Autonomous Navigation in Partially Observable Environments Using Hierarchical Q-Learning

Y. Zhou [*], E. van Kampen, and Q. P. Chu
Delft University of Technology, 2629HS Delft, The Netherlands

Abstract

A self-learning adaptive flight control design allows reliable and effective operation of flight vehicles in a complex environment. Reinforcement Learning provides a model-free, adaptive, and effective process for optimal control and navigation. This paper presents a new and systematic approach combining Q-learning and hierarchical reinforcement learning with additional connecting Q-value functions, which separate the effect of internal rewards from the external rewards. An online navigation algorithm with both Q-learning and Hierarchical decomposition is provided and applied to an illustrative task in a complex, partially observable environment. The environment is completely unknown at the beginning, and the agent learns the most efficient path online to avoid obstacles and to get to the target area. The present work compares the results using 'flat' Q-learning and hierarchical Q-learning. The results indicate that hierarchical Q-learning can help to accelerate learning, to solve 'curse of dimensionality' in complex navigation tasks, to naturally reduce the uncertainty or ambiguity at higher levels, and to transfer the learning results within tasks and across tasks efficiently. This proposed method can potentially design a near-optimal controller hierarchically for autonomous navigation without a-prior knowledge of the environment.

1 Introduction

In recent years, many studies have implemented Reinforcement Learning (RL) controllers to solve nonlinear, optimal control problems. Reinforcement learning is learning what actions to take to affect the state and to maximize the numerical reward signal by interacting with the environment, and to maximize expected future rewards ultimately. This method links bio-inspired artificial intelligence techniques to the field of control to overcome some of the limitations and problems in many control methods. Nevertheless, traditional RL methods solving the optimality problem is an off-line method, assuming that the state and environment is fully observable and that the observed states obey Markov processes. These methods, especially those applied to flying vehicles, are rendered intractable by 1) the nonlinearity of the unknown system, 2) the complexity of the task and environment, and 3) the partial observability of the system and environment. This paper focus on the last two problems induced by the complex and partially observable environment, which are common challenges in navigation.

Reinforcement learning often encounter the exponentially growing of states caused by the complexity of the environment and decision making, which is called the 'curse of dimensionality'. Many RL algorithms apply approximate dynamic programming (ADP), which uses a universal approximator with parameters to approximate the cost/value function, so that they can be used to solve the optimality problems with large or continuous state spaces online [1, 2, 3, 4]. However, with current 'flat' methods, the number of the parameters will still grow with the exponentially growing of the states and actions. Recent research attempts to deal with the 'curse of dimensionality' with hierarchical decomposition, which has always been a natural approach to problem solving [2, 5]. Hierarchical methods replace state-to-action mapping by a hierarchy of temporally abstract actions which operate over several time steps. Therefore, a complex problem may be solved by decomposing it into some smaller and simpler problems. Hierarchical decomposition speeds up learning for complex tasks more efficiently. Additionally, the experience and learning results gained during the learning can be generalized within not only a task and but also across tasks [6].

In the real world, the agent might not have perfect perception of the states or the environment [7]. The framework dealing with Partially Observable Markov Decision Process (POMDP) problems and deciding how to act in partially observable environments has been developed especially for these situations and remains an active area of research [8, 9, 10]. For instance, Nominal Belief-state Optimization (NBO) [9, 10] which combines application-specific approximations and techniques within the POMDP framework produced a practical design that coordinates the plants in the presence of occlusions. Another method, called the Posterior Belief Distribution (PBD) [11] , has been proposed to calculate the posterior distribution over beliefs after a sequence of actions. It is an online, forward-search algorithm to control a flying vehicle in a target monitoring task by evaluating the expected reward of a sequence of primitive actions, called 'macro-actions'. Other than that, Output-Feedback (OPFB) approximate dynamic programming algorithms [4, 12] have

[*] Email address(es): Y.Zhou-6@tudelft.nl

been proposed, as opposed to full state feedback, to tackle problems without direct state observation. POMDP methods act based on the estimation of the hidden variables and are theoretically more powerful. However, most existing work studies 'flat' models and cannot completely circumvent the partially observability of the environment. Hierarchical methods allow hidden variables to be estimated at different levels, and naturally reduce the uncertainty at higher levels.

This paper begins with a brief introduction to Markov and semi-Markov decision processes. Then, we present a hierarchical reinforcement learning (HRL) approach for autonomous navigation with Q-learning, which can help to systematically accelerate learning and to solve 'curse of dimensionality' in complex navigation tasks online. In section 4, the proposed method is applied to an illustrative navigation task with a discrete state space, and the results of using 'flat' Q-learning and hierarchical Q-learning are compared and discussed, showing how much the hierarchical method can improve the performance. The last part concludes the advantages and disadvantages of using the hierarchical method in this paper, and addresses the challenges and possibilities of the future research.

2 Markov Decision Processes and Semi-Markov Decision Processes

Reinforcement Learning is defined not by characterizing learning methods, but by characterizing a learning problem which can be described as an optimal control problem of Markov Decision Processes (MDPs) [13, 14]. MDPs are used to provide a mathematical framework to model decision making in situations where their outcomes are partly random and partly under the control of a decision maker [15]. MDP with finite state and action spaces is called a finite MDP and most of the modern theory of RL is restricted to finite MDPs [13]. A particular finite MDP is defined by its state and action sets, $\mathcal{S}$ and $\mathcal{A}$, and by the one-step dynamics of the environment. With any state s and action a, the probability of each possible next state s' can be obtained by transition probabilities $\mathcal{P}^a_{ss'}$:

$$\mathcal{P}^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a_t = a\}. \tag{1}$$

With any current state s, action a and any next state s', the expected value of the next reward can be given as follow:

$$\mathcal{R}^a_{ss'} = E\{r_t | s_t = s, a_t = a, s_{t+1} = s'\}, \tag{2}$$

where r_t denotes the immediate reward of the next state s' after taking a possible action a. These qualities completely specify the most important aspects of the dynamics of a finite MDP.

RL approach uses the concepts of a dynamical system's state and of a value function under certain policy π to define a functional equation, now often called the Bellman equation [13]. By modifying the current policy π and the state value function V or state-action value function Q, RL methods make them tend to the optimal policy π^* and the optimal value function, V^* or Q^*. For a specific problem, there is always one or more optimal policies which are better than or equal to all other policies. Because it is the optimal value function, its consistency can be written in a special form without reference to any specific policy. Thus, the Bellman optimality equation for state value function, V^*, is shown as follows:

$$\begin{aligned} V^*(s) &= \max \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \\ &= \max_{a \in \mathcal{A}} E_{\pi^*} \left\{ r_t + \gamma V^*(s') | s_t = s, a_t = a \right\}, \end{aligned} \tag{3}$$

where $\gamma \in [0, 1]$ is a parameter called the *discounted rate*. This equation also provide the relationship between the value of the current state $V^*(s)$ and its possible successor states $V^*(s')$.

Similarly, the state-action value function, denoted $Q^\pi(s, a)$, is defined as the expected return starting from s, taking the action a, and thereafter following policy π. And the corresponding Bellman optimality equation, Q^*, is shown as follows:

$$\begin{aligned} Q^*(s, a) &= E\{r_t + \gamma \max_{a' \in \mathcal{A}_{s'}} Q^*(s', a') | s_t = s, a_t = a\} \\ &= \mathcal{R}^a_{ss'} + \gamma \sum_{s'} \mathcal{P}^a_{ss'} \max_{a' \in \mathcal{A}_{s'}} Q^*(s', a'). \end{aligned} \tag{4}$$

An MDP defines only the sequential decision process rather than the time between one decision and the next [5]. The semi-Markov decision process (SMDP) is a generalization of MDP. An SMDP defines the time τ as the remaining/waiting time in the current stage. The amount of time can be either real valued in continuous-time discrete-event systems or integer valued in discrete-time systems, which is used in this paper. When action a is executed in the current state s_t, the joint transition probabilities of the next state s_{t+1} occurring after τ is written as $P(s_{t+1}, \tau | s_t, a)$ [5]. The Bellman optimality equation for V^* is

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}^a_{ss'} + \sum_{s', \tau} \gamma^\tau P(s', \tau | s, a) V^*(s') \right\}, \tag{5}$$

and the Bellman optimality equation for Q^* is

$$\begin{aligned} Q^*(s, a) &= \mathcal{R}^a_{ss'} \\ &\quad + \sum_{s', \tau} \gamma^\tau P(s', \tau | s, a) \max_{a' \in \mathcal{A}_{s'}} Q^*(s', a'). \end{aligned} \tag{6}$$

3 Hierarchical Q-learning

Navigation for mobile robots in known or small-scaled environment has been well studied. However, in practice, mobile robots often need to explore an initially unknown environment with limited sensors in complex autonomous navigation tasks. With state space decomposition into smaller, lower level state spaces, the task can be divided into smaller and easier tasks, where 'flat' algorithms can be used.

3.1 *Q-learning*

Q-learning is an off-policy temporary difference (TD) method. The learning of the state-action value function Q, which directly approximates the optimal state-action value function Q^* of the *estimation policy*, is independent of the *behaviour policy* [13]. This separation allows exploration with the behaviour policy and simplifies the analysis of the algorithm. In this paper, we use one-step 'flat' Q-learning algorithm to update state-action values in discrete form with immediate rewards r_t, as follows:

$$\begin{aligned} Q_{k+1}(s,a) &= (1-\delta)Q_k(s,a) \\ &+ \delta\left[r_t + \gamma \max_{a' \in \mathcal{A}_{s'}} Q_k(s',a')\right], \end{aligned} \tag{7}$$

where δ is a *learning-rate parameter*.

Q-learning can be applied to SMDP with action a executed in the current state s and immediate rewards r_{t+i} during the waiting time τ. In the discrete-time system, the Q-learning updates the estimated state-action value $Q_t(s,a)$ of the optimal $Q^*(s,a)$ as follows [5]:

$$\begin{aligned} Q_{k+1}(s,a) &= (1-\delta)Q_k(s,a) \\ &+ \delta\left[R_{t,\tau} + \gamma^\tau \max_{a' \in \mathcal{A}_{s'}} Q_k(s',a')\right], \end{aligned} \tag{8}$$

where $R_{t,\tau}$ is the accumulated reward during the waiting time:

$$R_{t,\tau} = r_t + \gamma r_{t+1} + \cdots + \gamma^{\tau-1} r_{t+\tau-1}. \tag{9}$$

SMDP Q-learning has 3 significant advantages [13, 5, 16]. First, it uses iterative data samples from the distributions obtained from the real world test or produced stochastic simulations, rather than access to the explicit knowledge of the expected rewards or the state-transition probabilities. This makes SMDP Q-learning can be used as a model-free RL method. Second, SMDP Q-learning uses state-action values. Thus, in discrete-event systems, finding optimal actions does not require one-step ahead search or accessing to the one-step action models, which is often difficult. Third, it is possible to store action-values for every state-action pairs for small-scale problems. And it is easy to extend this advantage to large-scale problems by using function approximation methods and/or hierarchical reinforcement learning methods.

3.2 *Decomposition and Hierarchies*

For large-scale control and navigation problems, decomposition of tasks and abstraction of actions allow systems to solve current sub-problems and to ignore irrelevant details at current level. Each higher level uses a partial description of the environment, which can partition the environment into *sub-environment* or *macro states*. This feature may naturally reduce the uncertainty or ambiguity induced by partial observability of the environment. The activities or decisions in higher levels are called *macro actions* or *behaviours*, which instruct the policy in lower level. The macro action spaces and values are dependent on the current macro state decomposed in their levels. The space of the original one-step actions for each state, which are called *primitive actions*, may stay the same or partially admissible depending on the higher level macro action.

Macro action b is defined over its input set, macro states m, and the current high level policy μ similarly to primitive actions, and additionally, a termination condition $\beta : \{0, 1\}$. Correspondingly, the Q-learning algorithm updates the estimation of the lower level state-action value with lower level action a executed in the current state s, and following a higher level macro action b until b terminates after τ steps:

$$\begin{aligned} Q_{k+1}(s,a,b) &= (1-\delta)Q_k(s,a,b) \\ &+ \delta\left[R_{t,\tau} + r_{t+\tau,b} + \gamma^\tau \max_{a' \in \mathcal{A}_{s'}} Q_k(s',a',b')\right], \end{aligned} \tag{10}$$

where $R_{t,\tau}$ is the accumulated lower level reward during the waiting time (see Eq.9), and $r_{t+\tau,b}$ is the higher level reward returned after the termination of b, which is executed at current macro state m. All the states visited before b terminates belongs to the current macro state m.

With decomposition of the environment, the primitive actions and macro actions can be evaluated and updated separately. In real applications, the state may be partially observable. With hierarchical algorithms, the higher level may collect some information and estimate other state features. With those similar features, the lower level states can be clustered together and constitute a macro state. In the one-step level, given a higher level action b, the Q-learning algorithm may keep its 'flat' form and assign state-action values with partially observable features:

$$\begin{aligned} Q^{i,b}_{k+1}(s,a) &= (1-\delta)Q^{i,b}_k(s,a) \\ &+ \delta\left[r_{s',t} + r_{b,t} + \gamma \max_{a' \in \mathcal{A}_{s'}} Q^{i,b}_k(s',a')\right], \end{aligned} \tag{11}$$

where i denotes that this is ith level Q-value following a $(i-1)$th level instruction b, the immediate reward $r_{s',t}$ is partially observable features related, and $r_{b,t}$ is the reward of accomplishment of following macro action b. With this Q-value $Q^b_{k+1}(s,a)$, the system can greedily choose a primitive action following the higher level instruction: taking 'macro action' b.

In higher levels, the system choose a 'macro action' b in current macro state m and evaluates and updates the macro state-action values. If the system has a partial observability, macro states m can be estimated states, or *belief macro states*. The Q-learning algorithm only considers the high level macro states m:

$$\begin{aligned} Q^j_{k+1}(m,b) &= (1-\delta)Q^j_k(m,b) \\ &+ \delta\left[r_{m',T} + \gamma \max_{b' \in \mathcal{B}_{m'}} Q^j_k(m',b')\right], \end{aligned} \tag{12}$$

where j denotes that this is jth level Q-value considering high level rewards, $r_{m',T}$ is the immediate reward when the system reaches the next macro state m', and T denotes the time, $t+\tau$ in Eq.10, when b terminates. It only updates after the termination of the macro action b.

3.3 Strategy Connecting Hierarchies

With good assigned rewards, Eq.11 and 12 can already be used to form the hierarchical Q-learning algorithm. However, when the system changes or the formulation of hierarchy changes, the learned policy may not be reused. To make this algorithm more flexible, transferable and reasonable, the lower level Q-learning algorithm, Eq.11, can be further divided into a 'flat' Q-learning algorithm and a connecting Q-learning algorithm. This division separates the external reward $r_{s',t}$ and the internal reward $r_{b,t}$ into different state-action value functions. Additionally, the discount factors for the external rewards and for internal rewards can be assigned independently as well.

In 'flat' RL algorithm, external rewards are usually used. External rewards are positive or negative outcomes from the external reward system or other environmental sources. They are tangible, such as consuming energy or reaching a charger in a robot navigation task. On the other hand, HRL algorithms often require assigned internal rewards, which are intangible and come from the sense of performance itself, such as accomplishment of following a higher level instruction or achievement of the sub-goal. The internal reward can either be a reward observed immediately after taking action a in the current state s, or be a delayed reward returned after termination of the macro action b. Assigning suitable internal rewards is difficult. It can affect the evaluation of the actions in each state following an instruction.

In this paper, a delayed internal reward evaluating the accomplishment of following the higher level macro action b is used. The internal rewards $r_{b,t}$ in each episode are assigned a same value $r_{b,T}$. Thus, the 'flat' lower level Q-learning algorithm is

$$\begin{aligned} Q^i_{k+1}(s,a) &= (1-\delta)Q^i_k(s,a) \\ &\quad + \delta\left[r_{s',t} + \gamma_{E,i}\max_{a'\in\mathcal{A}_{s'}} Q^i_k(s',a')\right], \end{aligned} \tag{13}$$

where i indicates that this is ith level 'flat' Q-value only considering the ith level external reward, and $\gamma_{E,i}$ is the discounted factor for the ith level external rewards. The connecting Q-learning algorithm is

$$\begin{aligned} Q^i_{k+1}(s,a,b) &= (1-\delta)Q^i_k(s,a,b) \\ &\quad + \delta\left[\gamma^{\tau}_{I,i}\cdot r_{b,T} + \gamma_{I,i}\max_{a'\in\mathcal{A}_{s'}} Q^i_k(s',a',b)\right], \end{aligned} \tag{14}$$

where i indicates that this is the additional ith level Q-value taking internal reward into account when following $(i-1)$th level macro action b, $\gamma_{I,i}$ is the discounted factor for the ith level internal rewards, and τ is the waiting time with $\tau = T - t$. Noted that this lower level Q-learning algorithm in Eq.13 is in the same form as the higher level Q-learning algorithm in Eq.12. This makes the hierarchical Q-learning algorithm systematic and reusable.

To evaluate and choose an action in current state s following a higher macro action b, we can combine the 'flat' lower level Q-value $Q^i_{k+1}(s,a)$ and the connecting Q-value $Q^i_{k+1}(s,a,b)$, as follows:

$$Q^{i,b}_{k+1}(s,a) = Q^i_{k+1}(s,a) + \omega_i \cdot Q^i_{k+1}(s,a,b), \tag{15}$$

where ω_i adjusts the weight of the internal rewards indicating the ith level accomplishment. Note that the state-action value in Eq.11 and 15 can be different even with $\gamma_E = \gamma_I$ and $\omega_i = 1$, because the maximum state-action values in Eq.13 and 14 for the next state s' are calculated separately. Comparing to Eq.11, this new estimated Q-value $Q^{i,b}_{k+1}(s,a)$ in Eq.15 with an on-line adjustable weight ω_i makes the evaluation process more flexible.

An n level hierarchical Q-learning algorithm consists of a primitive level Q-learning algorithm (Eq.13), $(n-1)$ higher level Q-learning algorithms (Eq.12), and $(n-1)$ connecting Q-learning algorithms (Eq.14). This hierarchical algorithm updates each level state-action value function independently, and evaluates the action in a state following a higher level instruction with a connecting strategy. Additionally, this algorithm separates external and internal rewards in different state-action value functions, which makes adjusting the importance of the internal rewards online possible and more flexible. These features make this hierarchical Q-learning algorithm more systematic, expandable, and easy to apply to real applications.

4 Application

4.1 Problem Setup

This section will present an illustrative application of the hierarchical Q-learning algorithm: an indoor discretized navigation problem for a flying robot in a maze. Similar to Parr's maze problem [17], this maze has about 3600 *absolute primitive states* and a target area, as shown in Fig.1. However, this task is more complex and realistic with partial observability, different starting locations, and limited mobility. It actually has to perform two tasks at the same time: avoiding the obstacles, and reaching the target area as soon as possible.

As depicted in Fig.2, in this indoor task, the flying robot only has 3-step short-sight (obstacle in the 1, 2, 3-step away or nothing) in 3 directions (front, left, and right) and knows its heading (north, south, east, and west). Thus, this robot may only have 256 *observed primitive states*, which induces ambiguity in this complex task. This robot has limited mobility: turn left, turn right, and move forward. Taking any of the primitive action will consume the energy and result in a penalty '-1'. If the robot moves towards an obstacle when it

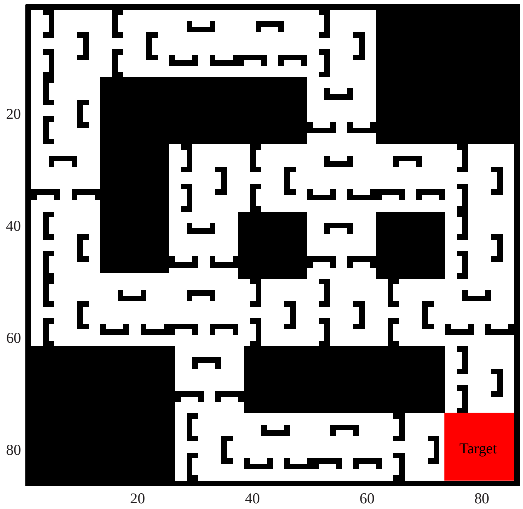

Figure 1: A POMDP maze problem with obstacles and a target.

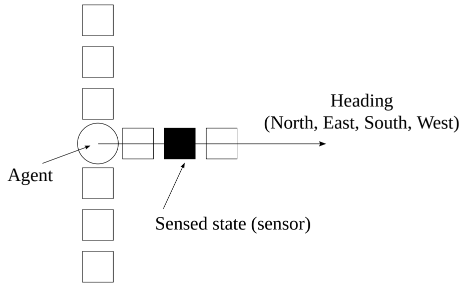

Figure 2: A primitive state of the flying robot.

is next to the obstacle, it will automatically turn around and result in a penalty '-3'. When it reaches the target area, the robot will land at this area and get a terminal reward '10'. This flying robot does not know the map of this maze in advance, but has an internal memory of its trajectory, which will be removed before another episode. This task focuses on the high-level autonomous navigation and online path planning rather than the system model or low-level control. We assume that the flying robot has the ability to perform the primitive actions.

4.2 Hierarchical Q-learning applied to Navigation

By using 'flat' Q-learning, the flying robot may only learn to avoid the obstacles, because it will not know its correct absolute position. To get to the target as soon as possible, it need to explore in the maze, to find its possible position, and to determine a proper direction from a higher level. This is how the hierarchical Q-learning method can be applied to this task.

This hierarchical method has 2 levels. The 1st level agent follows the ultimate goal of reaching the target area. It memorizes and draws the observed pattern of the local map, and estimates its possible position, which is called *belief macro state* $\widehat{m}$, by comparing the pattern with a memorized map. Each macro state is a decomposed environment and contains a subset of the primitive states. After each episode, the macro state-action value function can be updated with following algorithm:

$$\begin{aligned} Q^1_{k+1}(m,b) &= (1-\delta)Q^1_k(m,b) \\ &+ \delta\left[r_{m',T} + \gamma_{E,1}\max_{b'\in\mathcal{B}_{m'}} Q^1_k(m',b')\right], \end{aligned} \tag{16}$$

where T denotes it is the Tth (m,b) macro state-action pair, and $r_{m',T}:\{0,10\}$ is the immediate reward when the system reaches the next macro state m'. With the belief state $\widehat{m}$ and the state-action value function $Q^1_{k+1}(\widehat{m},b)$, the agent in the 1st level will give an instruction, macro action $b:\{$ go north, south, east, west $\}$, for the 2nd level agent.

The 2nd level agent follows the higher level instruction b by using the connecting Q-value updated by

$$\begin{aligned} Q^2_{k+1}(s,a,b) &= (1-\delta)Q^2_k(s,a,b) \\ &+ \delta\left[\gamma^{\tau}_{I,2}\cdot r_{b,T} + \gamma_{I,2}\max_{a'\in\mathcal{A}_{s'}} Q^2_k(s',a',b)\right], \end{aligned} \tag{17}$$

with a delayed internal reward $r_{b,T}:\{0.5,-0.5\}$, which assigns a reward for going the right direction and a penalty for going a wrong direction or not moving. In this equation, τ is the waiting time from current time t till the termination of the macro action b at time T: $\tau = T - t$. At the same time, it accomplishes the task in its own level, avoiding obstacles, using the primitive state-action value function updated by

$$\begin{aligned} Q^2_{k+1}(s,a) &= (1-\delta)Q^2_k(s,a) \\ &+ \delta\left[r_{s',t} + \gamma_{E,2}\max_{a'\in\mathcal{A}_{s'}} Q^2_k(s',a')\right], \end{aligned} \tag{18}$$

where $r_{s',t}:\{-1,-3\}$ is the immediate reward when the system reaches the next state s'.

4.3 Result and Discussion

Two algorithms are applied to this navigation problem: a 'flat' Q-learning algorithm and the hierarchical Q-learning algorithm presented in this paper. Fig.3 compares the numbers of primitive actions taken in each iteration for the 'flat' Q-learning and hierarchical Q-learning algorithm. With 'flat' Q-learning, the agent learns how to prevent from collision in the first several iterations. However, the performance will not be improved significantly hereafter. There are two main reasons: first, the huge number of actions taken before reaching the target area makes the effect of the terminal reward propagate very slow to the experienced states; and second, the partial observability leads to the ambiguity in real absolute states, which prevents the convergence of observed state-action values. These features impair the agent's learning ability of performing the second task: reaching the target area as soon as possible.

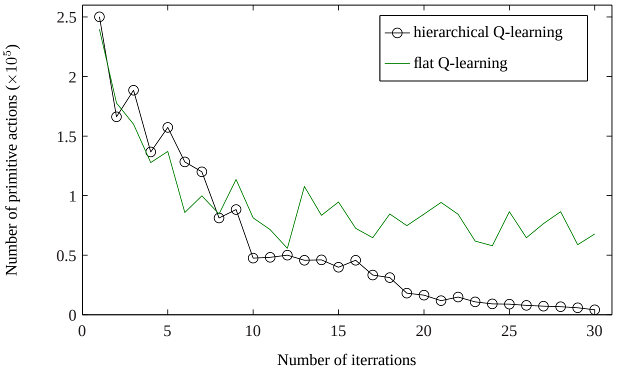

Figure 3: The number of primitive actions taken in each iteration with a 'flat' Q-learning and a hierarchical Q-learning algorithm (averaged of 5 runs).

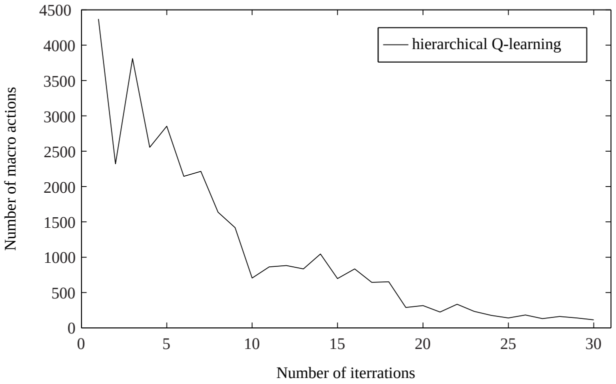

Figure 4: The number of macro actions taken in each iteration with the hierarchical Q-learning algorithm (averaged of 5 runs).

On the other hand, the hierarchical Q-learning algorithm improves the performance of both the collision avoidance task and searching for the target area task. The low level agent focus on avoiding obstacles and following high level's instruction. The high level agent focus its exploration of the state space and finds the optimal macro actions to get to the target area as soon as possible. In Fig.4, there is a clear trend of decreasing number of the macro actions taken during learning. This result indicates that the agent improves the performance also from a higher level. The hierarchical Q-learning algorithm appears to improve the performance considerably and much faster. Even with more iterations and longer learning time, the 'flat' Q-learning method can not improve the navigation performance as much as the hierarchical Q-learning method.

This application demonstrates the ability of the hierarchical Q-learning to perform a navigation problem with partial observability. Fig.5 presents the greedy macro actions learned by the high level hierarchical Q-learning in the memorized map. The grid with pink dotted line decomposes the environment into macro states. Since each macro state consists of a fixed size of area, the corridor area is not completely separated from the inaccessible area with walls with the environment decomposition. Some macro states can be partly corridor and partly inaccessible area with walls, which also have been experienced and have greedy macro actions. The agent created this map by stitching the constructed maps from the memory after each episode. The light gray states indicate that those areas haven't been experienced or observed. The dark gray states are experienced target states and are clustered with a red triangle.

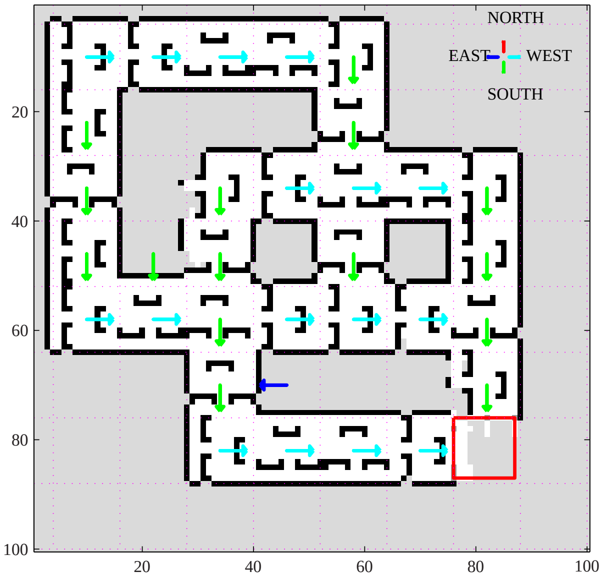

Figure 5: The greedy macro actions in the memorized map after 15 iterations with hierarchical Q-learning.

These results show that the state-action values is converging with experiences. With more learning episodes, the agent will explore all possible actions and exploit those experience to learn to follow higher level instructions and to accomplish the task as good as possible. Thus, the memorized map will be more complete, and the state-action value functions will all converge. The agent has the ability to find the hierarchical optimal path online from any initial position.

Furthermore, in a new high level task, such as searching for a different target area, or a new environment, such as an expanded maze, the agent can still use the low level and connecting Q-learning results, and only need to learn a new higher level policy. The experience and learning results during the learning can be transferred across tasks. This feature also speeds up learning for complex tasks more efficiently, and makes this present hierarchical Q-learning a transfer learning method.

5 Conclusion

This paper proposes a systematic hierarchical Q-learning method and applies it to a complex navigation task in a partially observable environment. This method combines the advantages of both the 'flat' Q-learning method and the hierarchical structure. The results indicate that the present hierarchical Q-learning method can help to accelerate learning, to solve 'curse of dimensionality' in a complex autonomous navigation task, to naturally reduce the uncertainty or ambiguity at higher levels, and to transfer the experience and learning results within and across tasks efficiently.

This paper investigates the complexity and partial observability of the task and environment in navigation tasks, but not yet the nonlinear, unknown system of flying vehicles. In the future, we will apply this method to a more realistic micro aerial vehicle control problem by further decomposing the complex flight task into high-level navigation tasks and low-level control tasks, such as reference tracking. In low-level control, the problem of nonlinearity of the unknown system can be solved by using other 'flat' methods such as incremental approximate dynamic programming. Thus, three of the common problems in flight control mentioned at the beginning of this paper, which are 1) the nonlinearity of the unknown system, 2) the complexity of the task and environment, and 3) the partial observability of the system and environment, can be solved by using the hierarchical reinforcement learning combining different 'flat' algorithms in each level.

References

[1] Said G Khan, Guido Herrmann, Frank L Lewis, Tony Pipe, and Chris Melhuish. Reinforcement learning and optimal adaptive control: An overview and implementation examples. *Annual Reviews in Control*, 36(1):42–59, 2012.

[2] Jennie Si. *Handbook of learning and approximate dynamic programming*, volume 2. John Wiley & Sons, 2004.

[3] Y. Zhou, E. van Kampen, and Qi Ping Chu. Incremental approximate dynamic programming for nonlinear flight control design. In *Proceedings of the EuroGNC 2015*, 2015.

[4] Y. Zhou, E. van Kampen, and Qi Ping Chu. Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback. In *AIAA Guidance, Navigation and Control Conference*, 2016.

[5] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[6] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[7] Olivier Sigaud and Olivier Buffet. *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[8] Alex Brooks, Alexei Makarenko, Stefan Williams, and Hugh Durrant-Whyte. Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems*, 54(11):887–897, 2006.

[9] Zachary A. Harris Scott A. Miller and Edwin K. P. Chong. A POMDP framework for coordinated guidance of autonomous uavs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009.

[10] Shankarachary Ragi and Edwin K. P. Chong. UAV path planning in a dynamic environment via partially observable markov decision process. *IEEE Transactions on Aerospace and Electronic Systems*, 49(4):2397–2412, 2013.

[11] Ruijie He, Emma Brunskill, and Nicholas Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40(1):523–570, 2011.

[12] Frank L Lewis and Kyriakos G Vamvoudakis. Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using measured output data. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(1):14–25, 2011.

[13] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[14] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[15] Mohamed Rida, Hicham Mouncif, and Azedine Boulmakoul. Application of markov decision processes for modeling and optimization of decision-making within a container port. In *Soft Computing in Industrial Applications*, pages 349–358. Springer, 2011.

[16] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[17] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.