19] from the UAV(s). In contrast, the UGVs are suitable for resolving small-scale spatial goals that may require closer inspections of objects. This asymmetry makes the learning problem challenging.

A primary challenge in the literature presented above is what to do if a human expert is not available to create a dataset for apprenticeship learning in new and complex control tasks and/or platforms. We present "Apprenticeship Bootstrapping" (ABS) for situations where an expert may not be available to build a database of demonstrations for a complex task. However, we assume that we have access to experts on the sub-skills level required to perform sub-tasks independently.

ABS, as the name indicates, uses IRL to approximate the reward functions over the sub-tasks, then use an RL with these reward functions to bootstrap a control policy for the composite task.

We use a ground-air interaction scenario, where a UAV attempts to maintain a group of mobile autonomous UGVs within its camera range. This task is decomposed into three sub-tasks: climbing (*i.e.* increasing the UAV height) when UGVs disperse to increase the radius needed to bring the UGVs within range; descending (*i.e.* decreasing the UAV height) when UGVs group together to decrease the radius needed to bring the UGVs within range; and lateral movements to track the UGVs as they move.

A human performed each sub-task separately then the success of ABS was tested by comparing the controller learnt from bootstrapping from the individual sub-tasks (we call it Primitive Scenario) with a controller that learnt from data collected from a human performing the composite scenario (we call it Composite Scenario) whereby all three sub-tasks co-existed. Data from all four scenarios (3 primitive and 1 composite) were collected. The proposed approach was successful and produced policies that are comparable to those learned directly from the data collected on the composite scenario.

Apprenticeship Bootstrapping via Inverse Reinforcement Learning Algorithm

In the reinforcement learning framework, 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) recovers/approximates the reward function by learning it from human demonstrations.

ABS via IRL [3] assumes that the reward function, $R(s)$, is a linear function represented as a weighted, w, sum of a state feature vector $\phi(s) = \{\phi(s_1), \ldots, \phi(s_i), \ldots, \phi(s_K)\}$:

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

IRL assumes that the human expert is working with the optimal policy; thus, there is an optimal weight vector w^* such that the optimal reward $R^*(s)$ is

$$R^*(s) = w^* \cdot \phi(s), \forall s \in S \tag{2}$$

IRL uses human demonstrations to then find the weight vector w that approximates w^*.

For a fixed policy, its value is defined as follows:

$$V^\pi(s_0) = w \cdot E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right] \tag{3}$$

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

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right] \tag{4}$$

If the dynamic model is unknown, the feature expectations vector $\mu(\pi)$ can be seen in Equation 5:

$$\mu(\pi) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{5}$$

where m is the number of trajectories.

When the feature mapping function, ϕ, and demonstrations are given, the Deep Q-learning (DQN) algorithm [4], [5] is used to discover $\tilde{\pi}$ in order to make $\mu(\tilde{\pi})$ as close as possible to $\mu(\pi_E)$.

ABS modifies the algorithm presented in [3] as shown in Algorithm 1. The main modifications center around using sub-tasks to approximate the reward signal needed for a subset of the state vector. These incomplete approximations are then fused through an expectation function to approximate the overall reward function for an RL to discover a policy over the composite task. The primary assumption we make is that there exists a human who can perform the sub-tasks. Each sub-task encode sub-skills for the composite task. However, the fusion of these sub-skills is left to the RL agent to learn how to switch and combine them. The modifications are shown in italics, while the remainder of the algorithm is identical to the one presented in [3].

It is important to emphasize that in our case, the sub-tasks are orthogonal. Therefore, taking the sum of feature expectations vector will take the union of the state space. In 1, the authors explained that averaging policies's feature expectation is equivalent to calculating the feature expectation of a distribution over policies. However, in our case each sub-task is defined with a different state space and actions; in other words, the sub-spaces are non-commensurable. When we average the policies' feature expectations, the non-commensurable state spaces are morphed into a composite state representing the overall state space needed for conducting the overall task.

Theoretical Basis of ABS

In Q-learning, a strategy is a sequence of transitions from a starting state to a goal state. This sequence can be obtained by using an appropriate action selection algorithm such as ϵ-greedy. A strategy is a skill that an agent learns to adopt when it is faced with the context imposed by the starting state and the sub-sequent contexts/states it encounters. A human in IRL is assumed to be skilled to be capable of producing such a strategy. However, this assumption comes with practical implications; the most important one is that a human with such skills is available. In practice, in complex tasks this human

may not be available because the task is new or because it is expensive to access someone with such high skills. This motivates ABS, where by decomposing the skill into sub-skills that require less skilled humans, we can bootstrap the higher skills from these building blocks.

The sub-skills represent a decomposition of the action space. Not all actions are needed for a sub-skill. It may also involve a decomposition of the state space since sub-skills are associated with simpler contexts that represent partial representations of the original context. Below, we will explain the above formally.

Define $S = \{S_1, S_2, ..., S_N\}$ and $A = \{A_1, A_2, ..., A_L\}$ to be the original complex state and action spaces of a complex task, respectively. Here, S_n and A_l represent the sub-state and sub-action spaces.

In the original IRL [3], the reward function, $R(s)$, is calculated through the features of each state as described in Equation 1. The feature function is used to extract K features over states: $\phi : S \rightarrow [0,1]^K$. We can define K_n to be the number of features associated with sub-state space n with $K = \bigcup_{n=1}^{N} K_n$.

Let $\phi^n(s)$ be the feature vector of the sub-state space n, where $\phi^n(s) = \{\phi_1^n, \phi_2^n, ..., \phi_{K_n}^n\}\ \forall s \in S_n$. To synchronize all feature vectors in all sub-state spaces, we will rewrite $\phi^n(s)$ as in Equation 6 by setting the features of other sub-state spaces to zero values as follows:

$$\phi^n(s) = \{0, ..., 0, \phi^n_{K_{n-1}+1}, \phi^n_{K_{n-1}+2}, ..., \phi^n_{K_{n-1}+K_n}, 0, ..., 0\}. \tag{6}$$

Supposing that the composite task might be divided into H sub-tasks. Let $D = \{D_1, D_2, ..., D_H\}$ be a set of demonstrations of all sub-tasks, where D_h is a set of h sub-task demonstrations. Each set of h sub-task demonstrations, D_h, is comprised of state-action pairs (s_t, a_t). To form the set D_h, in the sub-state space S_h, the expert is required to perform actions in the sub-action space A_h; meanwhile $S_h \in S$ is one of the sub-state spaces S_n in the whole complex state space, and $A_h \in A$ is one of sub-action spaces A_l in the whole complex action space.

In Abbeel and Ng 2004, the aim was to find a feature expectations vector of a π policy, $\mu(\pi)$, which is close to the expert feature expectations vector μ_E. The feature expectations vector is calculated as in Equation 5. Let $\mu_E^{D_h}$ be the expert feature expectations vector of the sub-task h, and then $\mu_E^{D_h}$ is calculated as in Equation 7

$$\mu_E^{D_h} = \frac{1}{m_h} \sum_{i=1}^{m_h} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{7}$$

where m_h is the number of trajectories, and $s_t \in S_h$.

If $\mu_E^{Primitive}$ is the expert feature expectations vector, which is calculated through the set of the expert demonstrations performed in sub-tasks, then $\mu_E^{Primitive}$ is as being represented in Equation 10

$$\mu_E^{Primitive} = \frac{1}{H} \sum_{h=1}^{H} \mu_E^{D_h} \tag{8}$$

$$= \frac{1}{H \times m_h} \sum_{i=1}^{H \times m_h} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{9}$$

$$= \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{10}$$

where M is the number of trajectories of demonstrations of all sub-tasks, $M = H \times m_h$ and $s_t \in S$.

In Equation 10, the feature vector $\phi(s_t^{(i)})$ is calculated as in Equation 6. This is consistent with the original mathematical proof of the original IRL algorithm when the reward function has still been calculated through the extracted features of the whole state space of the complex task. Besides, a sufficient combination among sub-action spaces is provided in the reinforcement learning step. This combination covers all possible actions in the complex task.

Therefore, using $\mu_E^{Primitive}$ still guarantees that the obtained policy is identical or even better than the policy learned from demonstrations performed in the complex task. This guarantee also demonstrated that the proposed changes in Algorithm 1 do not affect the correctness of the original IRL. The results in the next sections show that our proposed algorithm is able to find policies close to the human policy and even better than the original IRL policy obtained in the complex UAV and UGVs scenario.

Environment and Task Description

The simulations are conducted with three UGVs: UGV1, UGV2, and UGV3, and a single cooperating UAV. The aim is for the UAV to maintain all UGVs within Field of View (FoV) of the UAV's image sensor without losing any UGV from the FoV or creating a FoV that is much larger than what is needed to accommodate the smallest manifold containing the three UGVs. To collect human demonstrations, the system allows a human to teleoperate the UAV from a distance. A pictorial representation of the system is shown in Figure 1.

In all scenarios, UGVs start from their base, where we always assume that the initial location of the UAV is at the center of the UGVs' base. A number of manoeuvre profiles are designed for the UGVs. The task for the UAV is decomposed into two objectives. In the first objective, the UAV needs to minimize the distance between its own center of mass and the center of mass of the UGVs within its FoV. The second objective is to minimize the difference between the radius of the UAV's camera FoV and the ideal radius needed. We define the ideal radius as the radius of the smallest circle to encapsulate the manifold formed by the UGV formation. While the UAV has two cameras: a forward-looking camera and a top-down view camera, we only refer to the latter in this paper and don't use the former.

The above environment is built using the Gazebo simulator [20], ROS framework [21] and OpenAI Gym [22]. A

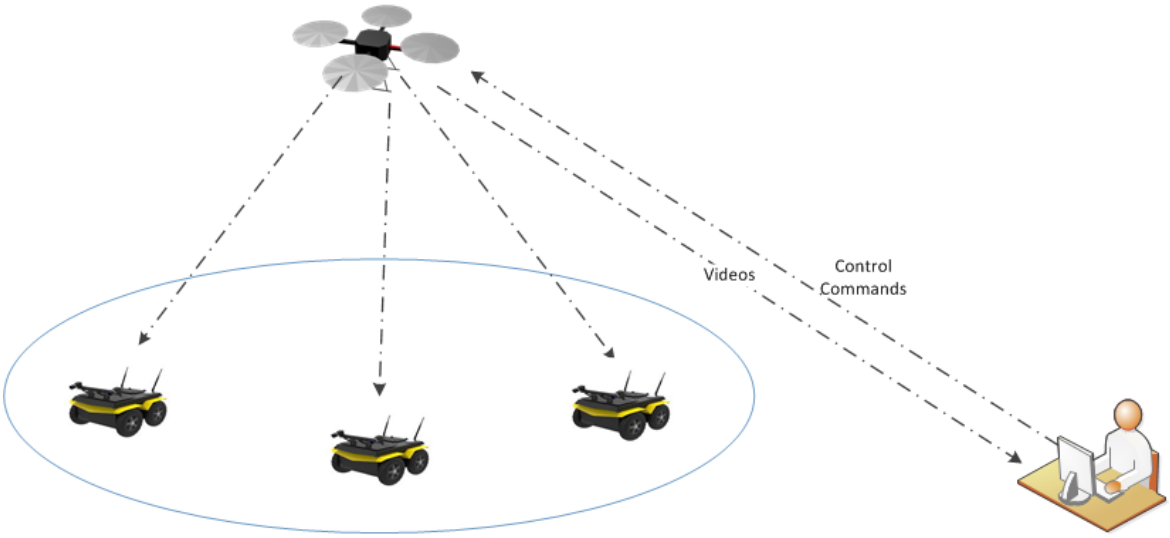

Fig. 1. The model of the UAV and UGVs Coordination Task

Algorithm 1 Apprenticeship Bootstrapping (ABS) via Inverse Reinforcement Learning Algorithm; a modification of the algorithm presented in [3] with the additional steps shown in italics.

Input: : *A feature mapping* ϕ*, and sub-task demonstrations* , $\{D_1, D_2, D_3, ..., D_H\}$.

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

1: Randomly choose policy $\pi^{(0)}$, estimate $\mu^{(0)} = \mu(\pi^{(0)})$ via Monte Carlo, and set $i = 1$.
2: *Initializing the parameters of DQN.*
3: **for** *each sub-task* **do**
4: *Calculating the expert feature expectations vector* $\mu_E^{D_h}$ *from* h *sub-task demonstrations,* D_h*, based on Equation 5*
5: **end for**
6: *Calculating the total expert feature expectations vector,* $\mu_E^{Primitive}$ *as described in Equation 10.*
7: *Set* $w^{(1)} = \mu_E^{Primitive} - \mu^{(0)}$ *and* $\bar{\mu}^{(0)} = \mu^{(0)}$.
8: *Set* $t^{(1)} = \|\mu_E^{Primitive} - \mu^{(0)}\|_2$.
9: **if** $t^{(i)} \leq \epsilon$ **then**
10: terminate
11: **end if**
12: **while** $t^{(i)} > \epsilon$ **do**
13: *Using DQN to compute the optimal policy* $\pi^{(i)}$ *with* $R = (w^{(i)})^T \phi$.
14: Compute $\mu^{(i)} = \mu(\pi^{(i)})$ and set $i = i + 1$.
15: Set $a = \mu^{(i-1)} - \bar{\mu}^{(i-2)}$.
16: Set $b = \mu_E - \bar{\mu}^{(i-2)}$.
17: Set $\mu^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{x^T y}{x^T x} x$.
18: Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$.
19: Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$.
20: **end while**

schematic diagram of the architecture to integrate the three systems together is presented in Figure 2.

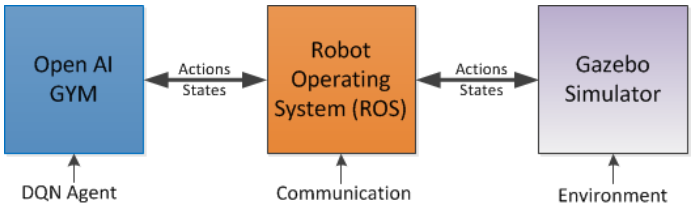

Fig. 2. The simulation environment protocol

Gazebo simulator [20] is used to design the task scenario. Tum-Simulator [23], which is a drone Gazebo simulator package, is used. In this paper, this package is simulated for the Parrot AR Drone 2. The unmanned ground vehicles simulator package is the Husky simulator [24], which simulates a Husky medium size robot mounted with Microsoft Kinect and laser rangefinder sensors.

The control interface agent is programmed in Python to be compatible with OpenAI Gym [22]. When collecting human demonstrations, the control interface agent receives a real video stream from the UAV bottom-viewing camera, and simultaneously communicates the human's actions back to Gazebo. The human controls the drone using a Keyboard. The human gets to see the image from the UAV top-down camera alone to have a fair setup comparable to the IRL Controller.

The Robot Operating System (ROS) [21] framework is used as an interface between the control interface agent and the Gazebo simulator environment. The actions and states are sent and received through ROS messages. We use ROS Indigo installed on Ubuntu 14.04.

The collected human demonstrations are then used by the

IRL training algorithm. Once training is complete, the same control interface agent is used, but this time with the human being replaced with the trained IRL for testing.

The UGVs'action space consists of 2 continuous real-valued actions representing the linear velocity, V, and the angular velocity (yaw rate), ω. The manoeuvre profile for the UGVs is prescripted according to the scenario being tested.

The UAV actions space consists of 24 discrete actions with each action representing the linear velocity including longitudinal, lateral, and altitude and is denoted using the values (p, r, a), respectively. We use 0.5 (m/s) for longitudinal velocity, 0.5 (m/s) for lateral velocity, and 1 (m/s) for altitude when the UAV attempts to move forward, turn left, or climb. Negative values were used for moving backward, turning right, and descending. Both longitudinal and lateral velocities were increased or decreased by 0.3 (m/s) when the maneuver involves a speed change.

The state vector of the environment is a 6-D tuple of the continuous variables presented in Table I.

TABLE I
THE STATES SPACE

State ID	State Name	State Description
1-2	(cx, cy)	Center of the UAV from bottom camera
3-4	$(UGVx, UGVy)$	UGV Center of Mass within the UAV image
6	ra_i	Ideal UGV radius within image
7	ra_a	Actual UGV radius within image

Meanwhile, (cx, cy) are received from the UAV down-facing camera. $(UGVx, UGVy)$ and (ra_i, ra_a) are calculated based on a pinhole camera model. ra_a is the distance from $(UGVx, UGVy)$ to the furthest UGV position within the bottom image.

The state space is limited to the four values $(cx, cy, UGVx, UGVy)$ for the Fixed-Altitude setup, the last two (ra_i, ra_a) for the Climb and Descend setups, and all 6 values are used for the 2 remaining setups.

The first objective is given in Equation 11, where $||.||$ denotes the second norm. The second objective is given in Equation 12. Both objectives are to be minimized as being indicated by the down-facing arrow.

$$\downarrow distance\_error = \int_t ||(cx, cy) - (UGVx, UGVy)|| \quad (11)$$

$$\downarrow radius\_error = \int_t (ra_i - ra_a)^2 \quad (12)$$

The DQN architecture used in this paper is given in Figure 3. The network consists of an input layer, two fully-connected hidden layers, and a fully-connected output layer. The number of nodes in each hidden layer is dependent on each setup, which is described in the Experiments section. The states defined in Table I are used as inputs, while the outputs, as discussed above, are discrete actions.

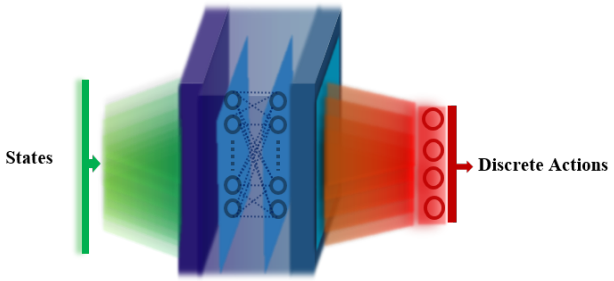

Fig. 3. The Deep Q-Network structure.

Human Demonstrations Scenarios

We use four maneuvers in this paper: Lateral-Movements-Fixed-Altitude maneuver, Climb-Only maneuver, Descend-Only maneuver, and Lateral-Movements-With-Climb-Descend maneuver. The first three maneuvers are what we call primitive manoeuvres, while the fourth maneuver is a composite and more complex manoeuvre requiring switching between the three primitive ones. There are 6 basic actions to control the UAV: roll actions (Move Left, Move Right), pitch actions (Move Forward, Move Backward), and yaw actions (Climb, and Descend). The formation of UGVs in each of the three primitive maneuvers is shown in Figure 4.

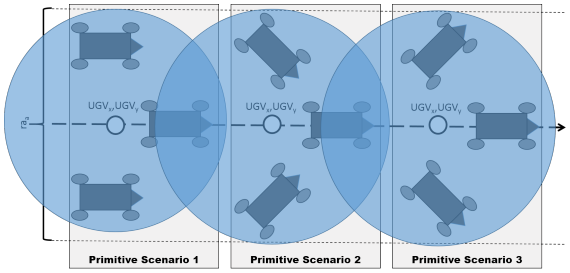

Fig. 4. UGVs Formation in the Three Primitive Scenarios.

In the Lateral-Movements-Fixed-Altitude maneuver, UGVs are allowed to move forward, turn left, and turn right. However, they need to move as a group with synchronized homogeneous action set to maintain their formation intact. This fixes the manifold of the UGV formation over the course of the scenario. In this scenario, the human needs to focus on four basic actions of moving right/left and moving forward/backward. The actions space is of 8 discrete actions including 4 basic actions and 4 speed-up basic actions.

In the Climb-Only maneuver, three distinct phases are followed. In the first and third phase, UGVs move forward with fixed formation and the same linear velocity. In the second phase, UGV2 and UGV3 separate from UGV1. They separate in opposite directions, which cause the radius required to bring them back to the UAV camera FoV to increase. This necessitates a UAV "climb" action. We label this behavior of

the UGVs "separation," as the UGVs get spread away from each other similar to the separation/repulsion force in a swarm Boid model.

The Descend-Only maneuver is identical to the Climb-Only maneuver except for the second phase, where UGV2 and UGV3 converge on the position of UGV1, reducing the radius required to bring them back to the UAV camera FoV. This necessitates a UAV "descend" action. We label this behavior of the UGVs "cohesion," as the UGVs get attracted to each other similar to the cohesion/attraction force in a swarm Boid model.

In both Climb-Only and Descend-Only maneuvers, the forward movement of the UGVs would mean that the human needs to only control the height of the UAV; climbing it in the first maneuver and descending it in the second. The actions space in both of the maneuvers is comprised of three basic actions including climb-forward, descend-forward, and fixed-altitude-forward.

The last maneuver, Lateral-Movements-With-Climb-Descend, is a combination of the above three. A scenario consists of 4 periods. Four movement patterns of the UGVs are labelled 1 to 4, representing Separation-Left, Cohesion-Right, Separation-Right, and Cohesion-Left manoeuvres, respectively.

Each scenario consists of four periods, where UGVs move according to a manoeuvre drawn randomly from the above four basic manoeuvres. For example, a scenario that follows 1-2-3-4 will start by by having the UGVs separate while turning left, then when this manoeuvre gets completed, UGVs start converging on each other while turning right, followed by turning right again while separating, and lastly turning left while converging on each other.

In this setup, the human needs to combine actions by moving the UAV in appropriate directions and climbing and/or descending it as necessary to achieve the mission objective; therefore, the actions space is composed of all 24 discrete actions.

In the rest of the paper, we will use the following short naming style for four maneuvers: Fixed-Altitude, Climb, Descend, and Combined, respectively.

EXPERIMENTS

Demonstrations from the human subject are collected for all setups; while the number of episodes are 23 episodes with 6742 instances, 21 episodes with 2520 instances, 24 episodes with 3164 instances, and 20 episodes with 2613 instances, respectively. Variations in the number of episodes, and consequently number of instances, were necessary because the four scenarios are of different lengths to complete each maneuver.

For each of the three first maneuvers, the original IRL in [3] is used. In the fourth maneuver, two tests were conducted. The first (composite scenario) used the human subject data from the fourth maneuver and the original IRL. The second (primitive scenario) used ABS and the human subject from the three primitive maneuvers. We analyze and compare all five scenarios.

The Deep Q-Network is trained with a replay memory size of 100,000 state-action pairs, the discount factor $\gamma = 0.99$, mini-batch size 32, and the learning rate 0.00025. During training, the $\epsilon - greedy$ is decreased linearly from 1 to 0.05, and fixed at 0.05 thereafter. The epsilon decay is 0.995 for every episode. Tensorflow and Keras libaries [25] were used to design the deep Q-network. The deep network was trained on an NVIDIA GeForce GTX 1080 GPU. Because there are differences in the number of states and discrete actions in different scenarios, the number of nodes in each hidden layer for each setup is different. The first and second hidden layers have 100 and 50 units, respectively, for the first three experiments; while in the last two, 300 units are used for both hidden layers. In all experiments, the number of episodes in the DQN step is 100. The feature mapping ϕ transfers each state to a binary array.

After 16 IRL iterations for the first three experiments (about 12 training hours), and 50 IRL iterations for the two last experiments (around 48 hours), the IRL agents were tested using the testing scenarios. In each experiment, the agent is tested 5 times on generated cases. For the Fixed-Altitude maneuver, the Climb maneuver and the Descend maneuver, the UGVs movement testing paths are nearly fixed in every episode; however, variations among the maneuvers are caused by uncertainties in the behavioral envelope of the UAV dynamics. For the Combined maneuver, five testing cases are generated, and used in both composite and primitive scenarios.

RESULTS AND EVALUATION

To evaluate performance, we calculate the distance between the UAV Center of mass (cx, cy) and the center of UGVs' mass $(UGVx, UGVy)$, as well as the difference between the actual radius ra_i and the ideal radius ra_a.

In Tables II and III, we present the average and standard deviation of these two metrics for each experiment. The errorbars in these tables are large because of the way the human is tasked. In each sub-task, the human focuses on the sub-task, while the performance is measured on all sub-tasks. For example, if the human is tasked to only control height, the human will only be allowed to adjust height, causing higher variations of distance errors. The human performance propagates to the performance of IRL.

In Table II, the IRL agents are able to align the UAV with the UGV's center of mass with performance nearly equivalent to the human subject in the Fixed-Altitude setup, and better performance in both the Climb and Descend setups. With respect to the radius, the IRL agents also managed to perform better than the human subject in the Climb and Descend setups.

Table III shows that both ABS and original IRL agents are able to perform nearly equivalent to the human subject. Despite that the reward functions used in ABS are based on an observed state space that is a subset of the overall state-

TABLE II

AVERAGE AND STANDARD DEVIATIONS OF ERRORS IN ALL TESTING EXPERIMENTS USING PRIMITIVE SKILLS. RESULTS HIGHLIGHTED IN BOLDFACE ARE DIFFERENT FROM HUMAN'S RESULTS AND THE DIFFERENCES ARE STATISTICALLY SIGNIFICANT AT $\alpha = 0.05$.

Experiment ID	Human		Original IRL	
	Distance Errors MSE $\mu \pm \sigma$	Radius Errors MSE $\mu \pm \sigma$	Distance Errors MSE $\mu \pm \sigma$	Radius Errors MSE $\mu \pm \sigma$
Fixed-Altitude	16.9 $\pm$ 11	5.1 $\pm$ 1.8	**26.7 $\pm$ 11.8**	**28.0 $\pm$ 20.6**
Climb	33.9 $\pm$ 20.3	3 $\pm$ 2.4	27.7 $\pm$ 18.3	2.5 $\pm$ 2.3
Descend	77.2 $\pm$ 65.9	3.8 $\pm$ 2.7	54.8 $\pm$ 45.5	**1.9 $\pm$ 2.1**

TABLE III

AVERAGE AND STANDARD DEVIATIONS OF ERRORS IN ALL TESTING EXPERIMENTS IN THE COMBINED EXPERIMENTS. RESULTS HIGHLIGHTED IN BOLDFACE ARE DIFFERENT FROM HUMAN'S RESULTS AND THE DIFFERENCES ARE STATISTICALLY SIGNIFICANT AT $\alpha = 0.05$.

Experiment ID	Distance Errors	Radius Errors
Human Performance	21.2 $\pm$ 13.1	6.1 $\pm$ 5.4
Original IRL	33.7 $\pm$ 22.5	12.6 $\pm$ 9
AB	23.3 $\pm$ 13.2	**12.6 $\pm$ 6.5**

space used in the original IRL. ABS seemed to have favored the distance metric more than the original IRL did.

It is worth mentioning that despite the variations discussed above, in all experiments, all three agents (human and artificial) always maintained the UGVs within the range of the camera in all maneuvers and all test cases.

To better understand the phenotypic differences between the human performance and IRL agents, we visualize the behaviour of the UAV when it is under human control and compare it with the behaviour when it is under IRL control in each scenario. For space limitations, we restrict the visualization to those presented in Figure 5. Comparing the ABS and original IRL reveals similar observed behaviors in the manner by which the UAV is flying.

In Figure 5, as expected, the IRL trajectory is qualitatively similar to human trajectory. The AB trajectory, however, has smoother oscillation but with larger magnitude. This smoother oscillation is desirable as too much oscillations generate inefficient flights; possibly outside the performance envelop of the UAV. However, this comes with a cost, where distance error increases. The more the UAV attempts to have a smoother trajectory, the less it is able to quickly adjust to the UGVs and the greater the distance error. The averaging of feature expectations vector favors the smoother trajectory. It is possible to control the trade-off between smoothness and distance errors by changing the fusion function.

CONCLUSION AND FUTURE WORK

In this paper, an apprenticeship bootstrapping method is proposed. It is aimed at problems requiring high levels of skills that a human expert may not be readily to provide demonstration data for the machine learning algorithm to train from. However, we assume that there are human experts who can perform the sub-skills that compose the overall high skill-level task and generate demonstration data for these sub-skills.

We have tested the apprenticeship bootstrapping method on a ground-air tracking task, where an unmanned aerial vehicle (UAV) attempts to maintain a mobile group of unmanned ground vehicles (UGVs) within its camera range. We used a human to generate the ground truth for training and testing using four maneuvers: Lateral-Movements-Fixed-Altitude maneuver, Climb-Only maneuver, Descend-Only maneuver, and Lateral-Movements-With-Climb-Descend maneuver. The first three maneuvers define the low-level skills required to perform the fourth maneuver; these are: the UAV needs to either track UGVs by moving forward, climbing and descending. In the fourth maneuver, the UGVs move in more complex maneuvers where all three forms of behaviour (lateral tracking, climbing and descending) are used simultaneously.

In the first three maneuvers, the original IRL is used to learn from human demonstrations. In the fourth setup, two experiments are conducted to assess our proposed apprenticeship bootstrapping. The first experiment uses the original IRL for the human demonstrations collected in this setup. Meanwhile, in the second experiment, the Apprenticeship Bootstrapping is used to learn from sub-task human demonstrations.

The results revealed that IRL agents reached a standard close to human performance in all experiments. For Climb and Descend maneuvers, the IRL agents are able to adjust the altitude better than the human subject. Most significantly, the results show that the Apprenticeship Bootstrapping agents perform in a comparable manner to humans and are competitive to agents trained on data collected from humans performing the more complex task.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[2] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, 2000, pp. 663–670.

[3] 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.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[5] 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, pp. 529–533, 2015.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[7] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 135–149, 2011.

[8] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward Q-learning: The combination of Sarsa algorithm and Q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013.

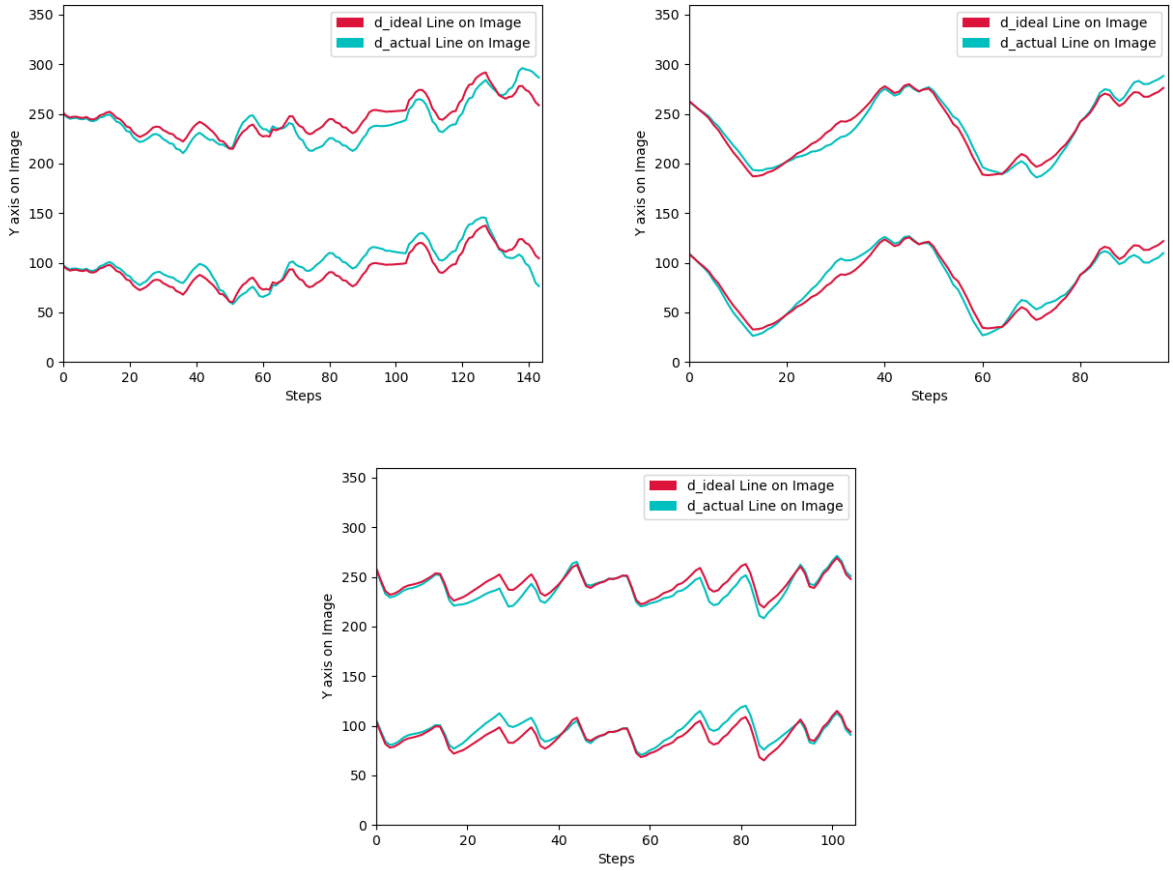

Fig. 5. The Ideal and Actual UGVs Circle Trajectories on Vertical Image in the Combined Maneuver. Top: Human Discrete Actions; Middle: ABS; Bottom: Original IRL. It is important to note that UGVs do not have identical dynamics in the three scenarios because of stochastic noise.

[9] X. Chen and A. El Kamel, "A reinforcement learning method of obstacle avoidance for industrial mobile vehicles in unknown environments using neural network," in *Proceedings of the 21st International Conference on Industrial Engineering and Engineering Management 2014*. Springer, 2015, pp. 671–675.

[10] A. El-Fakdi and M. Carreras, "Two-step gradient-based reinforcement learning for underwater robotics behavior learning," *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 271–282, 2013.

[11] Z. Miljković, M. Mitić, M. Lazarević, and B. Babić, "Neural network reinforcement learning for visual control of robot manipulators," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1721–1736, 2013.

[12] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[13] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 16–25, 2006.

[14] S. Minaeian, J. Liu, and Y.-J. Son, "Vision-based target detection and localization via a team of cooperative UAV and UGVs," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 7, pp. 1005–1016, 2016.

[15] S. Zhifei and E. M. Joo, "A review of inverse reinforcement learning theory and recent advances," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 2012, pp. 1–8.

[16] J. Chen, X. Zhang, B. Xin, and H. Fang, "Coordination between unmanned aerial and ground vehicles: A taxonomy and optimization perspective," *IEEE transactions on cybernetics*, vol. 46, no. 4, pp. 959–972, 2016.

[17] S. L. Waslander, "Unmanned aerial and ground vehicle teams: Recent work and open problems," in *Autonomous control systems and vehicles*. Springer, 2013, pp. 21–36.

[18] A. Aghaeeyan, F. Abdollahi, and H. A. Talebi, "UAV–UGVs cooperation: With a moving center based trajectory," *Robotics and Autonomous Systems*, vol. 63, pp. 1–9, 2015.

[19] H. Yu, R. W. Beard, M. Argyle, and C. Chamberlain, "Probabilistic path planning for cooperative target tracking using aerial and ground vehicles," in *American Control Conference (ACC), 2011*. IEEE, 2011, pp. 4673–4678.

[20] N. Koenig and A. Howard, "Gazebo-3D multiple robot simulator with dynamics," 2006.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[23] H. Huang and J. Sturm, "Tum simulator," *ROS package at http://wiki.ros. org/tum_simulator*, 2014.

[24] ClearpathRobotics, "ROS husky robot," *ROS package at http://wiki.ros.org/Robots/ Husky*, 2017.

[25] F. Chollet *et al.*, "Keras: Deep learning library for theano and tensorflow," *URL: https://keras. io/k*, 2015.