Article

Advanced Double Layered Multi-Agent Systems Based on A3C in Real-Time Path Planning

Dajeong Lee, Junoh Kim, Kyungeun Cho * and Yunsick Sung *

Department of Multimedia Engineering, Dongguk University-Seoul, Pildong-ro 1-gil, Jung-gu, Seoul 0460, Korea; djmstar@mme.dongguk.edu (D.L.); junoh.kim@dongguk.edu (J.K.)

* Correspondence: cke@dongguk.edu (K.C.); sung@dongguk.edu (Y.S.);

Tel.: +82-2-2260-3834 (K.C.); +82-2-2260-3338 (Y.S.)

Abstract: In this paper, we propose an advanced double layered multi-agent system to reduce learning time, expressing a state space using a 2D grid. This system is based on asynchronous advantage actor-critic systems (A3C) and reduces the state space that agents need to consider by hierarchically expressing a 2D grid space and determining actions. Specifically, the state space is expressed in the upper and lower layers. Based on the learning results using A3C in the lower layer, the upper layer makes decisions without additional learning, and accordingly, the total learning time can be reduced. Our method was verified experimentally using a virtual autonomous surface vehicle simulator. It reduced the learning time required to reach a 90% goal achievement rate by 7.1% compared to the conventional double layered A3C. In addition, the goal achievement by the proposed method was 18.86% higher than that of the traditional double layered A3C over 20,000 learning episodes.

Citation: Lee, D.; Kim, J.; Cho, K.; Sung, Y. Advanced Double Layered Multi-Agent Systems Based on A3C in Real-Time Path Planning. *Electronics* **2021**, *10*, 2762. <https://doi.org/10.3390/electronics10222762>

Academic Editor: Aldo Franco Dragoni

Received: 18 September 2021
Accepted: 9 November 2021
Published: 12 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: asynchronous advantage actor-critic; multi-agent system; simulation framework

1. Introduction

Recently, multi-agent systems have enabled the control of diverse types of agents in autonomous unmanned systems, such as unmanned autonomous land vehicles, aerial vehicles, and surface vehicles [1–3]. For example, autonomous vehicles are commonly controlled by end-to-end control systems, which receive captured images as input directly analyzed by a convolutional neural network (CNN) [4,5]. Unmanned aerial vehicles fly according to waypoints generated by an autoencoder-based approach [6]. By defining the waypoints, it is possible to autonomously control UAVs.

Given that each agent in a multi-agent system applies deep learning independently, both the complexity of deep learning and the learning time are increased. Deep learning should also consider the relationship among agents when agents select and execute actions. Deep Q-networks (DQNs) [7] were introduced to apply deep neural networks, which enables the application of reinforcement learning in such complex environments. However, machine learning remains limited in applications considering the relationships among agents to determine optimal actions. To this end, hierarchical approaches can be applied to multi-agent environments [8–10]. Two types of hierarchical approaches have been developed. First, within an agent, a hierarchical structure, which includes a meta-controller and controller, exists [8]. The meta-controller assigns goals to maximize extrinsic rewards, and the controller determines actions to maximize intrinsic rewards. Next, for multiple agents, a hierarchical structure is developed for low-level and high-level skills [9]. High-level skills are determined considering high-level policies of multiple agents, and then low-level skills are determined considering each agent. Therefore, two-step learning processes and execution processes are required for both skills, which increases the learning time and involved costs. Given that a high- and low-level skill-based approach requires

agents to select skills considering the relationships among themselves, it is necessary to reduce the learning problem.

This paper proposes an advanced double layered asynchronous advantage actor-critic (A3C) [11,12] for multi-agent systems. This method consists of an upper layer and a lower layer and actions are selected by A3C. The upper layer determines the A3C spatial state space of the lower layer based on the values of the lower layer. The lower layer selects actions not considering all states in the lower layer by A3C, but only considering the determined spatial state space of the lower layer, which reduces the amount of A3C state space.

We conducted research in the simulation stage to apply multi-agent system-based pathfinding to an actual marine environment. The 3D space of 40×40 km was converted into a 2D grid. Each agent must reach its final goal in the 2D grid space while avoiding the enemy. Pathfinding is used to reach the goal from the starting point, and a wide range of features are considered, such as topography, friendly ships, enemy ships, and fishing boats. When the A* algorithm [13], one of the representative algorithms in pathfinding, was applied to our experimental environment, only 20% of the total number of agents found the path successfully owing to the amount of state space. Therefore, non-machine learning pathfinding methods such as A* were not suitable as methods for pathfinding in our experimental environment. Currently, in a situation where the ratio of allies to enemies is 3:3, the learning of each agent is conducted using a double layered A3C; however, in the future, we intend to increase the number of operating agents. There are training time problems in the traditional double layered A3C methods where parameters of all layers must be trained, and the success rate to the final goal is low. In order to expand the number of multi-agents in industry fields, it is necessary to shorten the learning time and improve the performance. This paper aims at reducing the learning time while improving the performance in a hierarchical structure.

The contributions of this paper are as follows:

- An approach that enables A3C to be utilized in multi-agent systems where the problem of state space will be invoked is proposed.
- When actions are determined by multilayered approaches, we suggest one approach where only the bottom layer needs to be learned, such as A3C, while other layers do not require any additional learning.
- Our proposed approach can reduce the calculation amount and learning time of A3C simultaneously.

The remainder of this paper is organized as follows. Section 2 describes the related approaches. Section 3 suggests a multilayer agent using the A3C method. Section 4 explains the results of experiments conducted to validate this method. Finally, Section 5 concludes the work and discusses future research.

2. Related Works

This section introduces approaches to solve the problems of state space and learning time, in a variety of reinforcement learning methods, including DQN. More specifically, this section describes a wide range of processes to reduce the state space and learning time.

Grid path-planning with deep reinforcement learning [14] has been applied to divide the state space into action and obstacle states. In detail, the visible area of an agent is expressed as an image, and the action path is derived by considering obstacles using convolutional and residual learning neural networks. This approach suggests a deep neural network of pathfinding problems by demonstrating a similar performance to A*, the traditional pathfinding algorithm, even with a deep neural network architecture.

Reinforcement learning has been used to determine the path-planning of multiple agents, including robots and autonomous systems. To obtain the shortest path in consideration of surroundings, including obstacles, 2D or 3D space can be converted into a grid space [15]. In general, a grid expresses each physical space of the same size. Reinforcement learning-based variable grids identify the optimum path by expressing the space around

obstacles with smaller grid cells and the space without obstacles with larger grid cells to reduce the state space. The state space is reduced by expressing a low-risk space without obstacles of a larger size. The reduction of state space reduces the learning subjects and, ultimately, the learning time.

An autonomous pathfinding method using Q-learning [16] analyzed the environment structure in the captured images and calculated the shortest path from the present state to the target state. As the captured image is internally analyzed and the path of action is expressed on the grid-based map, the optimum path is identified based on Q-learning. Although the authors suggested an enhanced approach for identifying the path using Q-learning and the image-based state, their method did not solve learning time and space issues.

An algorithm integrating the double deep Q-learning (DDQN) and gated recurrent units (GRU) in a recurrent neural network (RNN) system [17] demonstrated obstacles on the grid map, calculated the value per cell, and determined a path of action. Each agent state was expressed using eight surrounding cells, transferred to the GRU as input, and identified the optimum path in consideration of obstacle avoidance. This approach suggests the possibility of a state-processing function using a time-series algorithm.

Spatial reinforcement learning [18] was an approach to calculate the necessary value function by reflecting the visual analysis approach of a human being to achieve the goals in a grid environment with obstacles. The goal in a grid has a higher value in the area of interest and a lower value if there are any obstacles. This approach reduces the learning time by reducing the convergence time toward the optimum function value by applying the visual analysis approach of a human being.

The next-best-view (NBV) selection approach based on deep reinforcement learning (DRL) in a complicated state space [19] is a new search architecture developed to estimate the order of visits to unexplored lower areas using the DRL model step-by-step and to optimize a plan among areas using the NBV selection algorithm. The waypoint in the lower area is designated using the NBV selection algorithm, and an agent plans the local path from the present position to the waypoint using the A* algorithm. This approach enables decision-making in a complicated state space by dividing and processing a series of processes that determine a path in phase.

Given that transfer learning is one of the hot-issued algorithms, it is possible to apply transfer learning into machine learning, such as reinforcement learning [20]. It is possible for robots to learn new skills by transfer learning. However, even though the performance of the robots can be improved by the new skills considering the difference compared to the learnt environment, there is the limitation of utilizing transfer learning for reducing the learning time and the amount of state space.

Hierarchical reinforcement learning (HRL) [21] hierarchically expressed a state and implemented a path-planning model for a mobile robot. This method determined an action depending on the state hierarchically expressed by the sensing environment, identifying features, and optimizing the state-action function.

HRL with advantage-based auxiliary rewards [9] suggests adapting low-level skills into tasks by adjusting auxiliary rewards. The auxiliary rewards make agents learn high-level policy based on the advantage function of the high-level policy. However, there is still the limitation of reducing state space and learning time.

There is also interactive influence-based hierarchical reinforcement learning [22]. The approach interacts between low-level and high-level policies, given that the low-level policy is utilized by the high-level policy through the low-level policy representation. Therefore, the high-level policy is updated, reducing the dependence on the changes of the low-level policy. However, the additional learning of the high-level policy is still demanded.

In general, the larger and more complicated a grid space becomes in reinforcement learning, the higher the number of states to consider. While traditional reinforcement learning proposes a variety of approaches to solve such issues, the problems of learning

time and state space issues remain. In this paper, we suggest an approach to innovatively reduce the state space by hierarchically expressing the state and determining the actions.

3. Multilayered Agent Using A3C

This section describes a multilayered agent using A3C to solve the state-space problem of A3C. In particular, the relationship between the layers and the application process of A3C are explained. In detail, the operating methods of each layer and the loss function of an actor and critic are described.

3.1. Overview

When a real 3D space is expressed in a 2D grid space and the actions of pathfinding for an agent are determined by traditional A3C, a wider grid space implies a larger state space to be considered in A3C. The wider the state space expressed in the 2D grid space, the greater the number of state spaces to consider. Accordingly, an agent may fail to achieve a certain performance level, or even if an agent attains a certain performance level, excessive learning time may be involved. One approach to solve such issues is to divide a spatial state into several layers and make an agent learn each layer [9]. However, this approach has a disadvantage in that the total learning time is extremely long because an agent must learn every layer. Therefore, in this paper, we propose a hierarchical approach to divide a state space into upper and lower layers when the state space becomes larger and to direct an agent to learn only the lower layer and then use what it learns even in the upper layer.

Our method expresses only the spatial states among all states in the upper layer on a grid and determines the number of cells in the grid to be considered in the lower layer on the basis of the spatial state above. This paper does not consider a non-spatial state space. The space state is restructured into a 2D space state, and each state is expressed in a cell unit to fit to a grid. Therefore, this paper proposes an approach to solve the learning time issue of traditional double layered A3C, which can be applied to environments with multiple agents. It also contains two approaches, including upper and lower layers.

The upper and lower layers were processed as follows. The upper layer determines the cells to be considered in the lower layer using the value of the lower layer, to which A3C is applied without any additional learning. More specifically, the average of the surrounding cells in the lower layer corresponding to the cells of the upper layer to be selected is calculated. Using the calculated average as the value of the relevant upper layer, the cell of the upper layer with the highest average is finally selected. The lower layer uses the traditional A3C. Multiple local networks comprising A3C are trained simultaneously. Then, local networks update the weighed values of the global network through the link to the global network at every episode of learning. Based on the updated data, the local networks are updated. Finally, an agent's actions are determined using the weights of the global network.

An agent receives and performs a sub-goal and a goal per episode. A goal is defined as the final destination, and the sub-goals are the points that an agent passes to proceed to a goal. It is assumed that an agent passes the sub-goal in a sequence in accordance with a predefined sequence and finally arrives at a goal. When sub-goals are designated, an agent learns the travel distance between the sub-goals or between a sub-goal and a goal. Thus, less learning time is required compared to learning the path to a final goal in one step.

Figure 1 illustrates the 3:3 multi-agent environment. The upper layer expresses an actual total area of $40\text{ km} \times 40\text{ km}$ into a 2D grid with 40×40 cells. The lower layer divides the total area of $40\text{ km} \times 40\text{ km}$ into 30×30 cells for learning; 3×3 cells ($3\text{ km} \times 3\text{ km}$) in the upper layer corresponded to 10×10 cells ($1\text{ km} \times 1\text{ km}$) in the lower layer. In Figure 1a, the red agent adopts the method proposed in this paper. The blue agent is an opponent to compete against. The green dots denote sub-goals or goals. To consider the diversity in the path of action of a red agent, several agents are grouped, and the relevant path is indicated using sub-goals and a goal. A group for each agent was designated in advance for every episode. Each red agent also has its own sub-goals and a goal. In Figure 1a,

the green dot to which an agent finally arrives among the green dots is the goal. Other green dots represent sub-goals. When such an environment is presented, a red agent aims to arrive at the final goal by passing the predefined sub-goals and avoiding blue agents. The final action to be implemented is determined and executed by each red agent using this system.

Figure 1. Double layered A3C architecture: (a) sub-goal settings, (b) upper layer, and (c) lower layer.

For the application cases using the method proposed in this paper, the upper layer in Figure 1b selects the upper layer cells for a red agent to pass using the values of the lower layer. The cells on the $3 \text{ km} \times 3 \text{ km}$ topography in the lower layer were determined. As shown in Figure 1c, an agent determines the final cell to move to among the cells defined in the lower layer. An agent needs to update its location in the upper and lower layers to perform actions. When an agent travels a certain distance after the lower layer receives data from the upper layer, the upper layer renews the decision on the space state in the lower layer.

3.2. Greedy Algorithm-Based Upper Layer Approach

The upper layer determines the cells using the values of the lower layer from learning without separate learning. One cell in the upper layer is equivalent to 10×10 cells in the lower layer. Accordingly, when the value representing the cell in the lower layer is determined, it can be used in the upper layer cells. We also suggest an approach to calculate the representative value using several cells in the lower layer. The representative value is calculated in the lower layer cells corresponding to a total of nine cells, including the location of an agent and its surrounding locations on the basis of the cells in the upper layer where an agent is included. Since it is difficult to calculate the values of all cells for the lower layer, the values of the surrounding cells around the relevant cell are calculated to obtain the average, as shown in Figure 1c. At this point, the average is calculated after excluding the cells whose value is zero. Table 1 presents the symbols and notation in Section 3.2.

Table 1. Explanation of notations in Section 3.2.

Notations	Explanation
a^U_i	i th action in upper layer
s^L_j	j th state in lower layer
s^U_t	t th state in upper layer
$V_L(s^L_j)$	a value of upper layer at s^L_j
$V_U(s^U_t, a^U_i)$	a value of upper layer at (s^U_t, a^U_i)
$ S^L $	the number of cells in the outermost area among the cells in the lower layer related to action a^U_i

Equation (1) presents the value $V_U(s_t^U, a_i^U)$ when an agent performs the action a_i^U at state s_t^U in the upper layer. $|S^L|$ is the number of cells in the outermost area among the cells in the lower layer related to action a_i^U . $V_L(s_j^L)$ is the value at state s_j^L based on the j th cells in the lower layer:

$$V_U(s_t^U, a_i^U) = \frac{1}{|S^L|} \sum_{j=1} V_L(s_j^L), \quad (1)$$

where $s_t^{H'}$ is a next state of s_t^H after executing a_i^H , and s_j^L is in $s_t^{H'}$ and a boundary of $s_t^{H'}$. As shown in Formula (2), a_i^U , the highest value among the values $V_U(s_t^U, a_i^U)$ calculated in the upper layer is a_t^U , the final output of the upper layer. a_t^U determines the state space to be considered in the lower layer.

$$a_t^U = \underset{a_i^U}{\operatorname{argmax}} V_U(s_t^U, a_i^U) \quad (2)$$

3.3. A3C-Based Lower Layer Approach

A3C is an algorithm that updates the global network asynchronously by operating multiple local networks simultaneously. An actor calculates the probability $\pi(s, a)$ of an action “a” being performed at the state “s”. A critic evaluates the value of the relevant state $V(s)$. As multiple actor–critic agents work independently, they exchange learning results with the global network.

As shown in Figure 2, A3C is divided into one global network and multiple local networks. Each network interacts with the environment. Multiple local networks work simultaneously using the thread. The environment notifies the initial state, sub-goals, and goal of an agent during every episode. When one episode of learning is completed, each local network updates the policy and values to the global network and receives the updated policies and values from the global network to resume learning. The process above is performed asynchronously.

Figure 2. Structure of the A3C algorithm.

Algorithm 1 shows how local networks of A3C function and how they exchange weights with the global network. θ_p and θ_v are the parameters of the global network. θ_p is the policy of an actor, and θ_v is the value of a critic. Line 2 copies θ_p and θ_v onto the global network as θ_p' and θ_v' , the parameters of the local network, respectively. Θ is the policy value of the global network, and θ_v is the value of the global network. $d\theta_p'$ and $d\theta_v'$ are initialized to zero to calculate the gradient of those elements. The calculation above continues until one episode is completed or reaches “max_step”, as shown in Line 5. As shown in Line 6, the action a_t is determined by states s_t^L and θ_p' . In Lines 7 and 8, a_t is performed. After receiving the reward, r_t , the new state s_{t+1}^L , and the completion state of an episode ‘done’ value, one step is added. In Lines 9–12, when one episode is completed, R , θ_p' , and θ_v' are calculated using state s_i^L , action a_i , and reward r_i , recorded in all steps of an episode. Finally, θ_p and θ_v' values calculated in Line 13 are asynchronously updated to the global network.

Algorithm 1 Local Network Process in A3C

Input: global parameters θ_p and θ_v , state s_t^L , max_step, gamma γ
Output: global parameters θ_p and θ_v
1: **procedure** Local Network Process in A3C
2: $d\theta_p' \leftarrow 0, d\theta_v' \leftarrow 0, \theta_p' \leftarrow \theta_p, \theta_v' \leftarrow \theta_v$
3: step $\leftarrow 1$
4: done $\leftarrow$ false
5: **while** step < max_step or done **do**
6: $a_t \leftarrow \text{get_action}(s_t^L, \theta_p')$
7: $r_t, s_{t+1}^L, \text{done} \leftarrow \text{step}(a_t)$
8: step $\leftarrow$ step + 1
9: **for** i: = 0 $\rightarrow$ step **do**
10: $R \leftarrow \text{accumulate_reward}(r_i, \gamma, R)$
11: $\theta' \leftarrow \text{accumulate_gradient}(d\theta_p', \theta_p', R, a_i, s_i^L, \theta_v')$
12: $\theta_v' \leftarrow \text{accumulate_gradient}(d\theta_v', \theta_v', R, s_i)$
13: update_global_weight($\theta_p', d\theta_p', \theta_v', d\theta_v'$)

Figure 3 shows the actor and critic structure of A3C using the proposed method. Both the actor and critic state s^L as the input. s^L is divided into s_A^L and s_B^L . s_A^L indicates that the data on 15×15 cells around a red agent are available for transfer (0) or not (1) in the arrangement of (15, 15, 1). s_B^L calculates and indicates the distance to a blue agent and location, as well as a goal in the arrangement of (25). Of the 25 states, 10 states are related to a goal, and the other 15 states are related to a blue agent.

Figure 3. Structure of the A3C model: (a) structure of s^L , (b) structure of actor, and (c) structure of critic.

The number of actions that the lower layer can select is denoted as “ i ”. Assuming that $\pi_L(s^L_t, a^L_i)$ is the probability of selecting the action a^L_i at a certain state s^L_t , the action a^L_t determined by the lower layer is calculated as shown in Formula (3):

$$a^L_t = \underset{a^L_i}{\operatorname{argmax}} \{ \pi_L(s^L_t, a^L_1), \pi_L(s^L_t, a^L_2), \dots, \pi_L(s^L_t, a^L_i) \} \quad (3)$$

3.4. Loss Function of Lower Layer A3C

One network of A3C comprises the actor calculating policies and the critic calculating the actor and values. The actor and critic calculate losses individually. This section explains the loss function of the actor and critic.

The actor calculates and adds the cross entropy and entropy, as well as calculates the loss. s^L is the state, and a^L is the set of actions that can be selected. Each action is indicated as $a^L_1, a^L_2, a^L_3, \dots, a^L_n$. $\pi(s^L, a^L_i)$ is the probability of selecting a specific action a^L_i at a certain state s^L . $A(s^L, a^L_j)$ is an advantage, and $c = 1 \times 10^{-10}$. The cross-entropy, ce , is shown in Formula (4):

$$ce = - \sum_{j=1}^n A(s^L, a^L_j) \log \left(\sum_{i=1}^n a^L_i \pi(s^L, a^L_i) + c \right) \quad (4)$$

Entropy, e , is calculated as shown in Formula (5):

$$e = \sum_{i=1}^n \pi(s^L, a^L_j) \log \left(\pi(s^L, a^L_j) + c \right) \quad (5)$$

Finally, loss is determined as shown in Formula (6):

$$loss = ce + 0.01 \times e \quad (6)$$

The critic calculates the loss by squaring the difference between the actual value (Q) calculated as shown in Formula (7) and the expected value (V):

$$loss = \frac{1}{n} \sum_{i=1}^n \left(Q(s^L, a^L_i) - V(s^L) \right)^2 \quad (7)$$

Wherein, the actual value (Q) is calculated by adding the future value to the present value. The future value is calculated using the depreciation rate, as shown in Formula (8):

$$Q(s^L, a^L_i) = R + \gamma V(s'^L) \quad (8)$$

As specified above, the actual value can be acquired by reflecting the data on all steps of one episode, and finally, the loss of the critic is calculated.

4. Experiments

This section describes the entire system structure used for the experiment, including the development environment in which the learning and experiment are performed, as well as the hyperparameter used in the proposed method. First, the map for the experiment and cases are explained, and the implemented experiments are presented.

4.1. Whole System Structure

The agents in this system are classified as blue and red agents for competition. The method was applied to red agents, which acted as multiple agents. All red agents could perform a total of nine actions, including up, down, left, right, four diagonal directions, travel, and standby. The maximum speed of the red agents was 45 km/h. In contrast, the maximum speed of the blue agents was 25 km/h. A blue agent is slower than a red agent

but has a shooting function. A blue agent can shoot, and its shooting range is 1.8 km. All the agents have physical strength. The physical strength of a red agent is lost when it is shot. Upon expending all of the physical strength, defeat is declared. This system verified our algorithm in this paper using a 3:3 simulation.

The full system was configured as shown in Figure 4. The real-time strategy simulation (RTSS) server controls and manages the entire game and the RTSS client. The simulation client performs simulation as interworking with a red agent. The RTSS server, RTSS client, and a red agent act on each computer.

Figure 4. Structure of the whole system.

The red agent receives the initial data from the RTSS client. Depending on the initial data, the red agent route is determined. The route includes information, including sub-goals and goals. After the simulation begins, a red agent receives state information that is frequently transferred. Initial data include information on scenario, map, team, team members and their ranking, location and state of each agent, number and location of goal, and ID of the present agent. State information includes information on the team, team members, and the location and state of each agent.

4.2. Development Environment

A red agent was developed and tested in Python 3.6, Tensorflow 1.15.0, and Keras 2.0.3. The development environment was divided into learning and experimental environments.

Our method adopted hyperparameters, as shown in Table 2, using A3C. The regularization term parameter to calculate the loss of an actor is 1×10^{-10} . Fifteen threads were used for A3C learning, and fifteen local networks were adopted. Each local network updates the policy of the actor and value of the critic to the global network after every step. The upper layer was updated when the renewal error exceeded 660 m. The interval of action performance in the lower layer was set to 2.0 s. The penalty value was excluded from the calculation of the reward.

Table 2. Hyperparameters used.

Hyperparameter	Value	Hyperparameter	Value
discount factor	0.99	learning rate	2.5×10^{-4}
regularization term parameter	1×10^{-10}	threads	15
moving penalty	0.0001	penalty of caught by enemy	1
goal reward	5	terrain penalty	1
death penalty	5	firing range	1.8 (km)
max step	300	max hp	5
renewal error	660 (m)	action interval	2.0 (s)

4.3. Full Map and Designation Cases

The experiment used a 40 km $\times$ 40 km map, including Baengnyeongdo Island. The red agents started from the top right direction and aimed to arrive at Baengnyeongdo Island. In this process, the red agents received the case data generated on the basis of

55 types of cases selected by experts and attained the goal of passing sub-goals in sequence. Only one case was used in the experiment. The route in the designated case is marked by a solid black line in Figure 5. The light green and dark green dots indicate sub-goals and goals. The green edges on the island are the places that the red agents could arrive. Red agents were divided into groups #1 and #2 to move toward the goal.

Figure 5. Information on the case designated for the experiment.

4.4. Change of Loss Depending on the Number of Learning by the Proposed Approach and Cumulative Number of Successes

The changes in loss depending on the number of learning events and the cumulative number of successes were compared. Figure 6 shows the change in loss depending on the number of learning steps. The loss of an actor stably decreased when more than 130,000 episodes were learned. Low loss was observed around the learning of 160,000 episodes. The loss of a critic decreased stably around the learning of 100,000 episodes. A low loss was found around the learning of 140,000 episodes. Next, the learning of more than 180,000 episodes in an actor- and critic-induced overfitting took place, and accordingly, the loss of both an actor and a critic was increased.

Figure 6. Change of loss depending on the number of learning using the proposed approach: (a) loss of actor, and (b) loss of critic. The x-axis is number of episodes (units: 10,000 episodes) and the y-axis is loss.

Figure 7 illustrates the number of arrivals to sub-goals and the goal depending on the number of learning events using the proposed approach. Each bar graph shows the cumulative number of arrivals, and the polygonal line indicates the winning rate per 20,000 episodes. As an example, the winning rate after learning 20,000 episodes in Figure 7a was the average winning rate of 20,000 total episodes from the first to the 20,000th episode.

As shown in Figure 7a, when an agent reached a sub-goal once, it continued to reach sub-goals. The rate at which goals were reached was maintained at over 80% after 40,000 episodes. The winning rate is not consistent in Figure 7b. The highest winning rate was recorded in 180,000 episodes. Subsequently, the winning rate declined owing to overfitting.

Figure 7. Cumulative number of goal and sub-goals reached depending on the number of learning using the proposed approach: (a) number of sub-goals reached, and (b) number of goals reached. The x-axis is number of episodes (units: 10,000 episodes), the left y-axis is number of sub-goals/goals reached, and the right y-axis is reach rate.

4.5. Change of Loss Depending on the Number of Learning by Traditional Double Layered A3C and Cumulative Number of Successes

The changes in loss and cumulative number of successes using the traditional double layered A3C were analyzed. Traditional double layered A3C adopted a double layered approach and applied A3C to each layer. Figure 8 illustrates the change in loss depending on the number of learning cycles using the double layered A3C. Compared with Figure 6 in Section 4.4, in the case of the lower layer, the trend of the loss is similar; however, in the case of the upper layer, the loss of the upper layer was not minimized with 200,000 episodes.

Figure 8. Change of loss depending on the number of learning using the double layered A3C: (a) loss of actor and (b) loss of critic. The x-axis is number of episodes (units: 10,000 episodes) and the y-axis is loss.

Figure 9 presents the number of sub-goals and goals reached depending on the number of learning using the traditional double layered A3C. The highest achievement rate to sub-goals was 0.9492 at learning of 200,000 episodes when using the traditional double layered A3C, which was similar to the highest reach rate using our method, 0.96375. As shown in Figure 9b, the more learning that was completed, the more attainment of the goals. While the reach rate to a goal was 0.9192 at 180,000 episodes when using the proposed method, the rate of reaching a goal was only 0.7323 at 200,000 episodes when using the traditional double layered A3C. The cumulative number of goals and sub-goals reached by our method was 90,134, and those reached by the traditional double layered A3C numbered 75,831, with a difference of 14,303. In terms of the percentage, the goal reach rate by the traditional double layered A3C was 25% lower than our algorithm, and the cumulative

number of goals reached by the traditional double layered A3C was 18.8% lower than that of the proposed method.

Figure 9. Cumulative number of sub-goals and goals reached by the traditional double layered A3C: (a) number of sub-goals reached and (b) number of goals reached. The x -axis is number of episodes (units: 10,000 episodes), the left y -axis is number of sub-goals/goals reached, and the right y -axis is reach rate.

4.6. Comparison of Learning Time and Calculation Time between the Traditional Double Layered A3C and the Proposed Approach

The learning time and calculation time were compared between the traditional double layered A3C and the advanced double layered A3C. The basis for the learning time was when the goal attainment rate of an agent was over 90% in the recent 20,000 episodes. The time was measured from the time an agent started learning to when the goal attainment rate was over 90% for the first time. Figure 10a presents the learning time in the upper and lower layers in the traditional double layered A3C and in the lower layer in the proposed method. The learning time in the lower layer using the traditional double layered A3C and our method was 317 and 306 min, respectively. However, the learning time in the upper layer using the traditional double layered A3C was 4132 min, which is approximately 13 times longer than that in the lower layer. Since the traditional double layered A3C required learning in both the upper and lower layers, the total learning time by the proposed method was reduced by 7.1%.

Figure 10. Comparison of learning time and calculation time between the traditional double layered A3C and the proposed approach. (a) Learning time comparison. The y -axis is time (min). (b) Calculation time comparison. The y -axis is time (ms).

Figure 10b compares the calculation time in the upper layer using the traditional double layered A3C and our algorithm. Since the traditional double layered A3C applied the weighed values that were learned before, the calculation time was 139 ms on average. Meanwhile, our method decided values in the upper layer after calculating the average of values in the lower layer, and the average calculation time was 626 ms, which was 4.5 times longer than that in the upper layer using the traditional layered A3C.

5. Discussion and Conclusions

This paper proposed an enhanced double layered A3C for a multi-agent system to express a state space in a 2D grid and enable an agent to decide the route to travel. The entire grid space was divided into the upper layer and the lower layer to reduce the spatial state. Moreover, only the lower layer was trained to reduce learning time, and decision making in the upper layer was based on learning in the lower layer. The upper layer determined the spatial state to be considered in the lower layer using the values from the lower-layer cells without any separate training. The lower layer decides the policies after learning the spatial state determined by the A3C.

To verify the performance of our method, the learning and experimental environments were implemented using a virtual simulator, and a 3:3 simulation was performed. In Figure 8b, the upper layer of the traditional double layered A3C was not minimized with 200,000 episodes, and it took 4132 min for the recent 20,000 episodes to reach more than 90% of the goal in Figure 10a. Therefore, the traditional method had a low rate of arrival of sub-goals and goals, as shown in Figure 9, because the upper layer was not properly trained. The proposed method reduced the learning time to less than 1/13 times compared to the traditional method by making decisions using the learned value of the lower layer without training the upper layer, which is only 7.1% of the total learning amount of the traditional method. We were able to improve the performance using the proposed method: the highest rate of reaching goals was 0.9192 at 200,000 episodes, which was 25% higher than that of the traditional double layered A3C (0.7323). In addition, the cumulative number of achievements of the goals increased by approximately 18.86% using A3C with double layers. Accordingly, total learning time in the upper layer and lower layer using the proposed approach was just 7.1% of that using the double layered A3C. The highest reach rate to the goal was 0.9192 at 200,000 episodes, which was 25% higher than that of the traditional double layered A3C, that is, 0.7323. In addition, the cumulative number of goals reached was increased by about 18.86%.

In the proposed hierarchical structure, only the learning process of the lower layer is required, and the upper layer makes decisions using the values of the lower layer without any additional learning process. Therefore, our advanced double layered method reduces the overall training time by 7.1% and reduces the overall state space. The experiments also show that the proposed method can achieve more than 18.86% higher performance than the traditional double layered method. By applying the advanced double layered method, it is possible to apply A3C and other reinforcement learning-based approaches where the problem of state space is invoked.

Further studies will be conducted on the diversity of the environment by identifying a variety of cases. Such future research will consider a diversity of agent strategies by reflecting the traveling speed and actions, including movement, breakthrough, and avoidance for each case. Moreover, we will investigate how to shorten the calculation time required when the upper layer considers the values from the lower layer and applies them.

Author Contributions: Validation and Software, Formal Analysis, D.L.; Conceptualization and Revision, Y.S.; Investigation, D.L., J.K. and Y.S.; Writing, D.L., J.K. and Y.S.; Review, K.C. and Y.S.; Supervision: K.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Future Challenge Program through the Agency for Defense Development funded by the Defense Acquisition Program Administration.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M.G.; Pineau, J. An introduction to deep reinforcement learning. *arXiv* **2018**, arXiv:1811.12560.
2. Demirhan, M.; Premachandra, C. Development of an Automated Camera-Based Drone Landing System. *IEEE Access* **2020**, *8*, 202111–202121. [[CrossRef](#)]

3. Shehzad, M.F.; Bilal, A.; Ahmad, H. Position & attitude control of an aerial robot (quadrotor) with intelligent pid and state feedback lqr controller: A comparative approach. In Proceedings of the 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 8–12 January 2019; IEEE: Piscataway, NJ, USA; pp. 340–346.
4. Sung, Y.; Jin, Y.; Kwak, J.; Lee, S.G.; Cho, K. Advanced camera image cropping approach for CNN-based end-to-end controls on sustainable computing. *Sustainability* **2018**, *10*, 816. [\[CrossRef\]](#)
5. Rida, I.; Al-Maadeed, N.; Al-Maadeed, S.; Bakshi, S. A comprehensive overview of feature representation for biometric recognition. *Multimed. Tools Appl.* **2020**, *79*, 4867–4890. [\[CrossRef\]](#)
6. Kwak, J.; Sung, Y. Autoencoder-based candidate waypoint generation method for autonomous flight of multi-unmanned aerial vehicles. *Adv. Mech. Eng.* **2019**, *11*, 1687814019856772. [\[CrossRef\]](#)
7. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
8. Kulkarni, T.D.; Narasimhan, K.; Saeedi, A.; Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3675–3683.
9. Li, S.; Wang, R.; Tang, M.; Zhang, C. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *arXiv* **2019**, arXiv:1910.04450.
10. Rida, I. Feature extraction for temporal signal recognition: An overview. *arXiv* **2018**, arXiv:1812.01780.
11. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning (PMLR), New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
12. Tewari, U.P.; Bidawatka, V.; Raveendran, V.; Sudhakaran, V.; Shreesail, S.K.; Kulkarni, J.P. Intelligent coordination among multiple traffic intersections using multi-agent reinforcement learning. *arXiv* **2019**, arXiv:1912.03851.
13. Liu, Z.; Liu, H.; Lu, Z.; Zeng, Q. A Dynamic Fusion Pathfinding Algorithm Using Delaunay Triangulation and Improved A-Star for Mobile Robots. *IEEE Access* **2021**, *9*, 20602–20621. [\[CrossRef\]](#)
14. Panov, A.I.; Yakovlev, K.S.; Suvorov, R. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia Comput. Sci.* **2018**, *123*, 347–353. [\[CrossRef\]](#)
15. Krothapalli, U.; Wagner, T.; Kumar, M. *Mobile Robot Navigation Using Variable Grid Size Based Reinforcement Learning*; Infotech@Aerospace: Louis, MO, USA, 2011; p. 1533.
16. Babu, V.M.; Krishna, U.V.; Shahensha, S.K. An autonomous path finding robot using Q-learning. In Proceedings of the 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 7–8 January 2016; IEEE: Piscataway, NJ, USA; pp. 1–6.
17. Quan, H.; Li, Y.; Zhang, Y. A novel mobile robot navigation method based on deep reinforcement learning. *Int. J. Adv. Robot. Syst.* **2020**, *17*, 1729881420921672. [\[CrossRef\]](#)
18. Zhu, D.; Li, T.; Ho, D.; Wang, C.; Meng, M.Q.H. Deep reinforcement learning supervised autonomous exploration in office environments. In Proceedings of the 2018 IEEE international conference on robotics and automation (ICRA), Brisbane, Australia, 21–25 May 2018; IEEE: Piscataway, NJ, USA; pp. 7548–7555.
19. Zhou, G.; Azizsoltani, H.; Ausin, M.S.; Barnes, T.; Chi, M. Hierarchical reinforcement learning for pedagogical policy induction. In Proceedings of the International Conference on Artificial Intelligence in Education, Chicago, IL, USA, 25–29 June 2019; Springer: Cham, Switzerland; pp. 544–556.
20. Scheiderer, C.; Mosbach, M.; Posada-Moreno, A.F.; Meisen, T. Transfer of Hierarchical Reinforcement Learning Structures for Robotic Manipulation Tasks. In Proceedings of the 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 16–18 December 2020; IEEE: Piscataway, NJ, USA; pp. 504–509.
21. Gustafson, N.J.; Daw, N.D. Grid cells, place cells, and geodesic generalization for spatial reinforcement learning. *PLoS Comput. Biol.* **2011**, *7*, e1002235. [\[CrossRef\]](#)
22. Wang, R.; Yu, R.; An, B.; Rabinovich, Z. I2HRL: Interactive Influence-based Hierarchical Reinforcement Learning. In Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI, Yokohama, Japan, 11–17 July 2020; pp. 3131–3138.