Performance Comparison of Function Approximation-based Q Learning Algorithms for Autonomous UAV Navigation

Huy Xuan Pham, Hung Manh La, David Feil-Seifer, Luan Van Nguyen

***Abstract*— Using reinforcement learning to enable Unmanned aerial vehicles (UAV) carry out missions in unknown environments in which their mathematical model may not be available, is an active research topic. However, there is a challenge in implementing RL for real-world applications. This paper provides a framework for using RL to allow the UAV to navigate successfully in such environments. A performance comparison of three different Q learning methods: classical Q Learning (QL), Fixed Sparse Representation-based Q Learning (FSR-QL), and Radial Basis Function-based Q Learning (RBF-QL), is presented. We conducted simulations to show how the UAVs can successfully learn to navigate through an unknown environment. Through the simulation comparison of these three different learning methods (QL, FSR-QL, RBF-QL), the RBF-QL outperforms the others in term of space reduction and learning speed (convergence time).**

I. INTRODUCTION

Unmanned aerial vehicles (UAV), or drones, in missions involving navigating through unknown environment is becoming more popular in many applications, as they can host a wide range of sensors to measure the environment with relative low operation costs and high flexibility. Most applications normally make assumptions on the model describing the target, or prior knowledge of the environment [1], despite the fact that the knowledge and data regarding the environment are usually not accurate or available. Learning algorithms, such as reinforcement learning (RL), offer an alternative approach to conventional control methods, as they allow the UAV to learn the right behaviors through the interactions with the environment [2].

Applying RL algorithms to UAV applications has long been studied. They have been demonstrated effectively in UAV control problem, for example, achieving desired trajectory tracking/following [3], path planing [4], or carrying loads [5]. But most of the existing work often neglects important issues in implementing the algorithm in real application. One particular issue is representing the state - action value function in continuous space [6].

This paper aims to highlight the impact of approximation in RL algorithm, through a UAV navigation problem. We first implemented a classical Q-learning algorithm to help the drone navigate successfully to a goal without knowing its exact position. Then, we discussed how it can be improved with function approximation techniques, Fixed Sparse Representation (FSR), and Radial Basis Function (RBF), to deal with large Q table size. The remainder of this paper is organized as follows. Section II provides details on the problem statement, and the approach we use to solve the problem. Basics in RL algorithm are discussed in section III. We address the importance of function approximation techniques in section IV. A simulation is conducted on section V to study the impact on performance of these algorithms, and finally, we draw conclusion in section VI.

Huy Pham and Luan Nguyen are the PhD students, and Dr. Hung La is the director of the Advanced Robotics and Automation (ARA) Laboratory. Dr. David Feil-Seifer is an Assistant Professor of Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557, USA. Corresponding author: Hung La, email: `hla@unr.edu`

II. PROBLEM STATEMENT

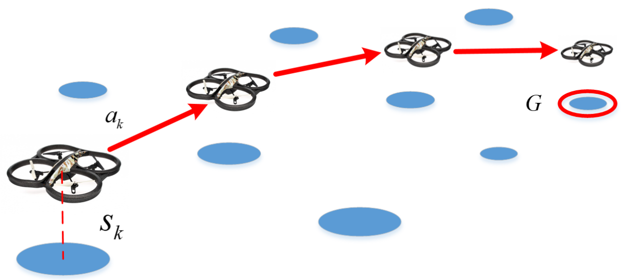

Fig. 1. A UAV navigation in unknown environment. Blue circles represent discretized state space, while the goal is marked by a red circle. The exact position of the goal is unknown to the UAV, but it can be recognized based on a special landmark.

Consider a simple problem of navigation, where a UAV, such as a quadrotors, needs to find a target (Figure 1) in an unknown environment. The exact position of the goal is unavailable, but the UAV can recognize it based on a pre-described landmark. We assume the UAV can localize itself. It would be a trivial problem if the UAV knows the location of the target and obstacles, because we can construct a robot motion planning based on these information using classical methods such as potential field [7]. However, in real applications like intelligence reconnaissance, such information is normally unavailable. RL algorithms do not require any prior knowledge of the environment, thus could be advantageous for solving this problem.

III. REINFORCEMENT LEARNING AND Q LEARNING

In RL, an agent learns a good behavior through interactions with the environment, judging its action given at a state by observing a reward signal from its surrounding, with a purpose to find an optimal policy that allow it to receive maximal reward over the learning course [2]. We defined a learning model assuming Markov Decision Process property as a tuple $< S, A, T, R >$, where S is a finite state list, and A is a finite set of actions. $s_k \in S$ and $a_k \in A$ are the state and the action that the agent takes at step k, respectively; T is the transition probability function, $T : S \times A \times S \to [0, 1]$, describing the probability of the agent that takes action a_k to move from state s_k to state s_{k+1}. In a deterministic setting, $T(s_k, a_k, s_{k+1}) = 1$. R is the reward function: $R : S \times A \to \mathbb{R}$ that quantifies the immediate reward of the agent for getting to state s_{k+1} from s_k after taking action a_k. We have: $R(s_k, a_k) = r_{k+1}$.

The agent in the system uses a state - action value function $Q(s_k, a_k)$ to measure the performance of an action in a given state, so it can determine the right action to take. A popular RL algorithm, known as classical Q-learning (QL) [8], allows the agent to iteratively compute the optimal value function of the state and action it has experienced into a Q-table, that latter can be recalled to decide which action it should take in every situation to optimize its rewards. In each iteration of QL, the state - action value function is updated by the following equation [2]:

$$\begin{aligned} Q_{k+1}(s_k, a_k) \leftarrow & (1-\alpha) Q_k(s_k, a_k) \\ & + \alpha[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a')], \end{aligned} \tag{1}$$

where $0 \leq \alpha \leq 1$ and $0 \leq \gamma \leq 1$ are learning rate and discount factor of the learning algorithm, respectively.

Algorithm 1: APPROXIMATED Q-LEARNING.

```
Input: α, γ ∈ (0,1], ε, number of episode N
Initialize θ_0(s_0, a_0) ← 0, ∀s_0 ∈ S, ∀a_0 ∈ A;
for episode = 1 : N do
    Measure initial state s_0
    for k = 0, 1, 2, ... do
        Choose a_k from A using greedy policy with
         probability p:
            a_k = { arg max_{a'} (φ^T(s_k, a')θ_k),   p ≥ ε;
                  { a random action in A,              otherwise.
        Take action a_k that leads to new state s_{k+1}:
        Observe immediate reward r_{k+1}
        Estimate φ(s_k, a_k) based on equation (3) or (4)
        Update:
            θ_{k+1} ← θ_k + α[r_{k+1} + γ max_{a' in A} (φ^T(s_{k+1}, a')θ_k)
                          − φ^T(s_k, a_k)θ_k] φ(s_k, a_k).
    until s_{k+1} ≡ G
```

IV. Q-LEARNING WITH FUNCTION APPROXIMATION

To guarantee convergence for the Q-learning algorithm, the agent must make sure that all state - action pairs continue to be updated [2]. In reality, the dimensions of the environment can be very large, making the representation of each state-action pair's values and Q table become a challenge, and increase the time needed for the RL algorithm to converge exponentially. The space needed to represent the Q-table could be reduced by using function approximation techniques.while still representing the distinct value for each state - action pair. We can approximate the state - action value function (Q-function) as follows [6]:

$$\hat{Q}_k(s_k, a_k) = \sum_{l=1}^{n} \phi_l(s_k, a_k)\theta_l = \phi^T(s_k, a_k)\theta_k, \tag{2}$$

where $\phi : S \times A \to \mathbb{R}$ is a state and action - dependent basis function vector with n elements. Many techniques have been proposed to select appropriate $\phi^T(s_k, a_k)$. In this work, we employ two different approximation techniques called Fixed Sparse Representation (FSR), and Radial Basis Function (RBF) to map the original Q table to a parameter vector θ [9].

In FSR-QL, each element of function ϕ is defined by:

$$\phi_l(s_k, a_k) = \begin{cases} 1, & if\ s_k = S_i \in S, a_k = A_j \in A \\ 0, & otherwise. \end{cases} \tag{3}$$

For RBF-QL, each element of function ϕ is defined as:

$$\phi_l(s_k, a_k) = \begin{cases} e^{-\frac{s_k - c_l}{2\mu_l^2}}, & if\ a_k = A_j \in A \\ 0, & otherwise, \end{cases} \tag{4}$$

where c_l and μ_l are the center and radius of the RBF l, which has the shape of a Gaussian bell. Pseudo codes for FSR-QL and RBF-QL are presented in Algorithm 1, modified from [6].

We can calculate the space saved with FSR and RBF techniques for a problem with state space $S = D_x \cdot D_y$, where D_x, D_y are the dimension of x, y coordinates, respectively. We use $|.|$ to detnote the number of element of a vector. If we use FSR as in equation (3), both $\phi(s, a)$ and θ are column vectors of the size $(D_x + D_y)|A|$, which is much less than the space required in the original Q-table, which requires the size of $|S| \cdot |A| = (D_x \cdot D_y)|A|$. For instance, if the environment space is $S = 10 \times 10$, the original Q-table needs space to store $10^2 \cdot 4 = 400$ distinct values, while the total space to store the approximated parameter vector θ requires only $(10 + 10) \cdot 4 = 80$ distinct values. The saving ratio, that is $\frac{D_x + D_y}{D_x \cdot D_y}$, will grow as the state space grows. In RBF, we can potentially save even more space, since we can use a more limited, fixed number of basis functions to represent the state space. Figure 2 shows such a configuration where we use $n = 4$ RBF, with centers at $[2, 2]; [2, 4]; [4, 2]; [4, 4]$, and radius $\mu = 2$. The total space to store the approximated parameter vector θ requires only $n \cdot |A| = 4 \cdot 4 = 16$, much less than the parameter size in FSR-QL and original Q table size in QL.

V. SIMULATION

This section provides details of simulations on MATLAB for the UAV navigation problem. We defined the environment as a 5 by 5 board (Figure 2). Suppose that the altitude of the UAV was constant, the environment is discretized into 25 states, from $(1, 1)$ to $(5, 5)$. The UAV needs to find a goal, located at $(5, 5)$ but unknown to the UAV, from starting

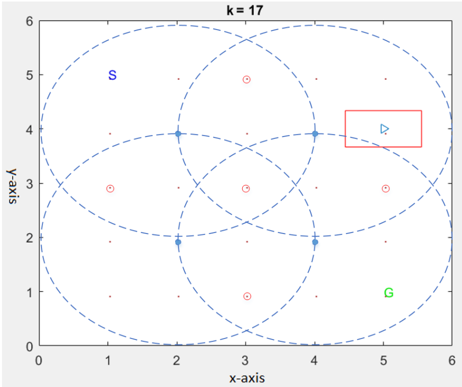

Fig. 2. The simulated environment at time step $k = 17$. Label S shows the original starting point, and Label G shows the goal.

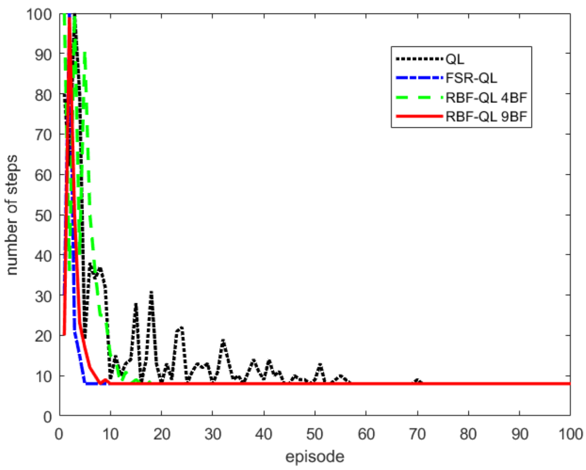

Fig. 3. The time steps taken over the learning course of each algorithm. RBF-QL with 9 RBF, denoted by red solid line, offered the most significant space reduction and very good time convergence speed.

position at $(1, 1)$ in the shortest possible way. The UAV can choose an action a_k from a set A: heading North, West, South or East in lateral direction. If the UAV selects an action that takes it out of the space border, it will stay still in the current state. The UAV can get maximum reward had it reached goal G, recognizable to the UAV using a specific landmark. Reaching all other places will result in a small penalty (negative reward):

$$r_{k+1} = \begin{cases} 100, & if\ s_{k+1} \equiv G \\ -1, & otherwise. \end{cases} \quad (5)$$

We ran four instances of the problem with different Q learning algorithms: QL, FSR-QL, RBF-QL with 4 basis functions, and RBF-QL with 9 basis functions. The first RBF-QL used 4 RBF with centers at $[2, 2]; [2, 4]; [4, 2]; [4, 4]$, and radius $\mu = 2$ (Figure 2). The second RBF-QL used 9 RBF with centers at $[2, 2]; [2, 4]; [4, 2]; [4, 4]; [1, 3]; [3, 3]; [5, 3]; [3, 1]; [3, 5]$. Each simulation used identical learning parameters: learning rate $\alpha = 0.1$, and discount rate $\gamma = 0.9$.

Figure 3 shows the resulting performance of the four learning instances. The black dotted line shows the number of steps in each episode it took the UAV using classical QL algorithm to reach the target. After 70 episodes of training the UAV to find the optimal course of actions (convergence) to reach the target from a certain starting position. The optimal number of steps the UAV should take was 8 steps, resulting in reaching the target in shortest possible way. The blue dashed line shows the number of steps for FSR-QL algorithm in each episode. It converges after only 5 episodes, much less than the original classical QL. The green dash-dot line shows the number of steps in each episode for RBF-QL with 4 RBF. Although it took more episodes (18) to converge than FSR-QL, it is still much less than the classical QL. The red solid line, that shows the number of steps in each episode for RBF-QL with 9 RBF, converged after 10 episodes, which is close to FSR-QL speed performance, while saving more space than the latter method. It can be noted that the RBF-QL performs better in convergence speed if more basis functions are used, since it would increase the accuracy of the approximation of the value function.

VI. Conclusion

This paper presented a performance comparison of different Q learning algorithms: QL, FSR-QL, and RBF-QL with different configurations that can help a UAV navigate to the target point in an unknown environment. The simulation showed that the Q learning with function approximation algorithms (FSR-QL, RBF-QL) outperforms the original QL in term of learning speed (convergence time). The RBF-QL offered the most significant space reduction. The result is not limited to the simple UAV navigation problem presented in this paper, and can be used in implementing RL algorithm for more important applications, such as wildfire monitoring, or search and rescue missions.

References

[1] H. M. La, W. Sheng, and J. Chen, "Cooperative and active sensing in mobile sensor networks for scalar field mapping," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 1–12, 2015.

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

[3] H. Bou-Ammar, H. Voos, and W. Ertel, "Controller design for quadrotor uavs using reinforcement learning," in *Control Applications (CCA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2130–2135.

[4] B. Zhang, Z. Mao, W. Liu, and J. Liu, "Geometric reinforcement learning for path planning of uavs," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[5] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for uavs with a suspended load," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4902–4909.

[6] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010, vol. 39.

[7] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, no. 3, pp. 207–222, 2002.

[8] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[9] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, J. P. How, *et al.*, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 6, no. 4, pp. 375–451, 2013.