AI

MDPI

Article

Hierarchical DDPG for Manipulator Motion Planning in Dynamic Environments

Dugan Um [1], Prasad Nethala [2,*] and Hocheol Shin [3]

[1] School of Engineering & Computing Sciences, Texas A&M University-Corpus Christi, TX 78412, USA; dugan.um@tamucc.edu
[2] Geospatial Systems Engineering, Texas A&M University, Corpus Christi, TX 78412, USA
[3] Nuclear Robot and Diagnosis Team, Korea Atomic Energy Research Institute, Daejeon 34057, Korea; smarthc@kaeri.re.kr
* Correspondence: snethala@islander.tamucc.edu

Abstract: In this paper, a hierarchical reinforcement learning (HRL) architecture, namely a "Hierarchical Deep Deterministic Policy Gradient (HDDPG)" has been proposed and studied. A HDDPG utilizes manager and worker formation similar to other HRL structures. However, unlike others, the HDDPG enables sharing an identical environment and state among workers and managers, while a unique reward system is required for each Deep Deterministic Policy Gradient (DDPG) agent. Therefore, the HDDPG allows easy structural expansion with probabilistic action selection of a worker by the manager. Due to its innate structural advantage, the HDDPG has a merit in building a general AI to deal with a complex time-horizon tasks with various conflicting sub-goals. The experimental results demonstrated its usefulness with a manipulator motion planning problem in a dynamic environment, where path planning and collision avoidance conflict each other. The proposed HDDPG is compared with an HAM and a single DDPG for performance evaluation. The result shows that the HDDPG demonstrated more than 40% of reward gain and more than two times the reward improvement rate. Another important feature of the proposed HDDPG is the biased manager training capability. By adding a preference factor to each worker, the manager can be trained to prefer a certain worker to achieve better success rate for a specific objective if needed.

Keywords: RL reinforcement learning; HRL hierarchical reinforcement learning; DDPG Deep Deterministic Policy Gradient; HDDPG Hierarchical Deep Deterministic Policy Gradient; HAM Hierarchical Abstract Machines

check for updates

Citation: Um, D.; Nethala, P.; Shin, H. Hierarchical DDPG for Manipulator Motion Planning in Dynamic Environments. *AI* **2022**, *3*, 645–658. https://doi.org/10.3390/ai3030037

Academic Editors: Phivos Mylonas, Katia Lida Kermanidis and Manolis Maragoudakis

Received: 26 May 2022
Accepted: 26 July 2022
Published: 3 August 2022

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

Copyright: © 2022 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/).

1. Introduction

The human brain is still traditionally seen as a black box in many analogies for research. The challenge for theorists is to incorporate novel theories into hierarchical decision models to improve our understanding of human decision-making [1]. However, while many fundamental concerns as to how the brain works remain unresolved, there are several well-founded hypotheses that support hierarchical models. Humans use hierarchical structure to perceive visual motion [2], which is the primary motivation behind choosing a hierarchical architecture to address temporal and spatial abstraction. By addressing subgoals in a hierarchical manner, the size of state space is reduced, allowing for transfer learning and the possibility of policy reuse.

In complex task planning, HRL (Hierarchical Reinforcement Learning) is a feasible architecture to be implemented for achieving long-term goals by breaking them into subgoals. A complex task, in our study, is defined as a task with subdividable multiple tasks or with tasks demanding sequential execution. An RL agent must learn how to recognize these subgoals and objectives on its own in a holistic HRL paradigm as well as how to build a policy hierarchy that makes use of them to accomplish the overall objective. Hierarchical reinforcement learning methods are supposed to offer a decision-making capability in

complex environments, especially at solving depth of planning horizon problems. Andrei Nica et al. [3] proposed a model-free algorithm to learn affordances that can be used to further learn subgoal options. The concept of affordances in Gibson et al. [4] suggests that only certain actions are feasible in certain states.

End-to-end HRL methods by Shubham et al. [5] accomplishes hierarchical management capability by using a higher-level policy in the hierarchy to search the database directly in a continuous subgoal space learning. However, when the subgoal space is large, such a policy may be difficult to implement. They offer an integrated approach in identifying important subgoals, an integrated subgoal discovery technique with an end-to-end HRL method, which is a heuristic approach for narrowing down the search space for higher-level policies. During this process, it concentrates on the subgoals with the greatest impact occurrence probability on distinct state-transition trajectories, which will finally lead to the desired outcome. The end-to-end HRL, also known as learning with Integrated Discovery of Salient Subgoals (LIDOSS), outperformed Hierarchical Actor Critic (HAC) [6] on a series of continuous control tasks in a physics simulation domain. The Hierarchical Abstract Machine (HAM) by Parr et al. [7], a non-deterministic finite state machine with the ability to activate lower-level machines through their transitions, is another interesting Hierarchical RL technique. Policies in HAM can be viewed as programs that select the best state to achieve the overall goal.

Another classical HRL approach called the MAXQ method by Dietterich [8] is a hierarchical learning algorithm in which the hierarchy of a task is obtained by decomposing the Q value of state–action pair into the sum of two components of V (a, s) and C (p, s, a). Where, V (a, s) is the total expected reward received when executing the action 'a' in state 's' (classic Q value function) and C is the total reward expected from the performance of the parent-task, noted by 'p', after taking the action a. In fact, the action 'a' may not only contain a primitive action, but also a sequence of actions.

In essence, one can understand the MAXQ framework as decomposing the value function of an MDP into combinations of value functions of smaller constituent MDPs. Each MDP is a finite set of sub-tasks, where each sub-task is formalized as a termination predicate, set of actions, and a pseudo reward. In the Feudal RL method by Dayan et al. [9], the managers only need to know the state of the system at the granularity of their own task choices, which is a notable result of information and reward concealment. They also have no idea what decisions their personnel made to fulfill their orders because the system is not set up to learn. They have two advantages; the first is information hiding (at different resolutions, the managerial hierarchy observes the surroundings) and the second is reward hiding (goals are used to communicate between managers and "employees", and employees are rewarded if they are met).

The most well-known HRL approach is probably the options framework by Bacon et al. [10]. From the options-framework-based HRL method, a Markov option is defined as a triple option of $<Io, \pi o, \beta o>$, where Io: the initiation set, πo: $S \times A \rightarrow [0,1]$: the option's policy, βo: $S \rightarrow [0,1]$: the termination condition. Unlike feudal learning, an algorithm adopting the options framework demonstrates its ability of convergence to an optimal policy if the action space contains both primitive actions and options. Otherwise, it will converge, but to a policy that is hierarchically optimal. Moreover, options itself can be used to define option hierarchies. However, due to its innate nature, options increase the complexity of the MDP. They also do not explicitly address the problem of task segmentation.

Many new hierarchical approaches have been developed recently, many of which are inspired by these pioneering hierarchical structures (feudal, options, HAM, and MAXQ of HRL). Wang, Zi et al. [11] used hierarchical DRL (HDRL) to study cell migrations using images of cells and tissues with a ubiquitous nuclear label, which provides detailed information regarding dynamic cell movements. Kulkarni, Tejas et al. [12] presents a h-DQN (Hierarchical- Deep Q Network) framework that consists of hierarchically arranged deep reinforcement learning modules that operate at various time scales, and the model makes decisions at two levels of hierarchy: a top-level module (meta-controller) takes in the

current state and selects a new goal, and a lower-level module (controller) chooses actions until the goal is reached or the episode ends.

Siyuan Li et al. [13] propose an HRL framework that uses the advantage function of the high-level policy to create auxiliary rewards for low-level skill training while maintaining reward design generality. This auxiliary reward allows for efficient, concurrent acquisition of high-level policy and low-level skills without the need for task-specific knowledge. Unlike existing hierarchical multitask RL, Sungryull Sohn et al. [14] proposed a neural subtask graph solver (NSGS) that encodes the subtask graph using a recursive neural network embedding, where the agent must generalize to a previously unseen environment characterized by a subtask graph that describes a set of subtasks and their dependencies.

Given the review of the short history of HRL, we envision the DDPG as a building component of a novel HRL architecture. The capability of a single DDPG is proven well for solving goal-oriented tasks, but it is insufficient to address complex multi-tasking or sequential tasking problems. In addition, since DDPG is prone to the curse of dimensionality as state grows and is born to be a single-goal oriented architecture, it is not suitable for a large-scale AI.

In this paper, we examine an HRL architecture using the DDPG as a building block for a large-scale AI. We envision a well-trained DDPG for a specific task as a reusable transfer component to constitute a larger group of DDPG to solve more complex tasks. Later, a well-trained group of DDPGs can also be adopted for another larger group, and so on. In the following section, we explain a general approach of constructing an HRL architecture via multiple DDPGs, followed by some experimental results in Section 3.

2. Methods

2.1. Reinforcement Learning

Reinforcement learning is known to be an effective solution provider for specific domain problems such as game or machine control. However, agents face a challenging task if they are to successfully apply reinforcement learning in scenarios that resemble the complexity of real-world problems: they must create accurate representations of their environment from high-dimensional sensory inputs and then generalize their prior knowledge to novel situations. Interestingly, reinforcement learning, and hierarchical sensory processing systems seem to work in harmony to help humans and other animals solve this problem. This is supported by a wealth of neural data that shows notable similarities between the phasic signals emitted by dopaminergic neurons and temporal difference reinforcement learning algorithms [15]. Therefore, we use the Reinforcement Learning as a building block of a large-scale AI. Reinforcement learning (RL) algorithm is born on the framework of Markov Decision Process (MDP) [16], whereby, for a given state, an admissible action takes place to maximize a reward. In more detail, an agent observes a system's state 's', contained in a finite set of state space φ, and executes an action 'a' selected from a finite, not-empty set of admissible actions A at each state in a sequence of stages (Bertsekas et al. [17]). Once the agent receives an immediate reward for a pair of state 's' and action 'a', the current state s will change to the next state of s' with probability $P(s' \mid s, a)$. The name, "one-step model" of action a is from the fact that the expected immediate rewards, $R(s, a)$ accompanies with the state transition probabilities, $P(s' \mid s, a)$, s, and $s' \epsilon\ \varphi$. The action a is chosen from a stochastic policy π: $\varphi \times U_{s\ \epsilon\ \varphi}, A_s \rightarrow [-1,1]$, with $\pi(s, a) = 0$ for $a! \epsilon A_s$. For a given policy π, s ($\epsilon\ \varphi$, $V^{\pi}(s)$) represents the expected infinite-horizon discounted return from state s, or simply the value of s given the condition that the agent uses policy π such that,

$$V^{\pi}(s) = E\left\{ r_{t+1} + \Upsilon \cdot r_{t+2} + \Upsilon^2 \cdot r_{t+3} + \ldots \middle| s_t = s,\ \pi \right\} \quad (1)$$

where Reward $R(s, a) = \sum (r_{t+1} + \Upsilon \cdot r_{t+2} + \Upsilon^2 \cdot r_{t+3} + \ldots)$ and Υ, $0 \leq \Upsilon < 1$, is a discount factor. The value of each state is independent of the history of state change; thus, it is MDP. Since the objective of MDP is to maximize the value of each state, it is an optimization problem of policy π, or action selection, given a state, s. This corresponds to the critic size

of actor–critic-based reinforcement learning. In RL, the importance is on estimating action-value function, that represents the value of each admissible action for each state. Given a policy π, the value of (s, a) pair, or $Q^{\pi}(s, a)$ is the expected infinite-horizon discounted return for executing an action a in state s such that,

$$Q^{\pi}(s,a) = E\left\{r_{t+1} + \Upsilon \cdot r_{t+2} + \Upsilon^2 \cdot r_{t+3} + \ldots \middle| s_t = s,\ \ a_t = a,\ \pi\right\} \tag{2}$$

The optimal action-value function, Q^*, assigns to each admissible state- action pair the expected infinite-horizon discounted return. By using dynamic programing in the formalism of Bellman function [18], $V_{\pi}(s)$ and $Q_{\pi}(s, a)$ are known to converge to the maximum values of V^* and Q^* as shown in equations below.

$$V^*(s) = \text{argmax}\left[R(s,a) + \Upsilon \cdot \sum_{s'} P(s'|s,a) \cdot V^{\pi}(s')\right] \tag{3}$$

$$Q^*(s,a) = R(s,a) + \Upsilon \cdot \sum_{s'} P(s'|s,a) \cdot \text{argmax}\, Q^*(s',a') \tag{4}$$

For practical use of above equations in RL formalism, two functions can be formulated in iterative dynamic programming equations such as,

$$V_{k+1}(s) = \text{argmax}\left[R(s,a) + \Upsilon \cdot \sum_{s'} P(s'|s,a) \cdot V_k(s')\right] \tag{5}$$

$$Q_{k+1}(s,a) = R(s,a) + \Upsilon \cdot \sum_{s'} P(s'|s,a) \cdot \text{argmax}\, Q_k(s',a') \tag{6}$$

RL (Reinforcement Learning) is one of many Dynamic Programming (DP) algorithms that is unique in its use of Monte Carlo, stochastic approximation. First, it avoids exhaustive sweeps by restricting computation to states or in the neighborhood of either real or simulated. Second, it simplifies the basic DP backup by estimating a backup's effect through sampling from the appropriate distribution. Finally, it represents value functions and/or policies more compactly than lookup-table representations by using non-linear function approximation such as linear combination of basis function, neural networks, etc.

In the proposed HDDPG architecture, V^* and Q^* will be kept intact for each worker after training, while V^* and Q^* of a manager will be created during the manager training. Therefore, action of a manager will be dependent on that of each worker. That is,

$$\begin{gathered} a|\pi_m = a\ \in \{a_1,\ a_2,\ a_3, ..a_n\}, \\ Q^m_{k+1}(s,a) = R(s,a) + \Upsilon \cdot \sum_{s'} P(s'|s,a) \cdot \text{argmax}\, Q^m_k(s',a') \end{gathered} \tag{7}$$

In Equation (7), Q^m_k is the action-value function of a manager, π_m is the trained policy of the manager, and $a_1,\ a_2,\ a_3, \ldots\ a_n$ represent the action of each worker.

2.2. Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradients (DDPG) proposed by Lillicrap et al. [19], also known as the policy-gradient actor-critic algorithm, uses some of the deep learning techniques from DQN and is both model-free and off-policy. Policy-gradient algorithms estimate a deterministic target policy, which is considerably simpler to learn, but uses a stochastic behavior policy with enough amount of exploration experience. Using the Actor-Critic learning technique, the policy function is represented separately from the value function. The Actor is the policy function neural network, while the Critic is the value function neural network. Given the environment's present state s, the actor emits an action a, and given the reward received, the critic generates a *TD* (Temporal-Difference) error signal. The output of the Actor is also required if the Critic is to estimate the action-value function $Q(s, a)$. DDPG learning is influenced by the Critic's output for both Actors and Critics. Neural networks can be utilized in Deep Reinforcement Learning to represent the Actor and Critic structures. The updating rule for the Actor network's weights is provided by the deterministic policy-gradient theorem. The gradients obtained from the *TD* error

signal are used to update the Critic's network. By breaking out the temporal correlations among several training episodes, the replay buffer stores the agent's experiences during training followed by random sampling of events for learning. If the Actor and Critic neural networks' outputs are used to directly update the Actor and Critic neural network weights, DDPG learning algorithm will diverge. In order to regularize the learning algorithm and improve stability, a set of target networks are built to generate the targets for *TD* error computation. Finally, the gradients acquired from the loss function can be used to update the weights of the Critic network, and the Actor network can also be updated using the gradient from the deterministic policy, the workflow of the DDPG agent can be observed in manager agent in Figure in [3].

2.3. Hierarchical Reinforcement Learning

Due to its innate training process, RL is not efficient for solving general problems with complexity or conflicting goals because sampling and scaling as the state and action spaces are very large. To overcome such complexity, hierarchical RL is born to offer generalization capability via transfer learning. As mentioned earlier, the most common hierarchical RL structures such as options or macro framework approaches by Sutton et al. [20] incorporate Semi-MDP, where the decision process not only depends on the sequential nature of the decision process, but the amount of time τ that passes between decision stages (also discussed by Puterman et al. [15] and Barto et al. [21]). The original state transition probability will be changed to the joint probability of $P(s', \tau \mid s, a)$. The reward, $R(s, a)$, then, gives the amount of discounted reward expected to accumulate over the waiting time.

Feudal RL provides a control hierarchy, in which a level of managers can control sub-managers in pyramidal configuration, as mentioned in Dayan et al. [9]. The training starts with each manager assigning sub goals to each sub-managers to perform actions to achieve only the assigned goal. Generally, this occurs by dividing the environment into equal amounts of spaces for each sub-manager. As a result, the higher levels work on a broader granularity with much smaller state space. Although it advanced the idea of HRL, it is known to be highly inefficient in some cases. FeUdal Networks (FUN) HRL proposed by Vezhnevets et al. [22] responded with modular Neural Network, in which each manager receives the state and reward from the environment and sends embedded states and sub-goal to workers. Another type of HRL is option-critic architecture by Sutton et al. [23], whereby learning occurs in both internal policy and termination conditions rather than using a random variable for termination conditions. Therefore, options do not have to be defined beforehand. However, the option-critic HRL still falls short of training a complex time horizon task with conflicting goals. In this paper, we examine a novel HRL architecture using the DDPG as a building block for a large-scale AI by which complex time–horizon problems can be tackled. We explain the basic principles of HDDPG in the next section.

2.4. Hierarchical DDPG (HDDPG)

Hierarchical DDPG (HDDPG) is inspired by FUN HRL in that it is driven by manage-worker relationships (Figure 1). HDDPG is similar to FUN, since it receives states and rewards from the environment, but it is different in handling the workers. Direct access to the state of the environment enables the manager to understand the changing circumstances and select works to achieve a dynamic task. This is especially efficient when tasks are conflicting to each other in their goals. For instance, path planning, and collision avoidance have two mutually conflicting goals since a planner intends to direct an agent to the goal while a collision avoidance algorithm may prevent it from reaching the goal. Hierarchical Abstract Machines (HAM) by Parr et al. [7] may be efficient to solve mutually conflicting goals via multiple agents or policies. This is true when the problem domain is simple and well defined. However, designing a state transition condition may be costly for multiple tasks since manual design of state transition may not be optimal for the best performance overall in complex environments. Therefore, we propose a HDDPG, whereby a manager-

worker framework is employed with probabilistically trained action selection to trigger each worker.

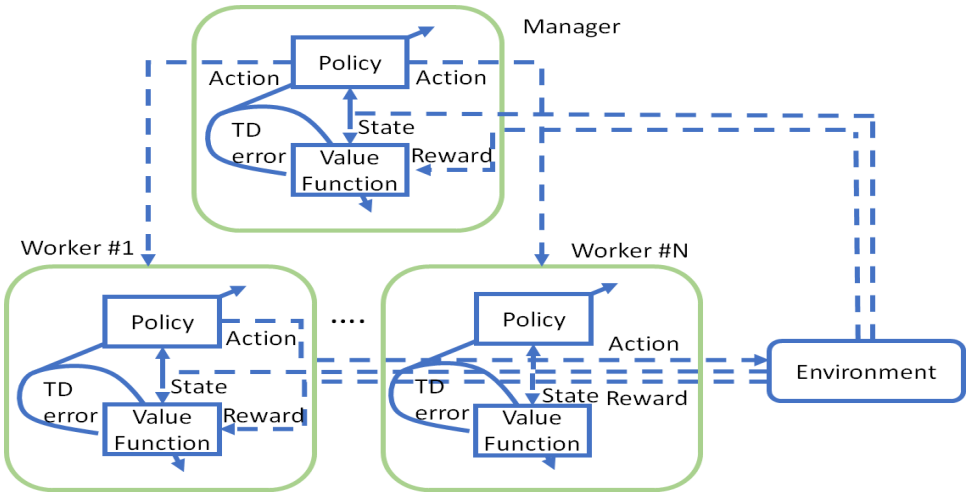

Figure 1. Hierarchical DDPG architecture.

For instance, a robotic path-planning task can be framed in a simple HDDPG structure as shown in Figure 1. Action selection takes place, and the set of state, next state, reward, and action are separately learned by the top-level DDPG. In this framework, each worker is trained beforehand so that each worker is tailored to achieve a specifically assigned goal. The goal preferred DDPG agent is trained for path planning to reach the goal, while the obstacle avoidance preferred DDPG agent is trained to avoid dynamic obstacles. The idea behind the HDDPG is that this basic structure can be expanded in a pyramidal configuration as much as it is required. Therefore, a complex task, once carefully divided into smaller tasks with specifically assigned goals, can be solved. The design specification of the hierarchical DDPG includes the following

1. Each DDPG will share the same environment.
2. A DDPG group is composed of a DDPG at a higher layer and multiple DDPGs at a lower layer.
3. Each DDPG or a DDPG group can become a sub layer of another DDPG.
4. Each DDPG will receive selected states from total states and reward from a tailored reward system for each DDPG.

First, unlike the FUN architecture, by sharing the same environment, there is no need for creating another environment or simpler version for each layer. Second, A DDPG assembly can be a group of workers and manage to achieve a complex task. Third, the HDDPG is flexible and expandable since each DDPG or a DDPG assembly can be added in the existing structure. In the same token, a DDPG can be divided into multiple DDPG for faster convergence to a goal. Finally, while sharing the environment, each DDPG in the HDDPG architecture needs to select specific states from the total state pool and a tailored reward system for optimal performance. The design specification of HDDPG enables transfer learning for multiple task execution with minimal learning period in a complex environment.

The Hierarchical DDPG algorithm (Algorithm 1) provides a control architecture coined for expansion towards a generalized AI, utilizing its flexibility and expandability. It is a human mimicry of one's growth process in that a baby learns how to move the muscles of arm, of foot, and of leg so that he or she can walk or run later. Basic muscle movement trained earlier will help develop further complex moves later in a harmonic fashion. Therefore, the HDDPG architecture enables accumulation of smaller policies to build a larger policy by transfer learning and object-oriented training. The object-oriented training enables reuse of already trained components via transfer learning.

Algorithm 1 HDDPG.

Randomly initialize critic and actor networks for training Goal and Obstacle preferred tasks.
Obtain the DDPG Worker_1 trained model for goal touch preferred task.
Obtain the DDPG Worker_2 trained model for Obstacle avoidance preferred task.
:
Obtain the DDPG Worker_n trained model for Obstacle avoidance preferred task.
Randomly initialize critic and actor networks of Manager_HDDPG agent model for managerial level tasks to deploy best worker action 'a' for the current state 's' to achieve maximum reward.
Initialize target
Initialize replay buffer RB
for maximum Episode do
    Initialize a random process N for action exploration.
    Receive initial observation state s_1.
    for maximum step do
        Retrieve action probability of manager from the manage policy:
        From given action probability range:
            Range 1: execute worker_1 action 'a_1'
            Range 2: execute worker_2 action 'a_2'
                :
            Range n: execute worker_n action 'a_n'
            Store reward r_t and new state s_{t+1} transitions in RB.
        If buffer is full, train the model
            N transitions (s_i, a_i, r_i, s_{i+1}) are randomly sampled from RB.
            Update all the networks and compute the loss function.
    end
end

Therefore, a worker must be trained earlier than a manager associated with it. In addition, a larger HDDPG can be created via a systematic expansion in the following procedure.

(1) Primitive DDPGs interact with an environment via system dynamics in the environment.
(2) Multiple HDDPGs can be grouped into a new HDDPG for synergy and complex goal creation.
(3) A manager's action is a probabilistic selection of a worker.
(4) All DDPGs in the totality of HDDPG interact with the same environment.
(5) The reward of all managers is the numeric sum of all the normalized rewards from its worker DDPGs.
(6) A new state can be added for superior performance at any level if needed.

3. Experiment

The purpose of a series of experiments is to test and validate the proposed HDDPG architecture. The selected task is a robotic path planning in a dynamic environment with moving goals and obstacles. That is, a robot needs to reach a random walking goal, while, at the same time, avoiding collision with multiple dynamic obstacles. To that end, a simulation environment is created with a snake robot (six vertical and six horizontal joints) with a dynamic goal and three moving obstacles (Figure 2). In order to facilitate training, we constrained the snake robot motion to planar motion. Therefore, six vertical joints of the snake robot are activated for training. The goal is to train six vertical joints of the snake robot to reach a dynamic goal, and, at the same time, to avoid collision with obstacles that are walking randomly in the workspace. This scenario is common for a collaborative robot in a smart shop floor, or in a collaborative working environment where the robot is directed to perform tasks while avoiding dynamic objects. We assume that the robot is equipped with a sensitive skin to measure distances to all obstacles in its workspace, while the end effector can evaluate the distance vector to the goal. The detailed architecture of

the HDDPG is illustrated in Figure 3 while the basic structure of each DDPG is kept intact from Guo et al. [24].

Figure 2. Training environment for goal-touch and obstacle-avoidance.

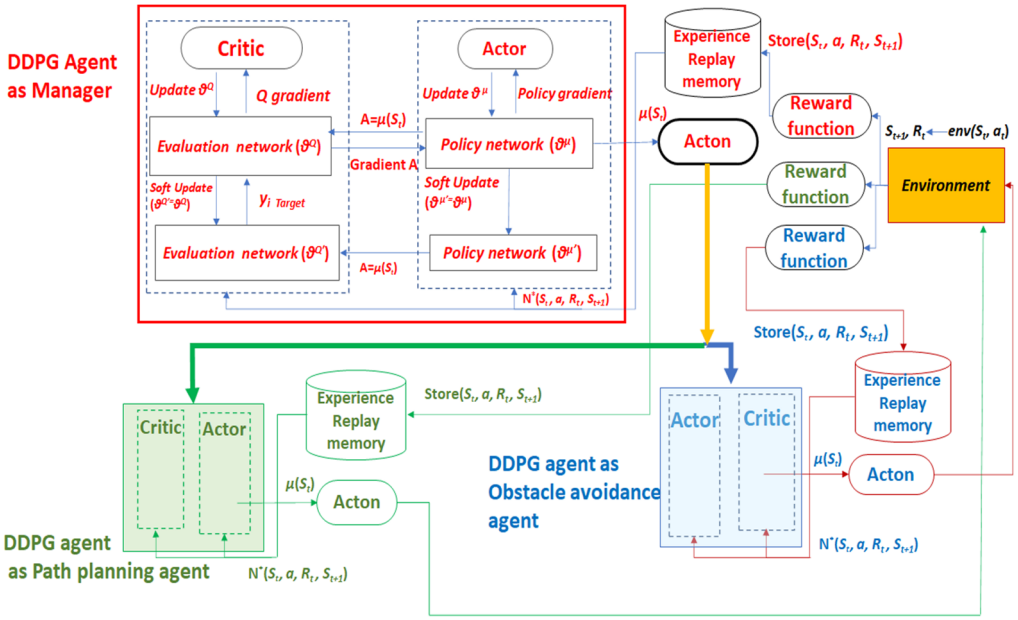

Figure 3. HDDPG architecture with manager–worker strategy.

The basic building block is the actor-critic DDPG, where the action selection and state evaluation take place. Soft update copies the weights learned from the training from current policies to the target policies to avoid bias by sequential transition of states. Each DDPG, whether a manager or a worker, has their own replay memory where the set of state transition and action with the reward will be saved and used for training. In HDDPG, unlike FUN, there is no need to create a tailored environment for each level, but the single environment will serve for all DDPGs. In addition, the action from a manager will not interact directly with the environment, but it selects a worker whose action will interact with the environment. Therefore, the action of a manager is a probability of action selection. For instance, the probability of selecting a DDPG with two workers will be 0.5 for each action and the probability of selecting a DDPG with four workers, for instance, will be 0.25.

3.1. HDDPG Architecture

To build an HDDPG architecture as shown in Figure 3, two primitive DDPGs are developed: a goal touching DDPG and a collision avoidance DDPG. First, each primitive DDPG (worker) is trained independently. State variables and reward systems for each DDPG are described in Tables 1 and 2, respectively. Notice that the total state variables

are 20 tuples for all DDPG, therefore each individual DDPG, either a manager or a worker in the totality of the HDDPG architecture, shares the environment and state variables. However, the reward system for a manager will be an algebraic summation of that of all worker DDPGs associated with it.

Table 1. State variables.

Vector variables	
1.	Normalized 5-joint location vectors (axis 2–axis 6): 5 variables
2.	Distance vectors from the end-effector to the goal: 1 variable
3.	Distance vectors from each joint to the closest nearby obstacle: 5 variables
Scalar variables	
1.	Distance from each joint to the closest nearby obstacle (axis 2–axis6): 4 variables
2.	Boolean variable indicating if goal is touched by the end-effector
3.	Boolean variable indicating if the closest nearby obstacle is touch by each joint: 5 variables
Total state variables: 20 tuples	

Table 2. Reward system.

Goal touch DDPG
Reward = −(distance between the end-effector and the goal)
Reward = Reward + 1 (*if* the end-effector touches the goal)
Collision avoidance DDPG
Reward = +(sum of distances between each joint and the closest nearby obstacle × r_{bf})
Reward = Reward − 1 (*if* a joint touch the closest nearby obstacle)
Higher level (manage) DDPG
Reward = Total reward of goal touch DDPG + total reward of collision avoidance DDPG

Figure 3 describes the architecture of the HDDPG. The manager DDPG agent shown with detailed workflow diagram with both Actor and Critic network structures. The DDPG algorithm uses the knowledge about the initial state as its input and calculates one or more action strategies as its output. The final output action is, then, obtained by adding the random noise to the action strategy, $\mu(S_t)$; this is a typical end-to-end learning mode. The agent emits an action when the task is initiated in accordance with the current state S_t in order to determine whether the output action is valid or not. This yields a feedback reward R_t of the environment. Rewards or penalty are given depending on whether the action is beneficial to the agent in achieving the goal.

In addition, the experience buffer pool is used to hold the current state information, the action, the reward, and the state information of the next time sequence as a training set (S_t, a, R_t, S_{t+1}). In order to further improve the stability and accuracy of the algorithm, the neural network simultaneously trains experience and continuously modifies action strategy by randomly selecting sample data from the experience buffer pool $N^*(S_t, a, R_t, S_{t+1})$, using the gradient descent approach to update and iterate network parameters.

θ^{μ} and θ^{Q} are the parameters of the Actor and Critic networks, while $\theta^{\mu'}$ and $\theta^{Q'}$ are the parameters of the target Actor and the target Critic networks, respectively. In the same manner, at the worker agents' level, the Goal Preferred DDPG agent and the Obstacle Avoidance preferred DDPG agent are shown in black box view of the DDPG explained above as the Manager DDPG agent. The actions from both the workers are effective on the environment. The updated states are observed by the manager, which obtains the resultant reward from the environment and probabilistically chooses the right worker for the current state of the environment with the action, a of the manager. State variables and reward systems are described in Tables 1 and 2, respectively.

3.2. Reward System Design

In addition, reward system design is a crucial task for any RL problem. Singh et al. [25] mentioned the criteria for an appropriate reward system and studied how to derive it while keeping its generality. The following two training criteria will be applicable for a HDDPG architecture.

Training criteria: reward maximum of a worker DDPG while training its manager DDPG cannot exceed that of its own maximum reward achieved during the individual worker training. Therefore, if $R^{bT}_{max}(W_n)$ is the maximum reward of a worker, n, before training, and $R^{aT}_{max}(W_n)$ is the maximum reward of a worker, n, after training, then,

$$R^{aT}_{max}(W_n) < R^{bT}_{max}(W_n) \tag{8}$$

The necessary condition of the above criteria is the reward balancing by normalization. The reward range of each worker must be normalized between zero and one; otherwise, the manager is not trainable due to reward bias. The most challenging part of HDDPG design is building a model by combining different DDPGs or HDDPGs as a group of a new HDDPG. In addition, the reward system designed for each DDPG must be carefully considered for the best performance in training a real-world agent. While the reward of each worker is from the environment as detailed in Table 2, the reward of the manager DDPG is an algebraic sum of all worker's rewards as below.

$$R_{\text{manager}} = \sum_n R(W_n) \tag{9}$$

However, the reward system for the worker DDPG that are interacting with the environment directly must be carefully designed to achieve a specific objective.

3.3. Results

To estimate the performance of the novel HDDPG architecture, HAM is first used for the same motion planning task. While simple in implementation, HAM needs a state machine for a state transition. In our experiment, a simple state transition condition is designed with the sum of distances from the robot to all obstacles as a transition condition. With a threshold value on the sum of distances, the manager triggers a specific worker. As shown in Figure 4, no training occurs in the manager via HAM.

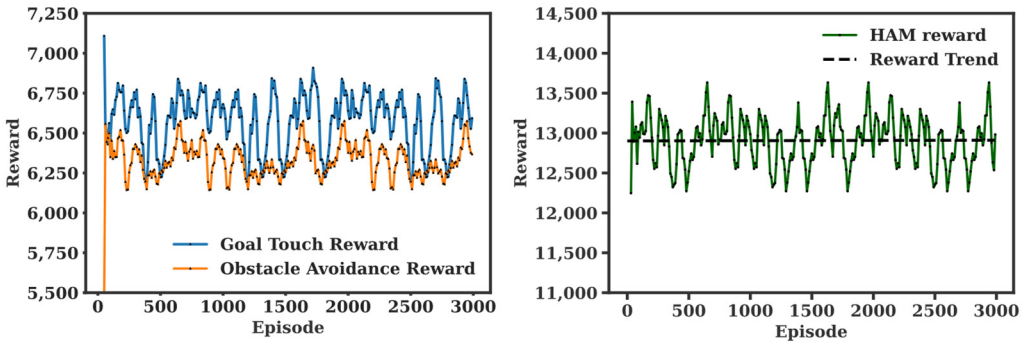

Figure 4. Path planning and obstacle avoidance by HAM.

Second, a single DDPG is trained for comparison with the same reward system described in Table 2. That is, a single DDPG is trained for two conflicting goals of path planning and collision avoidance. In Figure 5, the single DDPG demonstrates the reward gradually increasing from 14,250 to 14,500 on average during the first 3000 episodes. Now, for HDDPG training, two individual DDGPs are first configured and trained independently. Figure 6 shows the training results of the goal touch DDPG and collision avoidance DDPG, respectively. The training of each DDPG is terminated in 250 episodes since no significant improvement is gained afterward. The manager DDPG is trained in the same environment

with the same state variables, but with the reward system outlined in Table 2. The reward of each worker was normalized to ensure the trainability of the manager. As shown in Figure 7, the manager demonstrated its performance gain from the average total reward converging from 19,500 to 20,200. While two DDPGs are competing each other for the first 1200 episodes, both DDPGs are stabilized increasing their rewards steadily thereafter. The main reason is that the workers are competing to maximize their rewards, but the manager selectively activate each worker depending on the environmental change.

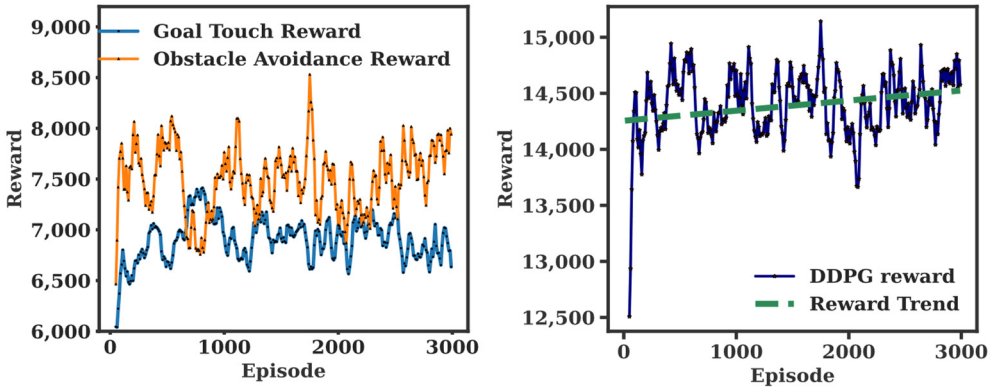

Figure 5. Single DDPG training results with 2 conflicting goals (path planning and collision avoidance).

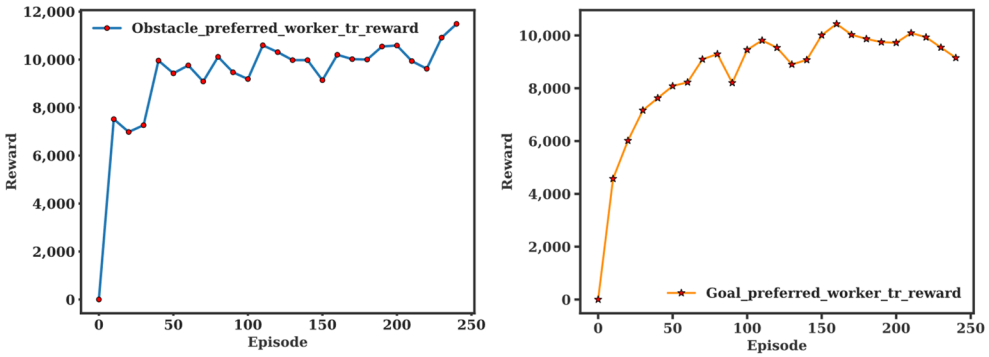

Figure 6. Individual worker training: rewards of obstacle avoidance worker (Obstacle_preferred_worker_tr_reward) and goal touch worker (Goal_preferred_worker_tr_reward).

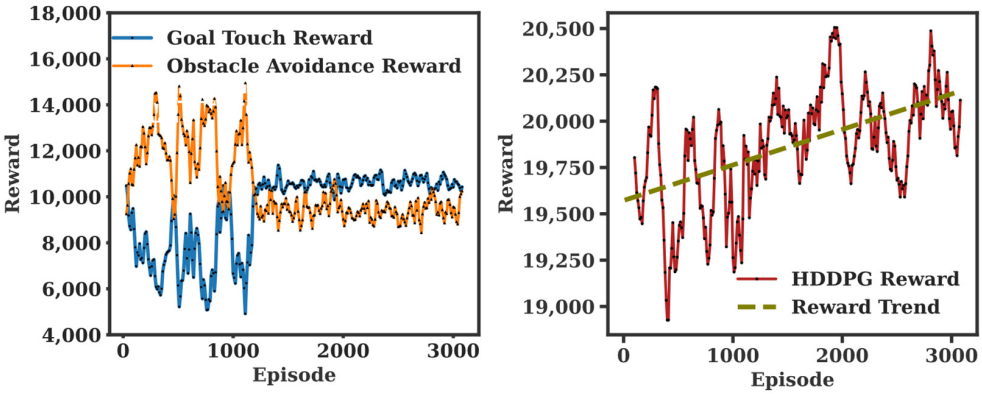

Figure 7. HDDPG training results.

In Figure 8, all three training results are illustrated for performance comparison. As mentioned earlier, no training is observed for HAM, but rewards are improved for both DDPG and HDDPG. While the average reward of the single DDPG for the first 3000 episode

is 14,390, the average reward of the HDDPG was 38% more (19,868) than that of the single DDPG with twice as much as the reward improvement rate.

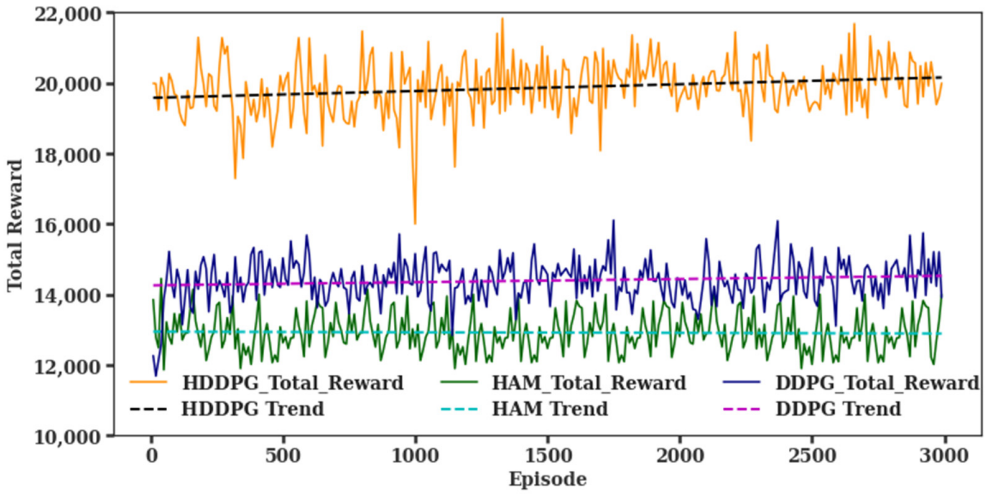

Figure 8. HDDPG demonstrated more than 35% of reward gain and more than 2 times of reward improvement rate (the slop of the linear regression).

The result demonstrates that the proposed HDDPG-based hierarchical reinforcement learning architecture outperforms a general reinforcement learning algorithm, or other hierarchical learning platform, such as HAM. In summary, with a higher average reward and a steeper trend slope than the others, the HDDPG outperforms the HAM and a single DDPG, as shown in Figure 9. The average reward for HAM was 12,500, and the flat trend indicates no evidence of reward gain. The single DDPG improved its reward, with an average reward of 14,500 and a 0.09 slope. However, the HDDPG outperformed both approaches, with an average reward of 20,000 and a reward trend slope of 0.2, indicating that the reward gain is about 20% over time.

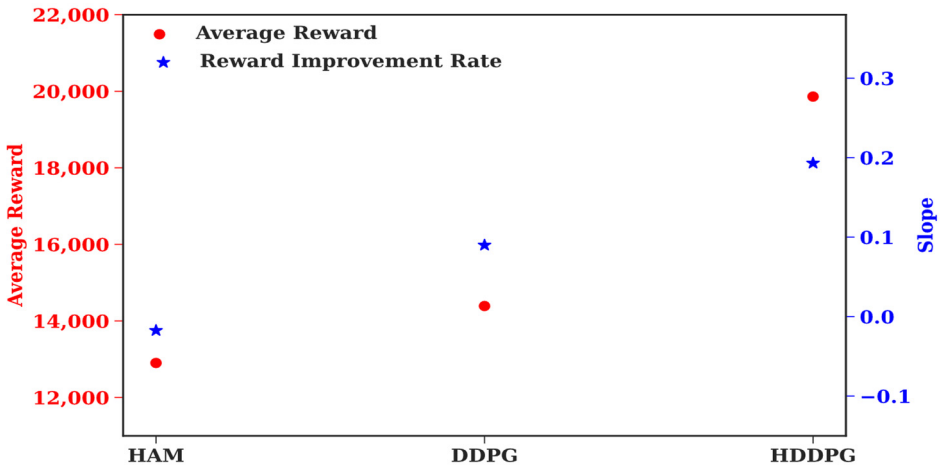

Figure 9. Comparing the average total reward (red dot) and its trend slope (blue star) of the HAM, DDPG, and HDDPG methods.

3.4. Biased Training

Another important feature of the proposed HDDPG is the biased manager training capability. By adding a preference factor to each worker, the manager can be trained to prefer a certain worker to achieve a specific objective more preferably than other objectives if needed. For instance, the manager can be trained to be stricter for collision avoidance

in a human–robot collaboration task to minimize collision with preference factor applied to the obstacle avoidance. In a mission critical task, however, the preference factor can be applied to the path planning worker. In Figure 10, biased manage training results are demonstrated with 15% preference factor for each worker. As illustrated, the goal reward demonstrates dominance compared to that of the obstacle reward in biased training for goal touch preferred case and, in the same manner, the obstacle reward is dominant for biased training for obstacle avoidance preferred case.

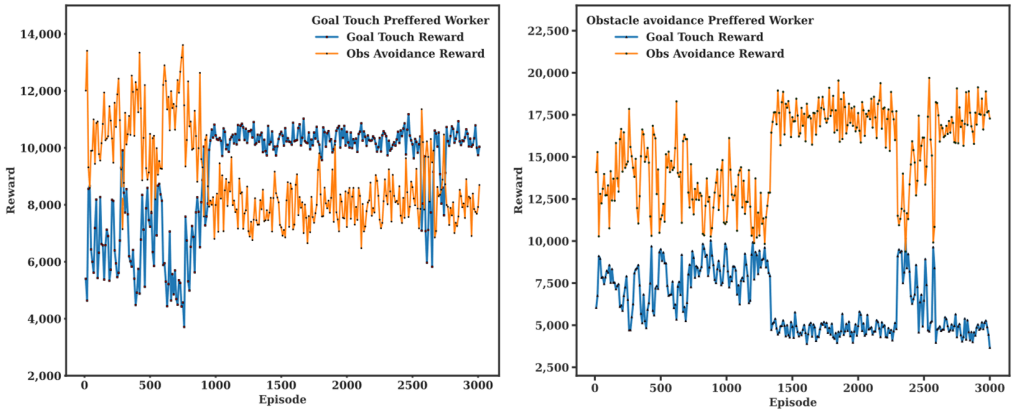

Figure 10. Biased manager training.

4. Conclusions

This paper demonstrates the idea of a novel hierarchical reinforcement learning architecture, namely Hierarchical DDPG (HDDPG). The new architecture is demonstrated with the goal of expanding a reinforcement learning scheme towards a generalized AI through flexibility and expandability. The motivation of the study is from the idea that most complex tasks are composed of multiple conflicting goals staked from simple tasks to highly complex tasks. The proposed HDDPG uses the DDPG as a building block of a generalized AI, sharing the same environment and selected states, but with a designated reward system for each DDPG to create the entirety of a hierarchical structure. Therefore, it is simple in implementation, but powerful to solve a complex task, especially with conflicting sub-goals. In this paper, we use a manager and two conflicting workers to demonstrate the performance of the HDDPG; one worker works to reach a dynamic goal, while another worker works to avoid multiple dynamic obstacles.

The proposed HDDPG is compared to HAM and single DDPG algorithms for performance evaluation. The results show the HDDPG demonstrated more than 38% of reward gain and more than two times the reward improvement rate. Another important feature of the proposed HDDPG is the biased manager training capability. By adding a preference factor to each worker, the manager can be trained to prefer a certain worker to achieve a better success rate for a specific objective if needed. For instance, the manager can be trained to be stricter for collision avoidance in a human–robot collaboration task to improve safety, while, in a mission-critical task, the preference factor can be applied to the path-planning worker.

Author Contributions: Investigation, D.U.; Methodology, P.N.; Resources, H.S.; Validation, D.U.; Visualization, P.N.; Writing—original draft, D.U.; Writing—review & editing, P.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

References

1. Isabelle, B.; Juan, C. The Brain as a Hierarchical Organization. *Am. Econ. Rev.* **2008**, *4*, 1312–1346. [CrossRef]
2. Bill, J.; Pailian, H.; Gershman, S.J.; Drugowitsch, J. Hierarchical structure is employed by humans during visual motion perception. *Proc. Natl. Acad. Sci. USA* **2020**, *117*, 24581–24589. [CrossRef] [PubMed]
3. Andrei, N.; Khimy, K.; Precup, D. The Paradox of Choice: Using Attention in Hierarchical Reinforcement Learning. *arXiv* **2022**, arXiv:2201.09653.
4. Gibson, J.J. *The Theory of Affordances*; Laurence Erlbaum Associates Inc.: Hilldale, NJ, USA, 1977.
5. Shubham, P.; Budhitama, S.; Hwee, T.A. End-to-End Hierarchical Reinforcement Learning with Integrated Subgoal Discovery. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**. [CrossRef]
6. Levy, A.; Robert, P.; Kate, S. Hierarchical actor-critic. *arXiv* **2017**, arXiv:1712.00948.
7. Parr, R.; Russell, S. *Reinforcement Learning with Hierarchies of Machines*; MIT Press: Cambridge, MA, USA, 1997; pp. 1043–1049.
8. Dietterich, T.G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell.* **2000**, *13*, 227–303. [CrossRef]
9. Dayan, P.; Hinton, G.E. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5, NIPS Conference*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1992; pp. 271–278.
10. Bacon, P.L.; Harb, J.; Precup, D. The Option-Critic Architecture. In Proceedings of the 31st Association for the Advancement of Artificial Intelligence conference, San Francisco, CA, USA, 4–9 February 2017.
11. Wang, Z.; Xu, Y.; Wang, D. Hierarchical deep reinforcement learning reveals a modular mechanism of cell movement. *Nat. Mach. Intell.* **2022**, *4*, 73–83. [CrossRef]
12. Kulkarni, T.D. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 3682–3690.
13. Siyuan, L.; Rui, W.; Tang, M.; Zhang, C. Hierarchical reinforcement learning with advantage-based auxiliary rewards. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 8–14 December 2019; Volume 126, pp. 1409–1419.
14. Sungryull, S. Multi-Task Hierarchical Reinforcement Learning for Compositional Tasks. Ph.D. Thesis, University of Michigan, Ann Arbor, MI, USA, 2020.
15. Mnih, V.; Kavukcuoglu, K.; Silver, D. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
16. Puterman, M.L. *Finite-Horizon Markov Decision Processes. Markov Decision Processes: Discrete Stochastic Dynamic Programming*; Wiley-Interscience: New York, NY, USA, 1994; pp. 78–79.
17. Bertsekas, D.P. *Dynamic Programming: Deterministic and Stochastic Models*; Prentice-Hall, Inc.: Hoboken, NJ, USA, 1987.
18. Bellman, R.E. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
19. Lillicrap, T.; Jonathan, H.; Pritz, A.; Heess, N.; Erez, T.; Yuval, T.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
20. Sutton, R.; Doina, P.; Satinder, S. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *J. Artif. Intell.* **1999**, *112*, 1–2. [CrossRef]
21. Barto, A.; Sridhar, M. Recent Advances in Hierarchical Reinforcement Learning. *Discret. Event Dyn. Syst.* **2003**, *13*, 341–375. [CrossRef]
22. Vezhnevets, A.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; Kavukcuoglu, K. FeUdal Networks for Hierarchical Reinforcement Learning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 3540–3549.
23. Sutton, R.; McAllester, D.; Satinder, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99), Cambridge, MA, USA; 1999; pp. 1057–1063.
24. Guo, S.; Zhang, X.; Zheng, Y.; Du, Y. An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning. *Sensors* **2020**, *20*, 426. [CrossRef] [PubMed]
25. Singh, S.; Lewis, R.L.; Barto, A.G. Where Do Rewards Come From? In Proceedings of the International Symposium on AI-Inspired Biology, AISB Convention 2010, De Montfort University, Leicester, UK, 31 March–1 April 2010; pp. 2601–2606.