Hierarchical Decision and Control for Continuous Multitarget Problem: Policy Evaluation With Action Delay

Jiangcheng Zhu, Jun Zhu, Zhepei Wang, Shan Guo, and Chao Xu

***Abstract*— This paper proposes a hierarchical decision-making and control algorithm for the shepherd game, the seventh mission in the International Aerial Robotics Competition (IARC). In this game, the agent (a multirotor aerial robot) is required to contact targets (ground vehicles) sequentially and drive them to a certain boundary to earn score. During the game of 10 min, the agent should be fully autonomous without any human interference. Regarding the lower-level controller and dynamics of the agent, each action takes a duration of time to accomplish. Denoted as an action delay, in this paper, this action duration is nonconstant and is related to the final reward. Therefore, the challenging point is making the agent "aware of time" when applying a certain action. We solve this problem by two approaches: deep Q-networks and lookup table. The action delay predictor in the decision-level is fitted by a lower-level controller. Through simulations by the example of the shepherd game, the effectiveness and efficiency of this approach are validated. This paper helps our team winning the first prize in IARC 2017, and keeps the best record of this mission since it was released in 2013.**

***Index Terms*— Deep reinforcement learning (RL), Monte Carlo sampling, policy evaluation.**

I. Introduction

NOWADAYS, learning to control a robotic system via machine learning is a hot topic in artificial intelligence. Reinforcement learning (RL), also called adaptive dynamic programming (DP), is closely related to optimal feedback control in both the formulation and solution method [1]–[5]. These RL studies on optimal control focus on the value iteration and policy iteration regard on the reward function as linear quadratic regulator.

A new trend of RL controller is end-to-end. Exploiting the new tools developed in deep learning, the state-of-the-art RL controller processes raw sensory input and observed rewards, and generates actuator signal in an end-to-end neural network. Some typical algorithms are deep Q-networks (DQNs) [6], [7], deep deterministic policy gradient [8], trust region policy optimization [9], and proximal policy optimization [10].

Contrasting to the end-to-end trend, robotics often has a hierarchical decision and control system. In the field of robotics, there are several studies that propose training a controller by learning approaches [11]–[15]. However, in many cases, classic controllers still dominate for their guaranteed performance. On the other hand, real environments are usually dimension explosive for end-to-end learning algorithms. Therefore, applying learning algorithm in high-level decision-making with low-level classic controllers is a practical option.

Different from discrete problems (see AlphaGo [16]) and continuous problems which can be discretized with constant time resolution (e.g., aerobatic helicopter [17], [18]), RL applied for decision-making for robotic system has a major difference, that one action takes a duration of time to finish. Specifically in time-sensitive tasks, the action duration may influence the final reward. The game is such a problem with hierarchical structure of high-level decision-making (discrete) and low-level controller (continuous), as shown in Fig. 1.

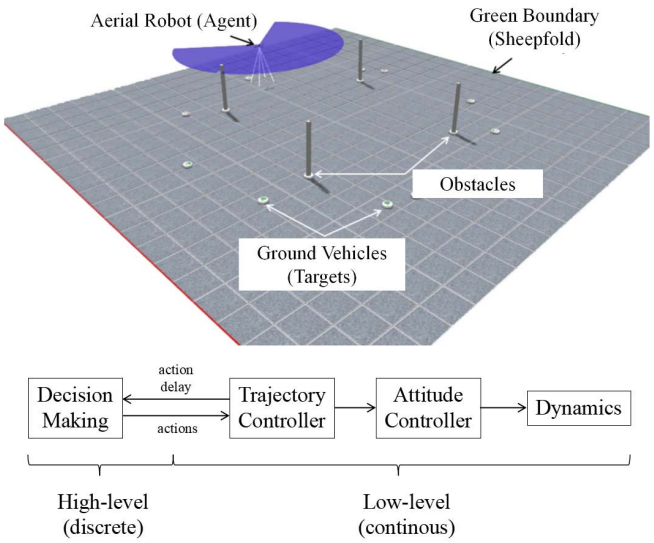


Fig. 1. IARC shepherd game.

Manuscript received March 30, 2017; revised December 12, 2017 and May 18, 2018; accepted May 31, 2018. Date of publication July 2, 2018; date of current version January 21, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61473253, in part by the Fundamental Research Funds for the Central Universities of China, and in part by the National Natural Science Foundation of China through Foundation for Innovative Research Groups under Grant 61621002. *(Corresponding author: Chao Xu.)*

J. Zhu, J. Zhu, Z. Wang, and C. Xu are with the State Key Laboratory of Industrial Control Technology and Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: cxu@zju.edu.cn).

S. Guo is with the School of Mathematical Sciences, Zhejiang University, Hangzhou 310027, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2018.2844466

2162-237X © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

The major contribution of this paper is to formulate this kind of problems as an RL framework. We design an instant reward that absorbs the action delay. Therefore, the action delay is independent of the RL level, and only be applied in the process to generate instant reward. We show that the action delay is a function of state and action, and apply it for instant reward generator.

We organize the rest of this paper as follows. Section II describes the motivation: the shepherd game. Section IV presents the problem formulation. Section V introduces two approaches to solve this problem: DQN and lookup table (LUT). Section VI introduces the instant reward generator and action delay predictor. Section VII provides some simulation results. Conclusion and discussion are drawn in Section VIII.

II. Motivation

We first state briefly the background of the International Aerial Robotics Competition (IARC) mission 7, which is called the "Shepherd mission." On a competition area of 20 m × 20 m, or it is the analogy of "grassland" corresponding to the "shepherd mission." On the competition arena, there are three parties of players, including a drone, 10 ground mobile robots, and four mobile obstacles, which represent "shepherd dog," "*sheep*," and "wolves" in this robotic game, respectively.

The robotic game imitates the actual shepherding process to some extent. First, the drone (or the shepherd dog) should be able to avoid collision autonomously with four mobile obstacles (or wolves). Second, there are two ways to change the moving direction of each ground mobile robot; one way is to land the drone in front of a ground mobile robot, then it makes a U-turn; the other way is to fly the drone physically touch the top of a ground mobile robot, then it makes a turn of $\pi/4$ to the right or left direction randomly. In addition, each ground mobile robot also makes a U-turn if its moving direction remains unchanged for 20 s except random variations. The moving direction of each ground mobile robot could alter within a small range due to manufacturing precision, which can be modeled as random error mathematically in the computer simulation. The final target of winning the competition is to drive at least 4 out of the 10 ground mobile robots across the green edge (or *sheepfold*) of the square arena (as shown in Fig. 1) within 10 min (i.e., the total time duration of a trial is $T_{\text{total}} = 10$ min). During the mission, if any ground mobile robot moves out the (cliff) edges rather than the green one, it will be removed from the arena immediately by the working staff. If more than six ground mobile robots escaped from the edges rather the green one, this trial fails.

This game requires high-level autonomy without any human intervention of drones from all the participants, including flight control, localization, path planning, collision avoidance, target detection, target tacking, driving, and so on. In addition, during the whole mission, drones are not allowed to fly higher than 3 m in order to disable a global view in the air of the arena. This makes the mission even more challenging in terms of self-localization and distribution awareness of all the ground mobile robots.

The challenge of this game is to investigate the high-level decision of the drone over the total duration of each trial (i.e., $\forall t \in [0, 10 \text{ min}]$), including the target sequence to drive, action sequence to deliver (i.e., either top touch or front collision). We propose a simulation-based data-driven approach to solve the decision problem. First, we build a simulator which can simulate the behaviors of ground mobile robots (both the 10 sheep robots and the four *wolf* robots) according to the competition rules, where the model of Dubin's car with random perturbations is used. For the drone, a fully nonlinear dynamic model of a quadrator in 3-D space is used, which is derived based on standard techniques in classical rigid-body dynamics, such as Euler–Lagrange mechanics. Within the simulator, a path planning problem is solved to generate a reasonably well trajectory for a drone from a given position in the air to the predesignated mobile robot on the ground with obstacle avoidance considered. Many available techniques to provide the solution for the path planning problem, such as time optimal control, artificial potential field [19], path integral control [20], minimum snap [21], and so on.

III. Numerical Description of Mission

Hereafter, we briefly represent the three agents in this game as: shepherd, sheep, and wolf. To describe the shepherd game numerically, we denote the shepherd as agent to make decision and applying action, sheep as an action target i (action target set $\mathcal{T} = \{i | i = 1, 2, \ldots, 10\}$). The shepherd can apply two kinds of action type j (action type set $\mathcal{O} = \{j | j = 0, 1, 2\}$, in which $j = 0$ means NONE operation).

For simplicity in the high-level decision-making, we drop the third dimension of the shepherd (i.e., the height of a shepherd), then we use (x_g, y_g) to represent the coordinate of the shepherd. Similarly, we use (x_i, y_i) to denote the position of the sheep i.

A. Trigger Signals

Trigger signals are impulses applied on a sheep. A sheep has a U-turn trigger signal by every $\Delta = 20$ s. As a global timer t, runs on all sheep on court, we do not denote timer on each sheep. The trigger signal from global timer can be stated as

$$\delta_{\text{timer},i}(t) = \begin{cases} 0, & (t \mod \Delta \neq 0) \\ 1, & (t \mod \Delta = 0). \end{cases} \tag{1}$$

There are two options for the actions, i.e., top touch or head touch, which makes the sheep rotate $\pi/4$ or π, respectively. The action can be taken either at the straight line status or the rotation status. We define the actions can be effective as the following events in terms of distance

$$d_{g,i}(t) = \|(x_g(t), y_g(t)) - (x_i(t), y_i(t))\|_2, \quad i \in \mathcal{T} \tag{2}$$

and

$$\theta_{g,i}(t) = \arctan \frac{(y_g - y_i)}{(x_g - x_i)}, \quad i \in \mathcal{T} \tag{3}$$

$$\Delta\theta_{g,i}(t) = |\theta_{g,i}(t) - \theta_i(t)|. \tag{4}$$

Define $\mathcal{R}_1$ as the radius of a sheep and $\mathcal{R}_2$ as the radius of top touchpad. For head touch, the shepherd needs to touch

the sheep on its head. Therefore, we denote the trigger signal of head touch as

$$\delta_{\text{head},i}(t) = \begin{cases} 0, & (\text{else}) \\ 1, & (d_{g,i}(t) = \mathcal{R}_1,\ \Delta\theta_{g,i}(t) < \pi/4). \end{cases} \tag{5}$$

The trigger signal of top touch is denoted as

$$\delta_{\text{top},i}(t) = \begin{cases} 0, & (\text{else}) \\ 1, & (d_{g,i}(t) < \mathcal{R}_2). \end{cases} \tag{6}$$

Please note that $\delta_{\text{head},i}$ and $\delta_{\text{top},i}$ are triggered only once for each attempting touch, i.e., there is only one trigger signal before the shepherd leaves and reenters the effective area $\mathcal{R}_1/\mathcal{R}_2$.

B. Modeling

Each sheep has two seperate status: straight line motion and rotation. We reorganize the Dubins car model into two separate models which can switch to each other due to trigger signals. Denoting $s_i = (x_i, y_i, \theta_i)^{\text{top}} \in [0, 20]^2 \times [0, 2\pi]$ for sheep i, the straight line motion and rotation can be modeled as

$$\begin{pmatrix} \frac{ds_i}{dt} \\ \frac{dt_{\text{turn},i}}{dt} \end{pmatrix} = \begin{pmatrix} \frac{dx_i}{dt} \\ \frac{dy_i}{dt} \\ \frac{d\theta_i}{dt} \\ \frac{dt_{\text{turn},i}}{dt} \end{pmatrix} = \begin{pmatrix} v_0 \cos\theta_i \\ v_0 \sin\theta_i \\ \nu \\ 0 \end{pmatrix}_{\text{if } t_{\text{turn},i}=0}$$
$$\text{or} \quad \begin{pmatrix} 0 \\ 0 \\ \omega_0 \\ -1 \end{pmatrix}_{\text{if } t_{\text{turn},i}>0} \tag{7}$$

where the linear motion velocity and angular velocity are denoted as v_0 (when there is no rotation command) and ω_0 (when there is unfinished rotation command). The random direction noise is denoted as ν, which is a zero-mean uniform distribution noise.

The value of countdown timer $t_{\text{turn},i}$ refers to the time for unfinished rotation. For the countdown timer $t_{\text{turn},i}$, it is defined as an internal state of sheep i. The sheep moves straight when $t_{\text{turn},i} = 0$, and rotates when $t_{\text{turn},i} > 0$. The trigger signal δ_i adds $t_{\text{turn},i}$ by a corresponding rotation time, as follows:

$$t_{\text{turn},i}(t) = \begin{pmatrix} t_{\text{turn},i}(t) + \frac{\pi}{\omega_0} & \text{if } \delta_{\text{head},i}(t) = 1 \\ t_{\text{turn},i}(t) + \frac{\pi}{4\omega_0} & \text{if } \delta_{\text{top},i}(t) = 1 \end{pmatrix}$$
$$\text{or} \quad t_{\text{turn},i}(t) = \left(t_{\text{turn},i}(t) + \frac{\pi}{\omega_0} \quad \text{if } \delta_{\text{timer},i}(t) = 1\right). \tag{8}$$

The $t_{\text{turn},i}$ can be considered as a *stack* of rotation command, as in Fig. 2. When the stack is empty, sheep moves straight. The trigger signals push rotation command to this stack as (8). When the stack is not empty, it pops rotation command to the sheep [i.e., $dt_{\text{turn},i}/dt = -1$ in (7)].

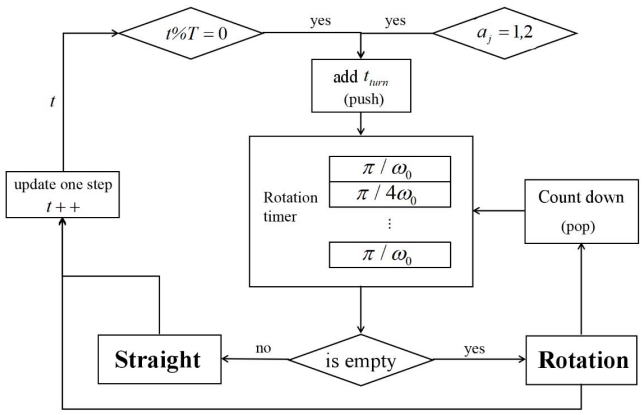


Fig. 2. Flowchart of rotation countdown timer t_{turn} for a single sheep.

IV. Problem Formulation

In this section, we introduce an important notion in this paper: periodic mean state. Then, we formulate the shepherd mission as a Markov decision process (MDP).

We denote t^k as the moment to make the kth decision, which can be accomplished after a duration t_a^k flying to the target sheep. We denote t_a^k as an action delay, for it is the delay from making decision to accomplishing action. At the moment, one action a^k is accomplished, the shepherd makes the next decision a^{k+1} at t^{k+1}. Therefore, $t^{k+1} = t^k + t_a^k$.

A. Periodic Mean State

With the step and time definition mentioned earlier, we can define the stepwise state for a sheep i as: $s_i^k = s_i(t^k)$, $s_i^{k+1} = s_i(t^{k+1}) = s_i(t^k + t_a^k)$

$$s_i(t^{k+1}) = \int_{t^k}^{t^k + t_a^k} \dot{s}_i(\tau) d\tau. \tag{9}$$

We plot the trajectories of a single sheep with the rules of [(7)–(9)]. Fig. 3(a) shows a periodic trajectory when no action is applied. When an action is applied, it jumps to a new cycle trajectory immediately, due to the change of countdown timer t_{turn} caused by trigger signal δ.

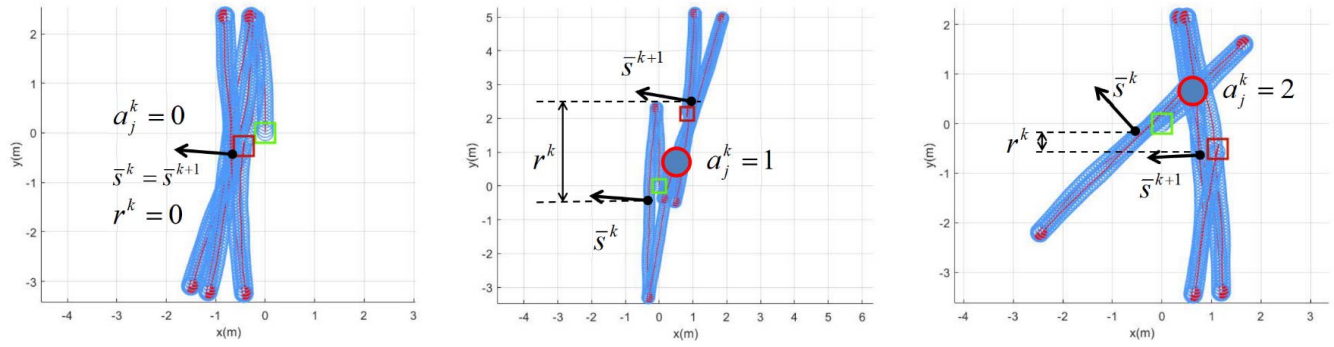
Therefore, we define the periodic mean state $\bar{s}_i$ as the mean value of the state of a sheep within two periods 2 T

$$\bar{s}_i(t) = (\bar{x}_i, \bar{y}_i, \bar{\theta}_i) = \frac{1}{2T} \int_t^{t+2T} s_i(\tau) d\tau. \tag{10}$$

Since the sheep does U-turn with a period of T with zero-mean direction noise and constant linear velocity, it is intuitive to say that its period mean state remains unchanged if there is no action applied on sheep i during this time duration

$$\bar{s}_i(t) = \bar{s}_i(t + \Delta t) \text{ if } a_{i,j}(t) = (i, 0), t \in (t, t + \Delta t). \tag{11}$$

However, if there is an action $a_j \neq 0$ applied on the sheep i at time t_a, the period mean state $\bar{s}_i(t)$ changes immediately at the time t_a: $\lim_{\delta t \to 0} \bar{s}_i(t_a - \delta t) \neq \lim_{\delta t \to 0} \bar{s}_i(t_a + \delta t)$, though the instant state remains the same $\lim_{\delta t \to 0} s_i(t_a - \delta t) = \lim_{\delta t \to 0} s_i(t_a + \delta t)$. It is because as in (8), the shepherd action only changes the $t_{\text{turn},i}$ instantly.



(a) (b) (c)

Fig. 3. Trajectory of sheep under different action a_j. Green and red boxes: initial and ending position, respectively. Red circle: position where the action a_j is applied on the sheep. (a) $a_j = 0$: none action. (b) $a_j = 1$: head touch. (c) $a_j = 2$: top touch.

The periodic mean state $\bar{s}_i$ of step $k+1$ is a function of state s and action a of step k

$$\bar{s}_i^{k+1} = \bar{s}_i(t^{k+1}) = \bar{s}_i(t^k + t_a^k) = \frac{1}{2T}\int_{t^k+t_a^k}^{t^k+t_a^k+2T} s_i(\tau)d\tau. \quad (12)$$

Note that there is no transition between $\bar{s}_i^{k+1}$ and $\bar{s}_i^k$, because time t is not an factor of $\bar{s}_i$. From (12), we use s_i^k rather than $\bar{s}_i^k$ to solve the $\bar{s}_i^{k+1}$. Therefore, to solve the next periodic mean state $\bar{s}_i^{k+1}$, we need to know the current state $s_i(t^k)$, the action a^k, the motion rule of the sheep $\dot{s}(t)$, and the action delay t_a^k. This is the reason using s_i^k rather than $\bar{s}_i^k$ as the *state* in the following MDP formulation.

B. MDP Formulation

The shepherd game is formulated as an MDP [22], which is widely applied in DP [23] and RL. MDP includes the factors: state space s, action space a, policy π, and instant reward r. The transition dynamics of MDP is random and defined by the probability density function $p(s^{k+1}|s^k, a^k)$.

In this paper, action space is denoted as a 2-D integer space. The first dimension is the target value from $\{1, \ldots, 10\}$, and the second dimension is the operation value from $\{1, 2\}$. The action set $\mathcal{A}$ as the Cartesian product of $\mathcal{T}$ and $\mathcal{O}$

$$\mathcal{A} = \mathcal{T} \times \mathcal{O} = \{a_{i,j} = (i, j) \big| i \in \mathcal{T}, j \in \mathcal{O}\}. \quad (13)$$

Let $s_g = (x_g, y_g) \in \mathbb{R}^2$ and $s_i = (x_i, y_i, \theta_i) \in \mathbb{R}^2 \times [0, 2\pi]$ be, respectively, the state vector of the shepherd and the sheep i. Let t be the global time in continuous timeline: $t \in (0, T_{\text{total}})$. The overall state s of MDP is defined as

$$\begin{aligned} s &= \{s_1, s_2, \ldots, s_{10}, s_g, t\} \\ &= \{(x_1, y_1, \theta_1), \ldots, (x_{10}, y_{10}, \theta_{10}), (x_g, y_g), t\}. \end{aligned} \quad (14)$$

The decision-making problem in the RL is formulated as solving the optimal policy for a cumulated reward: $R^k = \sum_{p=k}^{F} \gamma^{(p-k)} r(s^p, a^p)$. Policy refers to the mapping of states into actions: $\pi : s \to a$. Following the Q-learning framework, we implement ϵ-greedy policy in training and the greedy policy in inference

$$\begin{aligned} Q^\pi(s^k, a^k) &= \mathbb{E}_\pi[R^k | s^k, a^k] \\ a &= \arg\max_{a \in \mathcal{A}} Q^\pi(s, a). \end{aligned} \quad (15)$$

As the steps k in this MDP is not evenly distributed in time, we need to define an instant reward r which is step dependent rather than time dependent. The first option is the number of sheep into the sheepfold, i.e., if one sheep moves into the sheepfold, there is a $r = +1$ reward, otherwise $r = 0$. But this reward is too sparse in the training, which leads to a bad convergence result. The second option is to define the forward translation distance $\Delta\bar{y}^k$ based on the periodic mean state.

From (10), the s_i remains unchanged at $t^k + t_a^k$, while the periodic mean state $\bar{s}_i$ changes immediately due to the immediate change of $t_{\text{turn},i}$. Therefore, the forward translation distance $\Delta\bar{y}^k$ has an instant change at t^{k+1} as well.

As the target is placed in a $\{(x, y)\}$ coordinate, without losing the generality, we define that the sheepfold locates at the area $\{0 < x < 20, y \leq 0\}$. An action driven the sheep toward the $y = 0$ direction has positive reward

$$r^k = r(s^k, a^k) = -\sum_{i \in \mathcal{I}} \Delta\bar{y}_i^{k+1} \quad (16)$$

from which

$$\Delta s_i = (\Delta\bar{x}_i, \Delta\bar{y}_i, \Delta\bar{\theta}_i) = \bar{s}_i^{k+1} - \bar{s}_i^k. \quad (17)$$

V. Algorithms

Based on the MDP for the shepherd game, its state space is a multidimensional continuous space, and the action space is an integer space. Q-learning with value function [24] is suitable for this kind of problem.

We test two approaches to solve this problem. The first is to use a deep neural network to fit the Q function. The advantage of this method is that the original 33-dimensional state as (14) can be used by the neural network without manual feature extraction. The disadvantage is that due to the use of deep neural networks, convergence is not guaranteed and explainability is weak.

In the second approach, we analyze this game to make some simplification. Based on the periodic moving properties of the sheep, we build an LUT for state-space discretization. The advantage of this method is that the results are interpretable and controllable. The disadvantage is that this approach is not general for other problems.

A. Deep Q-Learning

We first introduce deep Q-learning, also called DQN [6], [7]. In these studies, the notion "deep" can be divided in two aspects. One is to use a deep visual network such as a convolutional neural network to extract information from the raw image. The other is to fit the Q-function through a multilayer full connected (FC) network. We focus on the latter one, i.e., training a deep FC network to approximate the Q-function (namely, value network).

Our DQN approach is slightly different from the classic one on the point that, one action needs to run for a duration of time: t_a, rather than a single timestep. The simulation and the way to calculate r is slightly different. The algorithm is summarized as Algorithm 1.

Algorithm 1 Training of DQNs

Input: Neural network $Q(s, a)$ initialized by random weights
Output: Trained value netwoks $Q(s, a)$

```
for step = 1 : N do
    while t < t_f do
        Choose action a^k k by ε-greedy policy
        Predict the running time of this action t_a^k
        Run the system for t_{a,k} with action a^k
        Calculate r^k as the cumulated instant reward during t_a^k
        Store the transition (s^k, a^k, s^{k+1}, r^k) to the replay buffer
        Sample training batch from replay buffer
        Update the value network
    end
end
```

B. Greedy Policy and Lookup Table

According to our reward based on the definition of periodic mean state, a head touch ($a_j = 1$) does not have future reward. This is because the sheep only jumps from one moving cycle to another moving cycle, with the cycle center changed. After this change finishes, there is no further effect from this action.

For top touch ($a_j = 2$), it has future effect since it changes the direction, then changes the instant reward of future head touch action. But from the experiment results, the top touch is often (not always) the first action applied on a sheep, followed by a sequence of head touch. Therefore, we can assume the greedy policy will be good enough (not global optimal) for this specific problem

$$a = \pi(s) = \arg\max_{a \in A} r(s, a). \tag{18}$$

We divide the reward vector $r(s, a)$ into independent reward referring to each target and action: $r(s, a) = \{r(s_i, s_g, t, a_{i,j}) | i = 1, \ldots 10\}$. The most important term in effect, the reward is the distance between the shepherd and the target sheep: $d_{g,i} = \|(x_g, y_g) - (x_i, y_i)\|_2$. Exploiting the reduction from position to distance, we can reduce the $\mathbb{R}^{22}$ state space to $\mathbb{R}^{10}$: $r(s, a) = \{r(d_{g,i}, \theta_i, t, a_{i,j}) | i = 1, \ldots, 10\}$.

TABLE I
LUT DESCRIPTION

Original	Preprocessed	Range	Resolution	Descritpion
s_g, s_i	d_i	$0 - 20m$	1 m	Agent-target distance
θ_i		$0 - 360°$	45°	Yaw angle of a target
t		$0 - 20s$	$2s$	Time in a period
a_j		$\{0, 1, 2\}$		Operation

Algorithm 2 Training of LUT

Input: Blank LUT, sample number M
Output: Reward LUT $\widehat{r}(d, \theta, t, a_j)$

```
for Each discrete table element do
    for samplenumber = 1 : M do
        Initialize a state tuple (d_0, θ_0, t_0) in this element range by uniform distribution
        Load this initialization tuple into training environment
        Run for one period with operation a_j and observe the result
        Update the policy evaluation by MC sampling
    end
    Store the mean value in the LUT;
end
```

Furthermore, for each target i, the distance, heading, and time are equivalent

$$r(d_{i_1}, \theta_{i_1}, t, a_{i_1,j}) = r(d_{i_2}, \theta_{i_2}, t, a_{i_2,j}) \quad \text{if } d_{i_1} = d_{i_2}, \quad \theta_{i_1} = \theta_{i_2}. \tag{19}$$

Therefore, our LUT can reduce to the reward on target independent distance and heading angle: $\hat{r}(d, \theta, t, a_j)$.

To store the training result from MC sampling, we build an LUT which discretizes the continuous state space, similar to [25]. Given a state s, we calculate the reward as (16). Since the existence of random events, the reward of each action $r(d, \theta, t, a_j)$ is measured by the sampling through the simulator, defined by (20). The reward value corresponding to pair (d, θ, t, a_j) is stored in a four-dimension LUT as given in Table I.

We initialize the state by sampling under a joint uniform distribution and discretize the initial state (d, θ, t, j) to index in the LUT. A simulation based on the behavior of the target (7) runs to generate the reward. The total sampling number is defined as M

$$\widehat{r}_M(d, \theta, t, a_j) = \frac{1}{M} \sum_{m=1}^{M} r^m(d, \theta, t, a_j). \tag{20}$$

C. Analysis

For LUT approach, denote the optimal action of greedy policy is a^*. The reward of a^* refers to the maximum value of the LUT $r(d_{g,i}, \theta_i, t, a^*_{i,j})$

$$a^* = \arg\max_{a} r(d_{g,i}, \theta_i, t, a_{i,j}) \tag{21}$$

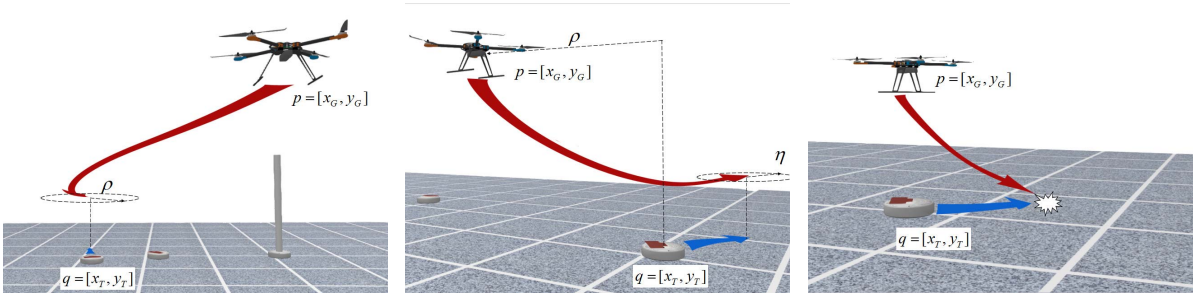


Fig. 4. Three phases of interactive process. (a) Cruise phase. (b) Track phase. (c) Approach phase.

TABLE II
COMPARISON ON TRAJECTORY PHASES

	Ending Condition	Controller	Delay	Prediction
Cruise	$d_i \le \rho$	Path integral	$t_{cr}(d_{g,i})$	From distance
Track	$d_{g,i} \le \eta$	PID	t_{tr}	Constant
Approach	$d_{g,i} \le \mathcal{R}$	PID	t_{ap}	Constant

while the action is chosen by

$$a = \arg\max_a \hat{r}(d_{g,i}, \theta_i, t, a_{i,j}). \tag{22}$$

For sample number $M_1 > M_2$, the estimation $\hat{r}$ is closer to the real value r due to law of large numbers [26]

$$\lim_{M\to+\infty} \widehat{r}_M(d, \theta, t, a_j) = r(d_{g,i}, \theta_i, t, a_{i,j}). \tag{23}$$

Therefore, $\pi_{M1}(s, a^*) > \pi_{M2}(s, a^*)$, i.e., the probability of choosing optimal action a^* will increase with the sample number M.

For the DQN approach, though the action delay exists, it can be drop into a Q-learning framework. The convergence of Q-learning is proven by [27]. For the value network, tricks such as replay buffer and target network [7] are applied to improve the convergence.

VI. INSTANT REWARD SIMULATOR

In both the DQN approach in Section V-A and LUT approach in Section V-B, the instant reward r cannot be directly observed from environment. In order to obtain the instant reward r regarding to the state and action, we run a simulator based on the model (7) for $t \in [t^k, t^k + t_a^k + 2T]$, then calculate the reward as (10) and (16).

The factor that cannot be directly obtained is the action delay t_a^k. In this shepherd game, as the action delay is from the flight to the target sheep, it should be a function of state, or briefly speaking, distance

$$t_a(s, a_{i,j}) = t_a(s_g, s_i, a_j) \approx t_a(d_{g,i}). \tag{24}$$

To verify our hypothesis that action delay is a function of distance, we analyze the flight to the sheep. We implement a three-phase trajectory generation algorithm [28] in our real-world flight test. The whole flight is divided into three phases: cruising, tracking, and approaching, as shown in Fig. 4 and Table II.

From our experiment results in [28], the time of the tracking phase and approaching phase is almost constant. The prediction $\widehat{t_a}$ should be larger than the real action delay t_a. Because

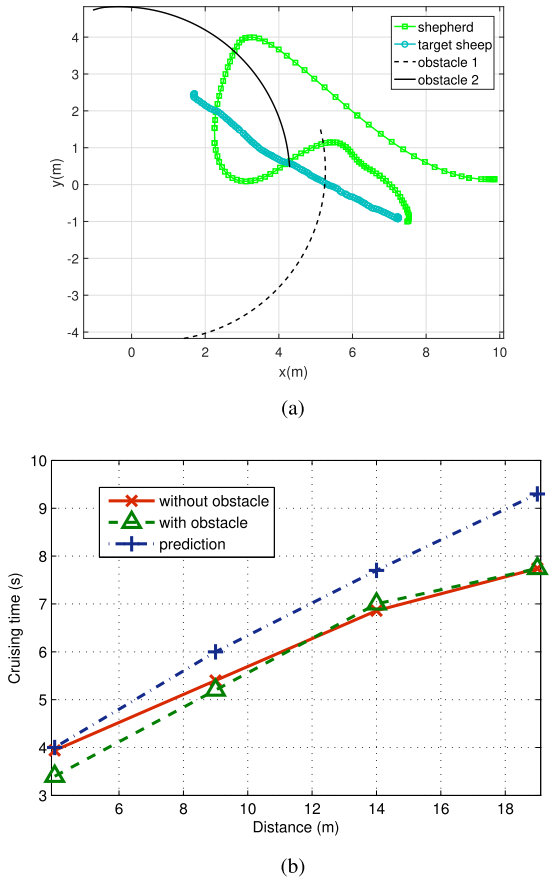


Fig. 5. Simulation results of path integral controller and time prediction of cruising phase. (a) Path integral controller trajectory with avoidance of two moving obstacles. (b) Time comparison of cruising phase.

the shepherd can reach and wait to meet the predicted time, but not vice versa

$$\widehat{t_a} \ge t_{\rm cr}(d_i) + t_{\rm tr} + t_{\rm ap}. \tag{25}$$

By using the path integral controller [29], [30], we implement an approximate constant-velocity cruising flight. We run a number of simulations in the Robot Operating System–Gazebo system with a quadcopter unmanned aerial vehicle and several ground vehicles. Fig. 5(a) is a sample of the horizontal trajectories of the sheep, shepherd, and two wolves. We setup the simulations with two cases and record the time $t_{\rm cr}$: one target and one obstacle (red solid line), one target without obstacle (green dashed line) as shown in Fig. 5(b). There is a redundancy between the blue dashed line and red/green lines in (25). This verifies our

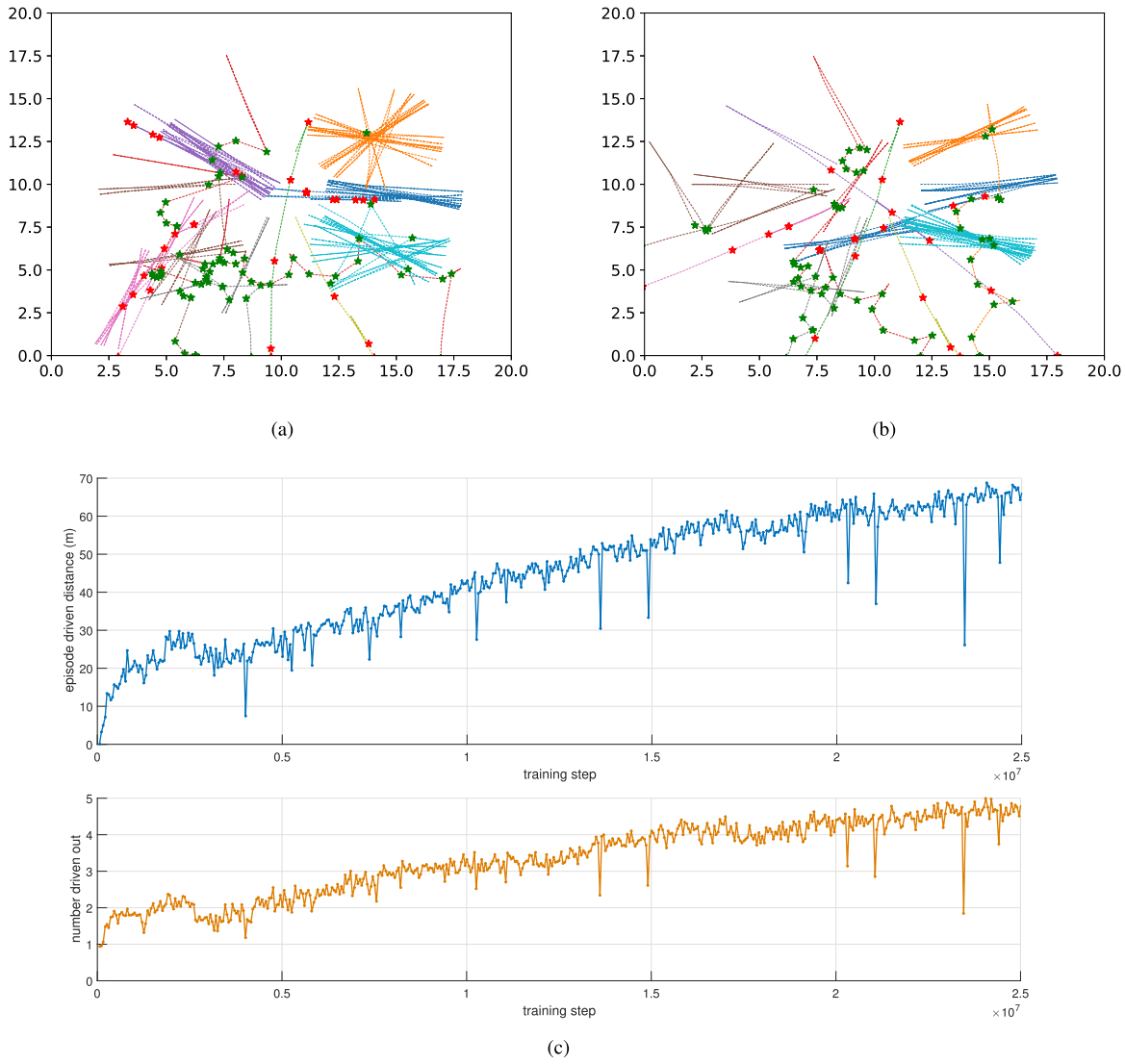


Fig. 6. Simulation results of DQN approach. (a) Trajectories of the targets. (b) Trajectory of the targets. (c) Performance on training steps. The ideal values of target out number and driving distance are 10 and 100 m, respectively.

hypothesis that the action delay can be a linear function of distance.

VII. Simulation Results

We test our LUT and DQN approaches by simulations. The LUT is implemented on MATLAB 2011b on a computer with Intel i7-4770 and 16-GB ROM. The DQN is implemented on Python 3, Tensorflow 1.4 on a computer with Intel i7-6700, NVidia GTX 960 M, and 16-GB ROM.

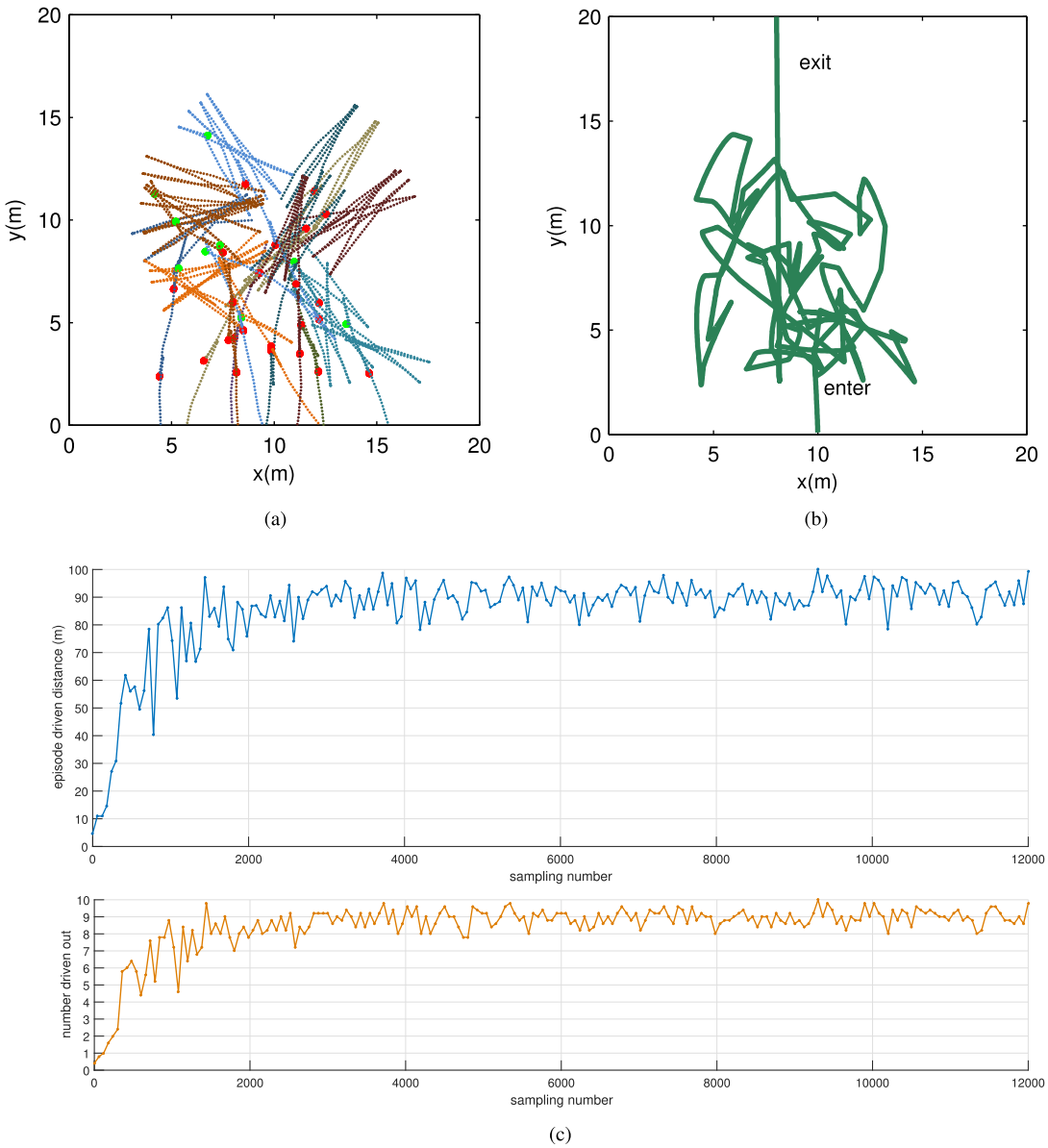
At the beginning of this simulation, all sheep poses at a circle centering at $(x = 10, y = 10)$ with a radius of 2 m. When one sheep touches the boundary of the 20-m square court, it stops moving. We run the simulation with an episode of 600 s ($t \in [0, T_{\text{total}}]$)

To compare the results, we introduce two metrics: episode driven distance d_Σ and episode driven number n_Σ. Episode driven distance is defined as $d_\Sigma = \sum_{i\in\mathcal{T}}(y_i(T_{\text{total}}) - y_i(0))$, whose optimal result is 100 m (all sheep starts from a circle centering at $y = 10$, and terminates at sheepfold $y = 0$). Episode driven number is the number of sheep into the sheepfold at the end: $n_\Sigma = \sum_{i\in\mathcal{T}} \mathbf{1}(y_i(T_{\text{total}}) = 0)$ ($\mathbf{1}$ is the indicator function). The optimal result of n_Σ is 10 (all sheep reach sheepfold).

A. DQN Approach

In our DQN approach, the value network is setup as a five-layer fully connected network. The number of neurons in each layer is [4096, 2048, 1024, 512, 256]. In one episode, after the state s^k is observed, the action a^k is generated by the ϵ-greedy policy in training and greedy policy in inference. Action delay t_a^k is calculated the timer prediction in Section VI by the distance of shepherd and target sheep in a_i^k. After that, we run the simulation as in Section VI to obtain the instant reward r^k. Then, the pair (s^k, a^k, r^k, s^{k+1}) is stored into a replay buffer.

Fig. 6(a) and (b) are the two typical episodes of simulation results, driving out five and six sheep to sheepfold, respectively. Fig. 6(c) is the performance on training steps.



(a) (b) (c)

Fig. 7. Simulation results of LUT approach. (a) Trajectories of the targets. (b) Trajectory of the agent. (c) Performance on sample number. The ideal values of target out number and driving distance are 10 and 100 m, respectively.

It shows an increasing performance in the whole training process. However, the final result is not as good as the LUT. The episode driven distance is around $d_\Sigma = 70$ m (compared to $d_\Sigma = 90$ m in LUT). The best number of driven out is $n_\Sigma = 5$ (compared to $n_\Sigma = 9$ in LUT).

From Fig. 6(a) and (b), we can see the DQN policy prefers to apply a sequence of top touch ($a_j = 2$). Though DQN also learns to apply sequential head touch as the green and brown trajectories, it is not efficient in other sheep. The policy of DQN shows a preference in conducting a sequence of same action type on a certain target sheep.

B. LUT Approach

Fig. 7(a) is a typical example of LUT simulation results, the shepherd drives 10 sheep into sheepfold within 600 s. Fig. 7(b) is the agent trajectory of this process.

In this simulation, when the sheep is not heading to the sheepfold, top touch [green dots in Fig. 7(a)] to change its direction. When a sheep is heading toward to or opposite to the sheepfold, the shepherd implements head touch ($a_j = 1$) at the right time to drive it to the sheepfold.

The performances d_Σ and n_Σ regard to sample number M is as shown in Fig. 7(c). In the test, the sample number M starts at 60 with an increasing step of 60. We run five episodes and use the mean value of d_Σ and n_Σ regarding to each M. This test shows a rapid growing performance from $M = 60$ to $M = 1200$. In this segment, the episode driven number n_Σ increase from 0 to around 9.

We can compare this performance with human performance in the manual compeittion IARC2015. Comparing to the human behavior, this shepherd implements a nonintuitive policy: head touching a sheep at the end of a period

($15 < t \mod T < 20$), driving the sheep to the sheepfold by a double U-turn (one is from δ_{head} and another is from δ_{timer}). While human operators tend to head touch the target at the beginning of a period ($0 < t \mod T < 5$). Our shepherd policy is verified to be the most efficient one, since it saves the time of one approaching phase for straight line motion, which is about 1 m more in each operation, comparing to the human policy.

VIII. Conclusion

This paper contributes toward the decision and control scheme of the shepherd game of IARC. The effectiveness and efficiency of our proposed method are verified by simulations of the shepherd problem. In the IARC2017, a timer-based policy, derived from this paper, supports our team winning the first prize of Asia-Pacific Venue and keeping the world best record of this mission. [1] The complete version of this method is not implemented due to the limitation of the sensory subsystem. The sensors cannot provide all the information required by this method.

The future work should be an approach from raw sensor rather than solved states input to continuous action output. An end-to-end approach without hand-crafted features would be updated. The reward would be "in sheepfold or not" rather than a defined function. We will also investigate the use of more sophisticated deep neural networks in the learning structure.

References

[1] P. He and S. Jagannathan, "Reinforcement learning-based output feedback control of nonlinear systems with input constraints," *IEEE Trans. Syst., Man, Cybern. B. Cybern.*, vol. 35, no. 1, pp. 150–154, Feb. 2005.

[2] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32–50, 3rd Quart., 2009.

[3] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Syst.*, vol. 32, no. 6, pp. 76–105, Dec. 2012.

[4] P. He and S. Jagannathan, "Reinforcement learning neural-network-based controller for nonlinear discrete-time systems with input constraints," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 2, pp. 425–436, Apr. 2007.

[5] H. Zhang, Y. Luo, and D. Liu, "Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490–1503, Sep. 2009.

[6] V. Mnih *et al.* (2013). "Playing Atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1312.5602

[7] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[8] T. P. Lillicrap *et al.* (2015). "Continuous control with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1509.02971

[9] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. (2017). "Proximal policy optimization algorithms." [Online]. Available: https://arxiv.org/abs/1707.06347

[11] B. Xu, C. Yang, and Z. Shi, "Reinforcement learning output feedback NN control using deterministic learning technique," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 635–641, Mar. 2014.

[12] X. Xu, Z. Hou, C. Lian, and H. He, "Online learning control using adaptive critic designs with sparse kernel machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 5, pp. 762–775, May 2013.

[13] M. Bojarski *et al.* (2016). "End to end learning for self-driving cars." [Online]. Available: https://arxiv.org/abs/1604.07316

[14] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. (2017). "End-to-end driving via conditional imitation learning." [Online]. Available: https://arxiv.org/abs/1710.02410

[15] M. Jaderberg *et al.* (2016). "Reinforcement learning with unsupervised auxiliary tasks." [Online]. Available: https://arxiv.org/abs/1611.05397

[16] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[17] A. Y. Ng *et al.*, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Berlin, Germany: Springer, 2006, pp. 363–372.

[18] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 19, 2007, pp. 1–8.

[19] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, Apr. 1991, pp. 1398–1404.

[20] H. J. Kappen, "An introduction to stochastic control theory, path integrals and reinforcement learning," in *Proc. AIP Conf.*, 2007, vol. 887, no. 1, pp. 149–181.

[21] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2011, pp. 2520–2525.

[22] R. Bellman, "A Markovian decision process," *J. Math. Mech.*, vol. 6, no. 5, pp. 679–684, 1957.

[23] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, no. 2. Belmont, MA USA: Athena Scientific, 1995.

[24] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," *J. Artif. Intell. Res.*, vol. 13, pp. 33–94, Aug. 2000.

[25] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834–846, Sep./Oct. 1983.

[26] W. Feller, "Law of large numbers for identically distributed variables," in *An Introduction to Probability Theory and Its Applications*, vol. 2, 2nd ed. New York, NY, USA: Wiley, 1971, pp. 231–234.

[27] F. S. Melo, "Convergence of *Q*-learning: A simple proof," Inst. Syst. Robot., Zürich, Switzerland, Tech. Rep., 2001, pp. 1–4.

[28] J. Zhu, J. Zhu, and C. Xu, "A simultaneous trajectory generation method for quadcopter intercepting ground mobile vehicle," *Int. J. Adv. Robot. Syst.*, vol. 14, no. 4, 2017.

[29] H. J. Kappen, "Path integrals and symmetry breaking for optimal control theory," *J. Stat. Mech., Theory Exp.*, vol. 2005, no. 11, p. P11011, 2005.

[30] V. Gómez, S. Thijssen, A. C. Symington, S. Hailes, and H. J. Kappen, "Real-time stochastic optimal control for multi-agent quadrotor systems," in *Proc. 26th Int. Conf. Automat. Planning Scheduling*, 2016, pp. 468–476. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13105

[1] 2017 IARC Competition Results and Technology Readiness Level: http://www.aerialroboticscompetition.org/mission7/downloads/2017_performance_result_summary.pdf

Jiangcheng Zhu received the B.Sc. degree in automation from Zhejiang University (ZJU), Hangzhou, China, in 2012, the M.S. degree in intelligent system, robotics and control from the University of California at San Diego, San Diego, CA, USA, in 2014. He is currently pursuing the Ph.D. degree with ZJU.

He is currently a Visiting Researcher with the Data Science Institute, Imperial College London, London, U.K. His current research interests include reinforcement learning and deep learning in robotics.

Jun Zhu received the B.Sc. degree in automation from Zhejiang University, Hangzhou, China, in 2016, where he is currently pursuing the M.Sc. degree with the College of Control Science and Engineering.

His current research interests include computer vision and artificial intelligence in robotics.

Zhepei Wang received the B.Sc. degree in automation from Zhejiang University, Hangzhou, China, in 2017, where he is currently pursuing the Ph.D. degree with the College of Control Science and Engineering.

His current research interests include motion planning and reinforcement learning.

Shan Guo received the B.Sc. degree in mathematics from Zhejiang University, Hangzhou, China, in 2012, where she is currently pursuing the Ph.D. degree with the School of Mathematical Sciences.

Her current research interests include optimal control and computational inverse problems.

Chao Xu received the Ph.D. degree in mechanical engineering from Lehigh University, Bethlehem, PA, USA, in 2010.

He is currently the TRUTH Associate Professor and an Associate Director with the Institute for Cyber-Systems and Control, Zhejiang University, Hangzhou, China. His current research interests include bioinspired locomotion and robotics, optimal control, and partial differential equation techniques in informatics.

Dr. Xu serves as the Managing-Editor for the *Journal of Industrial and Management Optimization*, an Associate Editor for the *Numerical Algebra, Control and Optimization*, and an Associate Editor for *Information and Control Technology* (a Chinese Journal).