Online robot guidance and navigation in non-stationary environment with hybrid Hierarchical Reinforcement Learning

Zhou, Ye; Ho, Hann Woei

DOI
10.1016/j.engappai.2022.105152

Publication date
2022

Document Version
Final published version

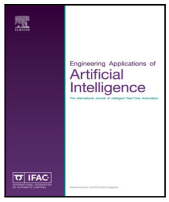
Published in
Engineering Applications of Artificial Intelligence

Engineering Applications of Artificial Intelligence 114 (2022) 105152

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Online robot guidance and navigation in non-stationary environment with hybrid Hierarchical Reinforcement Learning

Ye Zhou *, Hann Woei Ho

School of Aerospace Engineering, Engineering Campus, Universiti Sains Malaysia, 14300 Nibong Tebal, Pulau Pinang, Malaysia
Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629HS Delft, Zuid Holland, The Netherlands

ARTICLE INFO

Keywords:
Hybrid Hierarchical Reinforcement Learning
Online guidance and navigation
Non-stationary environment
Function approximation
State space decomposition

ABSTRACT

Hierarchical Reinforcement Learning (HRL) provides an option to solve complex guidance and navigation problems with high-dimensional spaces, multiple objectives, and a large number of states and actions. The current HRL methods often use the same or similar reinforcement learning methods within one application so that multiple objectives can be easily combined. Since there is not a single learning method that can benefit all targets, hybrid Hierarchical Reinforcement Learning (hHRL) was proposed to use various methods to optimize the learning with different types of information and objectives in one application. The previous hHRL method, however, requires manual task-specific designs, which involves engineers' preferences and may impede its transfer learning ability. This paper, therefore, proposes a systematic online guidance and navigation method under the framework of hHRL, which generalizes training samples with a function approximator, decomposes the state space automatically, and thus does not require task-specific designs. The simulation results indicate that the proposed method is superior to the previous hHRL method, which requires manual decomposition, in terms of the convergence rate and the learnt policy. It is also shown that this method is generally applicable to non-stationary environments changing over episodes and over time without the loss of efficiency even with noisy state information.

1. Introduction

Reinforcement Learning (RL) is a framework of intelligent, self-learning methods, that can be applied to different levels of autonomous operations and applications. It has been successfully applied to solve optimal decision-making problems in known or small-scaled environments (Sutton and Barto, 1998; Bellman, 1957). When applied to complex environments or multi-objective tasks, such as autonomous guidance and navigation problems (Parr and Russell, 1998a; Ni et al., 2013; de Oliveira et al., 2021), RL methods are, however, rendered intractable by the following challenges: (1) the large state and action space, which may lead to the 'curse of dimensionality'; (2) the multiple, conflicting objectives; and (3) the partial observability of the system state and unknown, non-stationary environments. Numerous studies, such as approximate dynamic programming (Si, 2004; Khan et al., 2012; Zhou et al., 2018, 2020; Zhou, 2022), multi-objective optimization strategies (Liu et al., 2015; Van Moffaert and Nowé, 2014; Vamplew et al., 2011; Kim and De Weck, 2006; Lin, 2005), deep reinforcement learning (Mnih et al., 2013, 2015; Silver et al., 2016; Bellemare et al., 2020), and the partially observable Markov decision process framework (Brooks et al., 2006; Scott A. Miller and Chong, 2009; Ragi and Chong, 2013; He et al., 2011; Hoey et al., 2016; Lieck and Toussaint, 2016), have provided successful solutions to part of the aforementioned challenges.

Hierarchical Reinforcement Learning (HRL) is a natural approach to dealing with the 'curse of dimensionality' and multiple objectives, with which a complex problem can be solved by decomposing it into several smaller and simpler problems (Barto and Mahadevan, 2003; Parr and Russell, 1998b; Sutton et al., 1999; Kobayashi and Sugino, 2020; Ma et al., 2020; Eppe et al., 2022). Instead of achieving the global optimality, HRL methods, such as Hierarchical Abstract Machines (HAMs) (Parr and Russell, 1998a,b; Zhou et al., 2016), options (Sutton et al., 1999), MAXQ (Dietterich, 2000; Ghavamzadeh et al., 2006), and HEXQ (Hengst, 2002), aim at reducing the computational cost and can yield a hierarchically optimal policy. Hierarchical decomposition speeds up learning for multi-objective tasks, by allowing different objectives in different levels, and naturally reduces the uncertainty and ambiguity induced by partial observability at high levels (Theocharous and Mahadevan, 2002; Foka and Trahanias, 2007; Sridharan et al., 2010). These HRL approaches, however, often use same or similar RL methods within one application, which prevents the exploitation of various methods. To improve the learning efficiency, a

* Corresponding author at: School of Aerospace Engineering, Engineering Campus, Universiti Sains Malaysia, 14300 Nibong Tebal, Pulau Pinang, Malaysia.
E-mail addresses: zhouye@usm.my (Y. Zhou), aehannwoei@usm.my (H.W. Ho).

https://doi.org/10.1016/j.engappai.2022.105152
Received 10 November 2021; Received in revised form 14 June 2022; Accepted 27 June 2022
Available online xxxx
0952-1976/

Abbreviations

RL	Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
hHRL	hybrid Hierarchical Reinforcement Learning
MC	Monte Carlo
TD	Temporal-Difference
GPS	Global Positioning System
OLS	Ordinary Least Square
RLS	Recursive Least Square
MSE	Mean Squared Error
HAMs	Hierarchical Abstract Machines

recent research proposed to allow for hybrid methods, state information, and rewards in multi-objective problems, namely hybrid Hierarchical Reinforcement Learning (hHRL) (Zhou et al., 2019). This method was validated on online guidance and navigation tasks where the environment is complex, partially observable, and a priori unknown. The simulation results indicated that the hHRL method, compared to flat or non-hybrid methods, can help to accelerate learning, to alleviate the 'curse of dimensionality' in complex decision-making tasks, and to reduce the uncertainty or ambiguity with different levels more efficiently.

However, the hHRL algorithm, same as most current HRL methods, also requires certain degree of task-specific manual designs, which involves engineers' preferences and impedes the transfer of learning from one application to another even similar application (Barto and Mahadevan, 2003). Specifically, in guidance and navigation problems, the state abstraction or environment decomposition can be decided in advance based on the a priori information about the environment. For example, in many indoor navigation tasks, the environment can be decomposed into several sub-environments based on physical isolation, such as floors, rooms, corridors, etc. for easier coarse navigation and high-level instructions (Theocharous and Mahadevan, 2002). For more general environments, such as in a maze navigation problem, as shown in Fig. 1, which does not have clear, observable isolation, it can be decomposed into grids based on physical distances (Zhou et al., 2019; Theile et al., 2020). Whereas, the size and shape of the grids are often assigned manually with certain a priori knowledge and may not be successful when transferred to other environments. Furthermore, the current hHRL algorithm has to learn a map from the sensor information, which is not memory-efficient and hard to extend to real-world scenarios. In the real world, mobile robots will navigate in environments with static obstacles, such as walls, and non-stationary obstacles, such as chairs and tables. To draw the map, the agent needs to detect the type of obstacles and only include the static obstacles in the map, which is inefficient in practice.

The aim of this paper is to address the aforementioned knowledge gaps by developing a more general and data-efficient hHRL method for online guidance and navigation problems, more specifically, by exploiting function approximation in high-level decision-making to automatically and intrinsically decompose the environment. The instruction, which is usually a direction in guidance and navigation problems, will be given based on the absolute state in the high level, and the estimated value of certain state will be generalized to the surrounding environment. This generalization process will automatically and dynamically form boundaries between macro states during the learning process. As suggested by early research (Vezhnevets et al., 2017; Nachum et al., 2018), the intrinsic reward that encourages the low-level agent to follow the instruction can be defined as a function of both the instructions and the resulted states, which will evaluate both the correctness and efficiency of the low-level policy. Similarly,

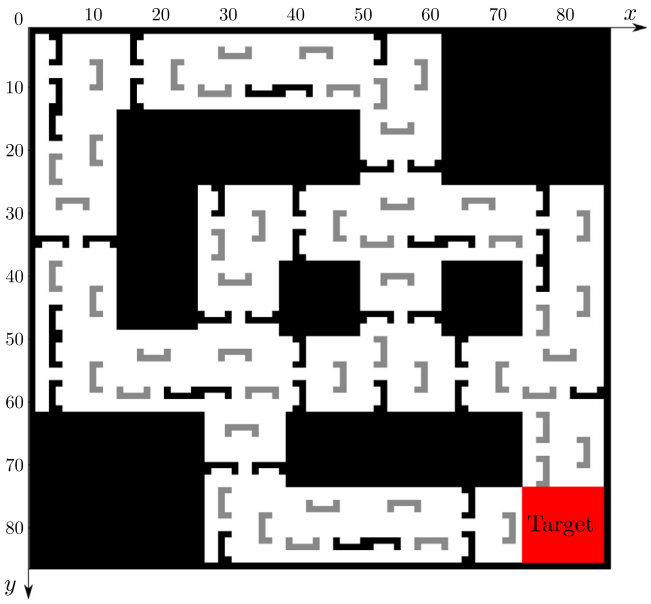


Fig. 1. A guidance and navigation problem with obstacles and a target in a non-stationary maze environment. The black color represents stationary obstacles, while the gray color represents non-stationary obstacles.

when the low-level policy does not follow the high-level instruction, the intrinsic reward function can be used to relabel the past high-level instructions for a more efficient learning process.

The **main contribution** of this paper is the development of a general, extendable online guidance and navigation algorithm under the framework of hHRL, which does not require task-specific designs and can be used in non-stationary environments. In contrast to our prior work (Zhou et al., 2019) that decomposes the environment manually into several smaller spaces, this algorithm uses a continuous state space representation for the generalization and scalability of guidance and navigation problems. In addition, this algorithm, instead of manually assigning supplemental penalties, uses on-policy learning for obstacle avoidance with only collision penalties. Thus, it can be used in online guidance and navigation problems, such as the benchmark Parr's maze navigation problem (Parr and Russell, 1998b), but with the non-stationary environment.

The remainder of this paper is structured as follows. Section 2 starts with an introduction to the hHRL concept and describes the framework for solving guidance and navigation problems. Section 3 presents the implementation details of the proposed algorithm, including each value function and the adaptation rules. Then, Section 4 validates the proposed algorithm in online guidance and navigation problems with non-stationary environments and measurement noises. Lastly, Section 5 shows the advantages and disadvantages of using the proposed algorithm and addresses the challenges and possibilities of future research.

2. Hybrid hierarchical reinforcement learning for guidance and navigation

Hybrid Hierarchical Reinforcement Learning (hHRL) was proposed (Zhou et al., 2019) to allow for different methods and types of state information in different levels and sub-tasks. By following the hHRL concept and rules, an online guidance and navigation algorithm has been established to deal with multiple objectives, partial observability, unknown environments, and learning across tasks. This algorithm has been successfully applied to several maze navigation problems with localization and mapping. However, this mapping method is not memory-efficient and hard to extend to real-world scenarios, especially

non-stationary environments. The state space decomposition also requires certain degree of manually task-specific designs, although it is proved insensitive to the design. This paper proposed a modified online guidance and navigation algorithm under the framework of hHRL, which does not require manually defined task-specific designs, and can be used in non-stationary environments. This section will briefly introduce the hHRL concept and the modifications to the previous algorithm.

2.1. Hybrid learning

HRL is a natural approach to solving complex problems that can be decomposed into several smaller problems with different targets. For each smaller problem, the agent needs to choose a specific RL method to evaluate and improve the policy with a value function and/or a policy function. To ease the assimilation of multiple targets, current HRL approaches often use the same or similar methods within one application. However, there is no single method that can benefit all targets (Wolpert and Macready, 1997; Takamuku and Arkin, 2007; Tan et al., 2011; Abbaszadeh Shahri and Maghsoudi Moud, 2021), which is also applicable to HRL. The choice of methods will depend on the type of tasks and rewards, the attributes of the state and action spaces, etc. For example, in different hierarchies and sub-tasks, the environment will often return different types of rewards, such as a single final reward, immediate rewards, and delayed rewards, where the Monte Carlo (MC) method, Temporal-Difference (TD) learning, or a joint method is desirable, respectively. In addition, the state and action spaces in different levels can be discrete, continuous, or even hybrid, which may also imply different methods. More specifically, some level has a complex, continuous state–action space, where the actor–critic method might be more useful, while another level may have a discrete action space, where Q-learning or SARSA may be more efficient.

Therefore, hHRL was proposed to better use the acquired information and appropriate methods to effectively tackle each sub-task. In other words, different sub-tasks are allowed to use different learning methods, state information, learning types, reward assignment systems, as long as the learned result can be assimilated within each hierarchy (Zhou et al., 2019).

2.2. Hierarchical structure

For online guidance and navigation in a priori unknown, non-stationary environments, decomposition of tasks and abstraction of actions allow agents to solve current sub-problems and to ignore irrelevant details at the current level. This paper adopts a general two-layer guidance and navigation hierarchical structure, as shown in Fig. 2: a higher-level policy decides the direction of the movement towards the target area, and a lower-level policy decides each primitive actions in the non-stationary environment to prevent collision while following the higher-level instruction. The higher-level policy, in every τ time steps, estimates the global or absolute state and produces a coarser-grained instructional *behavior*. This *behavior* $\boldsymbol{b} \in \mathcal{B}$: {[0, −1], [−1, 0], [0, 1], [1, 0]} represents a direction {go North, go West, go South, go East}, where the x axis is pointing to East and the y axis is pointing to South, as seen in Fig. 1. The direction is represented by a unit vector in the task domain and can be easily extended to three-dimensional or continuous guidance and navigation problems. The lower-level policy, after receiving the higher-level instruction $\boldsymbol{b}_t$, observes the relative states in terms of obstacles at each time step and produces *primitive actions* $a_t, a_{t+1}, \ldots, a_{t+\tau-1}$, where $a \in \mathcal{A}$: {turn left, turn right, move forward}. This can be broken down into two sub-tasks: the first sub-task (L1) is to follow the higher level instruction, while the second sub-task (L2) is to prevent collision with the wall or obstacles. These sub-tasks in the same level are trained separately for the ease of utilizing different types of rewards and of transferring the learned results to different scenarios or tasks.

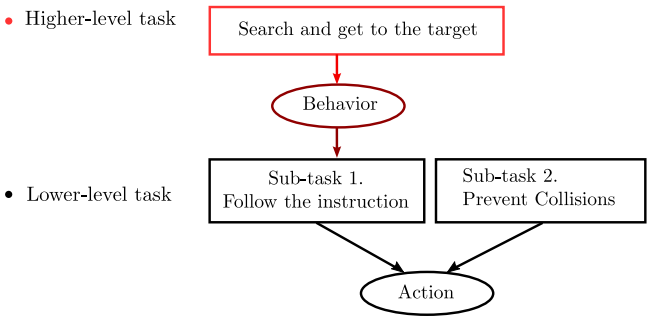


Fig. 2. A two-layer guidance and navigation hierarchical structure.

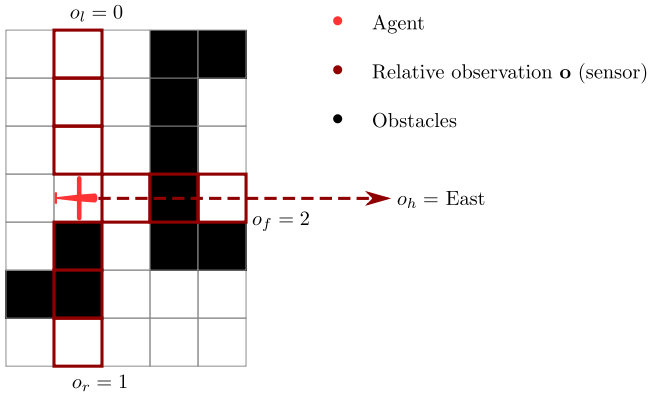


Fig. 3. An illustrative example of the agent with a relative observation, $\boldsymbol{o} = [o_f, o_l, o_r, o_h]$, in a discrete environment.

Instead of manual environment decomposition (Zhou et al., 2019), this paper uses a continuous state-space representation for the sake of the generalization and the scalability of any guidance and navigation environments. The adaptation of a single state–action value will be generalized to surrounding states naturally by using value function approximators. Therefore, this method can be used in both discrete domain and continuous domain without preset boundaries or sizes.

2.3. Relative and absolute states

In guidance and navigation problems, the agent often has an accurate relative state from cameras, sonar sensors, infrared systems, etc., which is a local observation $\boldsymbol{o}$ of its surroundings for accurate operations. To perform global tasks, the agent also needs to know the absolute state, which is the unique, absolute position of the agent in the task environment, $\boldsymbol{p}$. It can be obtained by using a Global Positioning System (GPS), calculating from accelerometers, recognizing from features, or estimating with several techniques. This information is often inaccurate but can be very useful for a high-level decision-making.

Same as the previous work (Zhou et al., 2019), the agent represents a mobile robot and has a relative observation, $\boldsymbol{o} \in \mathcal{O}$: 3-step short-sight (obstacle in the 1, 2, 3-step away or nothing) in 3 directions (front, left, and right), represented by $o_f \in \mathcal{O}_f$, $o_l \in \mathcal{O}_l$, and $o_r \in \mathcal{O}_r$, and its heading angle $o_h \in \mathcal{O}_h$ representing {East, South, West, North}, as shown in Fig. 3. Thus, this relative state has only 256 possibilities. Although relative state brings ambiguity into maze problems, it also, to some extent, limits the growing of the state space and consumes less computational and memory resources. In addition, the relative state information can directly meet the objective of preventing collisions and remains flexible in unknown, non-stationary environments. Therefore, the relative state will be used in the lower-level.

Absolute position has its own benefit in target approaching tasks, especially for small-scaled problems. With local information of relative

Algorithm 1 A two-level hHRL algorithm for online guidance and navigation

Start the ith episode.
while not reach the target area **do**
  Observe absolute position $\boldsymbol{p}_t$, choose $\boldsymbol{b}_t$ using behavior policy with $Q_H(\boldsymbol{p}_t, \boldsymbol{b}; \Theta_i)$ and ϵ-greedy.
  Receive instruction $\boldsymbol{b}_t$, observe relative state $\boldsymbol{o}_l$, choose action a_l
  for $l \in [t, t+\tau)$ **do**
    Take action a_l, observe immediate reward r_l and next state $\boldsymbol{o}_{l+1}$.
    Choose action a_{l+1} using action policy $Q_L(\boldsymbol{o}_l, \boldsymbol{b}_t, a)$ and ϵ-greedy.
    Update $Q_{L2}(\boldsymbol{o}_l, a_l)$ with SARSA.
    $\boldsymbol{o}_l \leftarrow \boldsymbol{o}_{l+1}$, $a_l \leftarrow a_{l+1}$
  end for
  Determine the intrinsic reward $\varrho_{t+\tau}$.
  Update $Q_{L1}(\boldsymbol{o}_l, \boldsymbol{b}_t, a_l)$ for all $(\boldsymbol{o}_l, \boldsymbol{b}_t, a_l)$ in $[t, t+\tau)$ with batch offline Q-learning
  Relabel the high-level behavior $\check{\boldsymbol{b}}_t$ and save the experience sample $(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t, \boldsymbol{p}_{t+\tau})$
end while
Update the parameters in Q_H for all visited $(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t)$ with discounted reward MC and obtain Θ_{i+1}.
$\Theta_i \leftarrow \Theta_{i+1}$

observations, the agent may perform obstacle avoidance, and can estimate its location in the discrete maze with features and memorized maps (Zhou et al., 2019). However, this mapping method is based on a manual decomposition of the environment and is not memory-efficient. And it is hard to extend to real-world scenarios, especially continuous or non-stationary environments. For a possible future extension to real-world applications, the agent will receive an inaccurate absolute position $\boldsymbol{p} \in \mathcal{P}$: position in the maze along the x and y axes, represented by $[p_x, p_y]$. The accessibility of each state in this non-stationary environment may vary with time, and the amount of absolute states can ever-increase due to the increasingly expanded environment. Therefore, the acquired absolute state cannot be the only deciding factor for actions but will be used to decide the high-level behavior.

3. Implementations

This section presents the detailed implementation of the proposed hHRL algorithm for online guidance and navigation, focusing on the value functions and their adaptation in each level. As shown in Fig. 1, the navigation environment has stationary and non-stationary obstacles and a target area. And the agent only has a limited set of actions $\mathcal{A}$ (see Zhou et al. (2019) for the detailed description of its mobility). The agent does not know the map of this maze but will interact with the environment by receiving an *immediate penalty* $r = -2$ when it collides with obstacles, and a *final reward* $R = 100$ when it reaches the target area. To encourage the low-level agent to follow the instruction, the higher-level behaviors $\boldsymbol{b}_t$ will be defined as the goals for the low-level agent, and the reward will be given based on the correctness and efficiency of the low-level policy using an *intrinsic reward* function ϱ:

$$\varrho_{t+\tau}(\boldsymbol{p}_t, \boldsymbol{p}_{t+\tau}, \boldsymbol{b}_t) = \frac{(\boldsymbol{p}_{t+\tau} - \boldsymbol{p}_t) \cdot \boldsymbol{b}_t^T}{\tau}. \tag{1}$$

In the lower level, as shown in Fig. 2, the agent will make a decision in order to (1) follow the higher-level instruction and (2) to prevent collisions. The evaluations of these two sub-tasks are based on intrinsic rewards ϱ evaluated at every time τ, $\tau = 30$ in this implementation, and collision penalties r at every time step, respectively. Their associated state variables are also different, e.g., the heading of the agent may not relate to collisions but will influence the direction of its trajectory. To improve the learning efficiency, these two sub-tasks will be trained separately with different RL methods, state information, and rewards under the hHRL framework. The trained result should be in the same structure, i.e., state values or state–action values, so as to be summed up to decide what actions to take. The learning approaches of the hHRL method is presented in Algorithm 1.

3.1. Higher-level policy learning

The higher-level behavior is trained with discounted reward MC episode by episode. When an episode ends at time T, the agent will receive a final reward $R = 100$ if it reaches the target, and the value function will be updated. The adaptation is based on the trace-back of the experienced higher-level transition tuples $(\boldsymbol{p}_t, \boldsymbol{b}_t, \boldsymbol{p}_{t+\tau})$ stored in a replay buffer. However, the transitions obtained in the past episode do not necessarily reflect the lower-level policy in the next or future episodes (Nachum et al., 2018; Fujimoto et al., 2018). This will deteriorate the learning efficiency and still remains as an issue in HRL implementations. One practice is to convert the experienced behaviors $\boldsymbol{b}_t$ into the ones that agree with the lower-level policy in the future $\check{\boldsymbol{b}}_t$. A recent research (Nachum et al., 2018) calculated the probabilities of a number of candidate higher-level targets under the current lower-level policy and chose the maximal goal to relabel the experience. This method can be used for an infinite number of goals or stochastic processes but requires high computational resources. Our previous work (Zhou et al., 2019) drew a memory map and correct the behavior based on the decomposed state experience. This method assumes that the lower-level policy will converge to the perfect one ultimately and needs less computations, but requires manual decomposition of the environment and is limited to a finite number of behaviors.

This paper uses a new strategy to relabel the behavior based on the reward. Since the lower-level policy is updated step-wise, the current policy will only valid for one time step and will vary with time during the next episode. In addition, the environment is non-stationary, which will also affect the action selection with the same behavior and lower-level policy. On the other hand, it is known that the lower-level policy will be updated to maximize the intrinsic reward ultimately. Therefore, the behavior which maximizes the reward can be used to relabel the experienced transition tuples $(\boldsymbol{p}_t, \boldsymbol{b}_t, \boldsymbol{p}_{t+\tau})$, as follows:

$$\check{\boldsymbol{b}}_t = \arg\max_{\boldsymbol{b} \in \mathcal{B}} \varrho(\boldsymbol{p}_t, \boldsymbol{p}_{t+\tau}, \boldsymbol{b}). \tag{2}$$

This method is computationally efficient and can be used for an infinite number of behaviors without using the current policy or a perfect model. And consequently, the training in different levels will be independent to each other and converge to a sub-optimal solution intuitively.

The MC offline learning can be used for backwards adaptation through the list of relabeled transition pairs $(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t)$ with a return G (Sutton and Barto, 1998). For a better denotation, the sequence of the experienced transition pairs $(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t)$ in the list is referred as k_t. Then, this return is initialized with $G(k_T) = 0$ and updated backwards through the transition pairs with $G(k_t) = r + \gamma G(k_t + 1)$, where r is an immediate reward, and $\gamma \in (0, 1)$ is the discount factor. Because the high-level agent may only receive a final reward $R = 100$, the returns for the

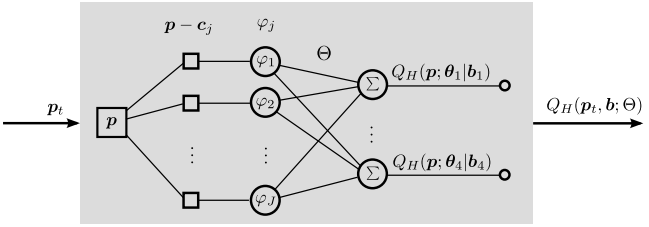


Fig. 4. The Q network with RBF activation functions, where the output of each square unit is a vector. This Q network inputs the absolute position $\boldsymbol{p}$ and outputs its Q values for all the valid behaviors $\boldsymbol{b} \in \mathcal{B}$: $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3, \boldsymbol{b}_4\}$.

experienced transition pairs are $G(k_t) = \gamma^{k_T-1-k_t} R$ reaching the target, and $G(k_t) = 0$ not reaching the target. And the Q value of experienced transition pairs in this episode $Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t)$ can be updated towards the returns $G(k_t)$ with a learning rate $\alpha \in (0, 1)$, as follows:

$$Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t) \leftarrow (1-\alpha) Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t) + \alpha G(k_t). \tag{3}$$

Therefore, the final reward can be propagated backwards efficiently.

3.2. Higher-level policy generalization

Guidance and navigation problems often have large state spaces, which can ever-increase due to the increasingly expanded environment of applications. And the transition pairs experienced and updated in the high level are very limited, and most absolute states in the state space have never been experienced exactly before. Our previous work (Zhou et al., 2019) decomposed the state space into macro states (grids) and updated the Q value for each of those states. This paper, instead of manually partitioning the state space, converts the discrete maze environment into continuous state space and generalizes the Q values of the discrete experienced samples using a function approximator with parameters $Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t; \Theta) \approx Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t)$. Because the high-level agent only receives a final reward and updates the Q values when an episode terminates, the Q value adaptation in Eq. (3) can be rewritten as

$$Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t; \Theta_{i+1}) \leftarrow (1-\alpha) Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t; \Theta_i) + \alpha G(k_t), \tag{4}$$

where i denotes that the parameters Θ_i are used in the ith episode.

Since the high-level Q values are updated in batches, it is possible to use any existing supervised learning methods to approximate the Q-value function, such as multivariate regression, artificial neural networks, and kernel methods. This paper uses a Radial Basis Function (RBF) network for the generalization, which can be used for both the large-scale global approximation or local fine-tuning. As illustrated in Fig. 4, the input of the Q network is the absolute position $\boldsymbol{p} : [p_x, p_y]$. The hidden layer has J Gaussian radial functions, whose value depends on the distance between the input to a center point $\|\boldsymbol{p} - \boldsymbol{c}_j\|$ as below:

$$\varphi_j(\boldsymbol{p}) = e^{-(\|\boldsymbol{p}-\boldsymbol{c}_j\|/r_j)^2}, \tag{5}$$

where $\boldsymbol{c}_j$ is the jth center point, and r_j denotes its radius. And the output of the hidden layer can be denoted as $\boldsymbol{\varphi}(\boldsymbol{p})$, where $\boldsymbol{\varphi} : \mathcal{R}^2 \rightarrow \mathcal{R}^J$. The output layer of this Q network is a fully-connected linear layer for all the valid behaviors $\boldsymbol{b} \in \mathcal{B}$: $\{\boldsymbol{b}_1 = [0, -1], \boldsymbol{b}_2 = [-1, 0], \boldsymbol{b}_3 = [0, 1], \boldsymbol{b}_4 = [1, 0]\}$. When $\boldsymbol{c}_j$ and r_j are fixed, the network output is linear to the parameters $\Theta = [\boldsymbol{\theta}_1\ \boldsymbol{\theta}_2\ \boldsymbol{\theta}_3\ \boldsymbol{\theta}_4] \in \mathcal{R}^{J*4}$, which are the weights connecting the output of each radial functions to each single output of the Q network.

This Q network is nonlinear in the input, but linear in the parameters. And there are several ways to update the network parameters. According to Eq. (4), the direct method is to minimize the error using Ordinary Least Square (OLS) in batches. And it will find the unique optimal parameter values. However, the OLS method needs to do a matrix inversion at each update. If there is not enough effective data, the matrix might not be invertible, and the parameters cannot be identified. In addition, the adaption with a batch of Q values only in the current episode will totally forget the previous learned parameters. Therefore, we can use the Recursive Least Square (RLS) method with a forgetting factor or the gradient-descent method with a step-size parameter, so that the training will not only concern the experience samples in the current episode.

In this paper, we use the gradient-descent method because it is more computationally efficient and generally applicable for both linear and nonlinear approximators. It tries to minimize the Mean Squared Error (MSE) of the current Q values $Q_H(\boldsymbol{p}_t, \check{\boldsymbol{b}}_t; \Theta_i)$ and the calculated returns $G(k_t)$ on the experienced samples. Instead of using a learning rate in Eq. (4), we use a step-size parameter η to reduce the MSE for each valid behavior along the negative gradient:

$$\boldsymbol{\theta}_{\imath,i+1} = \boldsymbol{\theta}_{\imath,i} + \eta \sum_{\check{\boldsymbol{b}}_t = \boldsymbol{b}_\imath} [G(k_t) - \boldsymbol{\theta}_{\imath,i}^T \boldsymbol{\varphi}(\boldsymbol{p}_t)] \cdot \boldsymbol{\varphi}(\boldsymbol{p}_t), \tag{6}$$

where $\imath$ denotes the $\imath$th valid behavior.

3.3. Lower-level learning to follow instructions (L1)

After receiving the behavior $\boldsymbol{b}_t$, the lower-level will execute the current policy to generate a sequence of tuples $(\boldsymbol{o}_l, a_l, \boldsymbol{o}_{l+1})$ and will receive an intrinsic reward $\varrho_{t+\tau}$ after the termination of the behavior, as in Eq. (1). This reward is defined as the projection of the travel distance within time $[t, t+\tau)$ to the unit vector $\boldsymbol{b}_t$ divided by τ, which reflects the resulted movement of the sequential state–action pairs in a time step. This intrinsic reward is used to train the action value function $Q_{L1}(\boldsymbol{o}, \boldsymbol{b}, a)$ with batch offline Q-learning, as follows:

$$\begin{aligned} Q_{L1}(\boldsymbol{o}_l, \boldsymbol{b}_t, a_l) \leftarrow\ & (1-\alpha) Q_{L1}(\boldsymbol{o}_l, \boldsymbol{b}_t, a_l) \\ & + \alpha \left[\varrho_{t+\tau} + \gamma \max_{a \in \mathcal{A}} Q_{L1}(\boldsymbol{o}_{l+1}, \boldsymbol{b}_t, a) \right], l \in [t, t+\tau). \end{aligned} \tag{7}$$

3.4. Lower-level learning to prevent collision (L2)

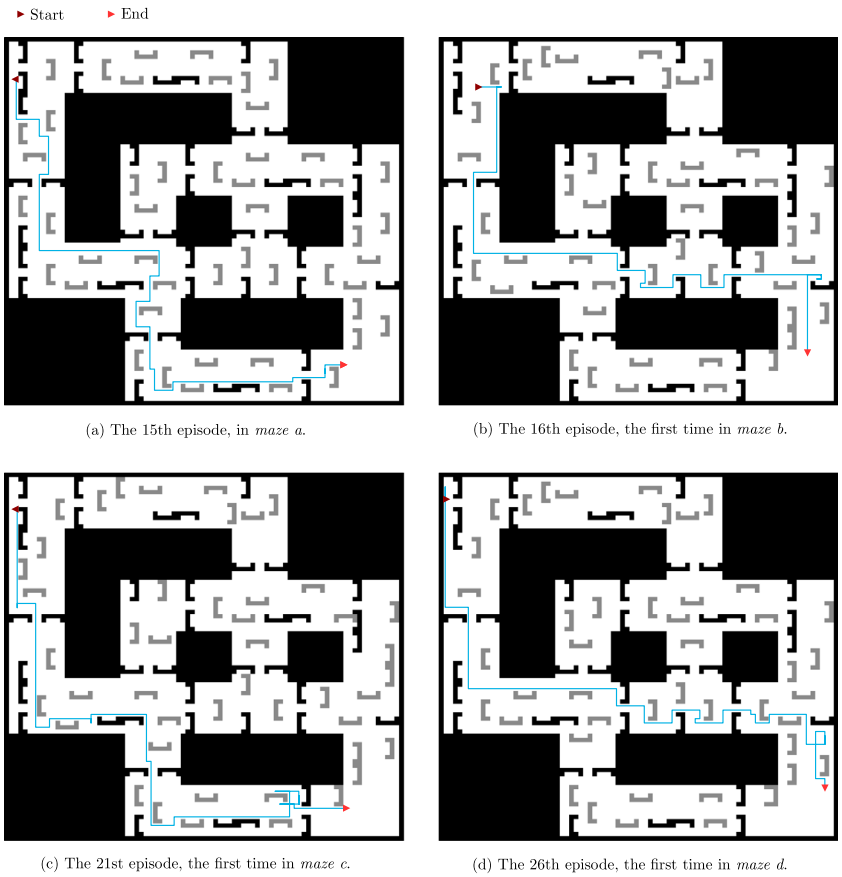
The second sub-task in the lower-level is learning to prevent collisions online. After taking an action, the agent may receive an immediate penalty $r = -2$ from the environment if there is a collision. The lower-level policy also keeps exploring with an ϵ-greedy strategy. If using one-step off-policy methods, previous actions, such as pushing the agent towards an obstacle, will not receive any penalty. An option is to use eligibility traces concerning n-step backups, such as Q(λ), which will bridge the gap between the action and future penalties. Unfortunately, off-policy Q(λ) needs to cut off traces when an exploratory action is taken and thus will lose much of its advantage (Sutton and Barto, 1998). Therefore, this paper uses the easy-to-implement SARSA, an on-policy method, to train the action value function for this sub-task. The collision penalty, if happens in the future, will also be backtracked through the Markov chain.

This task does not directly relate to the current behavior $\boldsymbol{b}_t$ or even the agent's heading $o_h \in \mathcal{O}_h$, which can, therefore, reduce the dimension of the value function to $\mathcal{O}_f \times \mathcal{O}_l \times \mathcal{O}_r \times \mathcal{A}$ in the implementation. The action value function can be written as $Q_{L2}(o_f, o_l, o_r, a)$, but we will keep the notation $Q_{L2}(\boldsymbol{o}, a)$ for simplicity in the rest of this paper. The value function is updated after each quintuple of experiences $(\boldsymbol{o}_t, a_t, r_{t+1}, \boldsymbol{o}_{t+1}, a_{t+1})$ using SARSA, as follows:

$$Q_{L2}(\boldsymbol{o}_t, a_t) \leftarrow (1-\alpha) Q_{L2}(\boldsymbol{o}_t, a_t) + \alpha \left[r_{t+1} + \gamma Q_{L2}(\boldsymbol{o}_{t+1}, a_{t+1}) \right]. \tag{8}$$

3.5. Assimilation of learned results

In this maze navigation problem, a two-layer hierarchical structure is applied, and different methods are selected according to different types of sub-tasks, rewards, and state–action spaces. At the high level, the agent learns to search and get to the target by implementing the offline MC algorithm with a continuous Q network. This method has high data efficiency and learning speed in these episodic tasks that



(a) The 15th episode, in *maze a*.

(b) The 16th episode, the first time in *maze b*.

(c) The 21st episode, the first time in *maze c*.

(d) The 26th episode, the first time in *maze d*.

Fig. 5. The traces of the agent in non-stationary environments, where the gray color represents non-stationary obstacles and may change over episodes.

only receive a single final reward. At the low level, the agent applies offline, off-policy Q-learning to perform the instruction following sub-task. By propagating the reward back, this method will learn from delayed intrinsic rewards in batches more efficiently. And to tackle the preventing collision sub-task, the agent train the value function using online, on-policy SARSA with immediate penalties. This method will naturally backtrack the future collision penalty through the Markov chain.

The high-level policy, at time t, decides the instructional behavior $\boldsymbol{b}_t$ and passes it to the low-level sub-task: following the instruction (L1). And based on this instruction, the agent, at time $l \in [t, t+\tau)$ with observation $\boldsymbol{o}_l$, will calculate the Q-values for all the possible actions with $Q_{L1}(\boldsymbol{o}_t, \boldsymbol{b}_t, a)$ in evaluation of this sub-task. Similarly, the Q-values $Q_{L2}(\boldsymbol{o}_t, a)$ can be calculated to evaluate the values for the second sub-task in the same level: preventing collisions (L2). Following the rules of hHRL (Zhou et al., 2019), the agent, at time $l \in [t, t+\tau)$ with observation $\boldsymbol{o}_l$ and instruction $\boldsymbol{b}_t$, will assimilate the Q-values for possible actions and find the greedy action, as follows:

$$\begin{aligned} a^* &= \arg\max_{a\in\mathcal{A}} Q_L(\boldsymbol{o}_l, \boldsymbol{b}_t, a) \\ &= \arg\max_{a\in\mathcal{A}} [Q_{L1}(\boldsymbol{o}_t, \boldsymbol{b}_t, a) + Q_{L2}(\boldsymbol{o}_t, a)], \text{for } a \in \mathcal{A}. \end{aligned} \tag{9}$$

The value functions for each sub-task can be evaluated independently and asynchronously, which eases the training process and the transfer of learning to other guidance and navigation tasks and other systems.

In addition, this method does not sum up the complete value functions within the whole state space or state–action space, which may vary among different methods, but only the values for the current possible actions within the action sub-space. In other words, the state spaces in different sub-tasks in the same level does not need to be the same, which allows the agent to utilize the state information more efficiently. For instance, the state information used in the first sub-task is the instruction and relative states with heading $\mathcal{O}_f \times \mathcal{O}_l \times \mathcal{O}_r \times \mathcal{O}_h \times \mathcal{B}$. But the instruction and the agent's heading will not directly affect the evaluation of the second sub-task, thus the state space is reduced to $\mathcal{O}_f \times \mathcal{O}_l \times \mathcal{O}_r$. This trait makes the hHRL method allows for different methods in sub-tasks within the same level.

4. Numerical experiments

In this section, two different scenarios will be simulated to validate the proposed algorithm for online guidance and navigation problems in non-stationary environments. First, this method will be applied to the maze with non-stationary environments, which varies over episodes, and the result will be compared to the one using the manual decomposition (Zhou et al., 2019). Second, this method will be applied to the maze with non-stationary obstacles, which may change from time to time. In addition, high-frequency measurement noises will be superimposed to the measurements of the absolute position to further validate the robustness of the proposed method. Note that all the policies will follow an ϵ-greedy strategy, and the exploration rate used in this paper is initialized as $\epsilon_0 = 0.9$ and is decayed by 0.9 after each update until it reaches the minimum exploration rate $\epsilon = 0.1$.

4.1. Non-stationary environment over episodes

The maze is a priori unknown to the agent, and the initial position may vary from episode to episode. These simulations use pseudo-random initial positions (pre-determined and in the northeast area in the maze) for 30 episodes. The agent starts its exploration with the original maze in Fig. 1, denoted as maze a, for 15 episodes, and from the 16th episode onwards, the non-stationary obstacles may change their position every 5 episodes. Fig. 5(a) shows the agent's trace in the 15th episode in *maze a*. The environment changes to *maze b*, *maze c*,

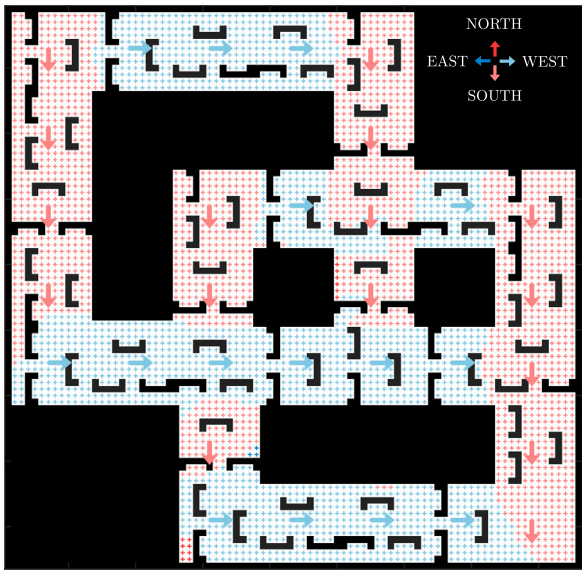


Fig. 6. The greedy behaviors after 30 episodes presented in the original Parr's maze. The colored dots represent the greedy behaviors for each absolute position.

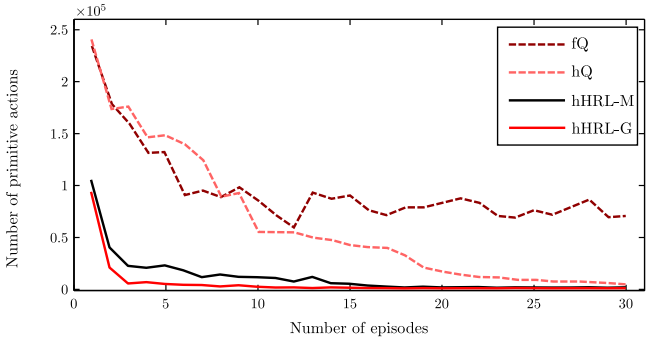


Fig. 7. The number of primitive actions taken in each episode in a non-stationary environment changing over episodes (averaged of 10 runs). The algorithms fQ and hQ have the same agent and the same initial maze, but the obstacles are all stationary. The hHRL-M algorithms decompose the environment to macro states manually and uses a partial map, and hHRL-G is the proposed algorithm without task-specific design or memorizing the map.

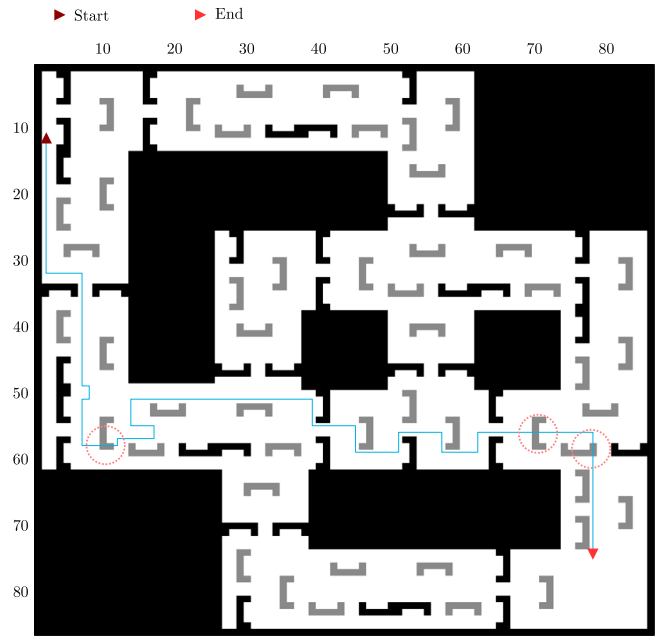


Fig. 8. The trace of the agent in the non-stationary environment, where the gray obstacles may appear intermittently within an episode. The pink dotted lines encircle the traces that go upon the non-stationary obstacles' states when they are not there.

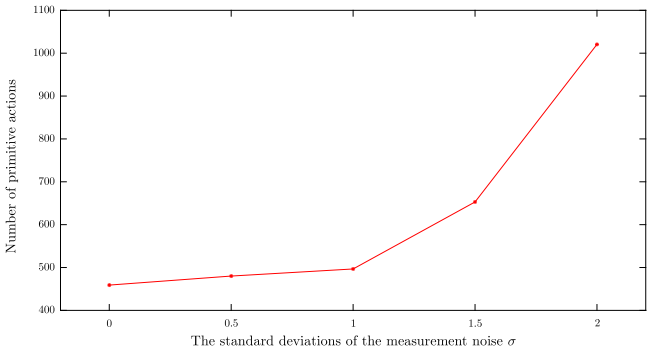
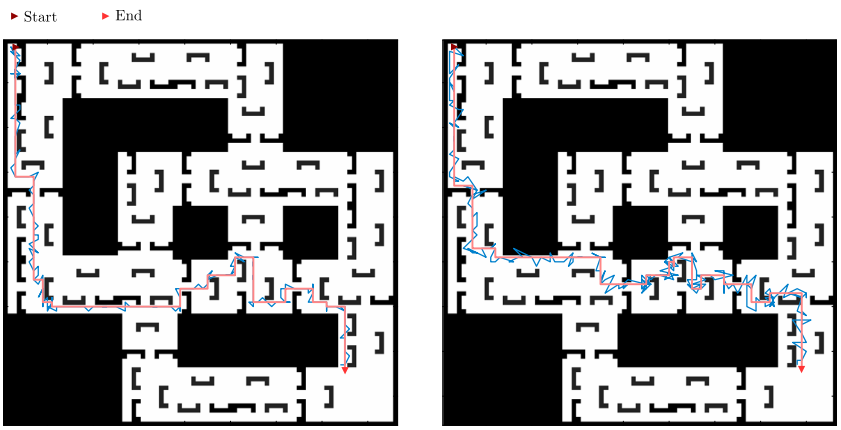


Fig. 9. The averaged number of primitive actions from the 21st episode onwards with noisy absolute positions (averaged of 10 runs). The simulated white noise has a Gaussian distribution $\mathcal{N}(0,\sigma)$ with different standard deviations $\sigma = 0.5, 1, 1.5, 2$.

and *maze d* in the 16, 21, and 26th episodes for the first time, where the traces of the agent are recorded in Figs. 5 (b), (c), and (d), respectively. Note that the policy will follow an ϵ-greedy strategy with a minimum exploration rate $\epsilon = 0.1$. Therefore, random actions or behaviors can be observed in the traces of the agent from the result. The greedy behaviors after 30 episodes are presented in the original Parr's maze in Fig. 6. The colored dots in the map represent the greedy behaviors for each absolute position.

The proposed method in this paper, which is Generally applicable and denoted as *hHRL-G*, is compared to the previous hHRL algorithm (Zhou et al., 2019) with Manual decomposition of the state space, denoted as *hHRL-M*, in this experiment. Note that the hHRL-M algorithm in the previous study (Zhou et al., 2019) did not know the absolute position of the agent from the sensor, but memorized its trajectory and observations and simultaneously drew a partial map consisting of only stationary obstacles. And in the next episode, the agent will compare its current trace with the partial map to localize itself in the map and to estimate its macro state. The non-stationary obstacles may vary from episode to episode and cannot be used as a map for the next episode comparison. Therefore, it requires the agent to use extra sensors or analysis to tell whether the obstacles are stationary, such as walls, or non-stationary, such as chairs and tables.

For a fair comparison, the hHRL-M algorithm will also receive an absolute position and use it to directly determine its macro state. And in this experiment, the sensor information for both hHRL-G and hHRL-M is accurate. In addition, the results are also compared with 'flat' Q-learning (*fQ*) and hierarchical Q-learning (*hQ*), which have a hierarchical structure but only using Q-learning for all sub-tasks (Zhou et al., 2019, 2016). These two algorithms will be applied to the same maze problem with the same agent, but the obstacles will not change during the whole learning process. Fig. 7 illustrates the number of primitive actions taken in each episode using fQ, hQ, hHRL-G, and hHRL-M algorithms. Based on the results, hHRL methods improve the performance considerably and much faster compared to fQ and hQ algorithms, which are although trained in stationary environment. The main reason is that hHRL methods allow for different, appropriate methods for different sub-tasks. More specifically, the MC off-line learning is more efficient in the high-level episodic sub-task with a single final reward. Comparing the results of two hHRL algorithms in Fig. 7, it was found that although the manual decomposition of the environment into a small number of macro states will speed up the learning, the policy trained with hHRL-G without manual decomposition converges slightly faster than the one trained with hHRL-M. In addition, from the



(a) White noise with $\sigma = 0.5$, 239 primitive actions. (b) White noise with $\sigma = 1$, 245 primitive actions.

Fig. 10. The traces of the agent in the presence of measurement noise, where the blue-colored lines and red-colored lines record the measured positions and the actual positions, respectively.

16th episode onwards, the averaged number of primitive actions with hHRL-M and hHRL-G are 694 and 459, respectively.

The results indicate that hHRL-G performs better than hHRL-M, and it is ensured that hHRL-G requires less manual designs, such as the decomposition of the environment or assigning supplemental penalties for approaching obstacles. Thus, hHRL-G is efficient inherently from the hHRL concept and generally applicable with the before-mentioned modifications, such as generalizing training samples with function approximations, using higher-level behavior as the goal indicator, and relabeling the experienced transition according to intrinsic rewards.

4.2. Non-stationary environment over time

In a real-world environment, some non-stationary obstacles may change over a long period, such as desks and chairs, and some others may change from time to time, such as doors, human, and other robots. Therefore, the proposed algorithm will be validated in a non-stationary environment in Fig. 1, where the obstacles may change over time. In other words, some of the obstacles may appear or disappear intermittently within an episode after every a few time steps, but the obstacles will not be placed upon the current agent's state. Fig. 8 presents the agent's trace in the 10th episode, which takes 429 primitive actions.

The above results illustrate that the proposed algorithm is applicable to non-stationary environments changing over episodes and over time without the loss of efficiency. By using this modified hHRL algorithm for guidance and navigation, the agent does not need to memorize the partial map of the environment, manually define the macro state, or estimate whether the obstacle is stationary or not, but only need the information about its relative observation and absolute position.

4.3. Validation in the presence of measurement noise

The real-world agent often observes an accurate relative state from cameras, sonar sensors, infrared systems, etc., and an inaccurate absolute position by using a GPS or an indoor positioning system. Therefore, we will validate the proposed method to the maze with noisy $\boldsymbol{p}$ measurements, and the simulated white noise has a Gaussian distribution $\mathcal{N}(0, \sigma)$. The standard deviations of the noise σ are chosen to be $[0.5, 1, 1.5, 2]$, and the averaged number of primitive actions from the 21st episode onwards are presented in Fig. 9. The result indicates that the performance of the proposed hHRL method remains at the same level in the presence of the simulated white noise with $\sigma \leq 1$. If the amplitude of the noise is further increased, the convergence will take a longer time, and the primitive actions will increase. This is mainly because the high-level decisions and the training adaptations are made based on the noisy absolute positions.

Fig. 10 shows the traces of the agent in the presence of measurement noise with $\sigma = 0.5$ and $\sigma = 1$. The blue-colored lines record the measured positions with noise, and the red-colored lines are the actual trace of the agent in this maze. These results show that the proposed hHRL method is robust to measurement noise. In real applications, the measurement noise in the absolute positions can be reduced with filtering methods. In such a way, the tolerance of the measurement noise can be higher, and the performance can be further improved. This is beyond the scope of this paper and is recommended for future research.

This paper validated the proposed algorithm in Parr's maze navigation problem with non-stationary environment and measurement noise. It is proposed to use unit vector to represent the instructional behaviors, which can be used to calculate the intrinsic reward and relabel the experienced transitions. It can also be easily extended to three-dimensional or continuous problems without changing the method in high-level policy training. For more complex continuous action space problems, online actor–critic methods (Zhou et al., 2020; Zhou, 2022) can be used in the lower level to combine the control of nonlinear, unknown system to this online guidance and navigation method. Since this hHRL method allows different methods in the same level, it is open to being expanded both upwards to tackle more complex, multiple-objective tasks and downwards to control more complex, nonlinear, or continuous systems.

5. Conclusions

This paper proposed a new online guidance and navigation algorithm under the hHRL framework. Compared to our previous algorithm, the improvements of this algorithm include that (1) it does not need manual decomposition of the environment or store maps in the memory, (2) it uses function approximations for generalization in the higher-level, (3) it uses on-policy method for collision avoidance with environment penalties, and (4) it directly uses the high-level instruction as goal indicator and correspondingly define the intrinsic rewards to evaluate the performance of following instructions and to relabel the experienced transitions. The results indicate that the newly proposed algorithm is better in the convergence rate and optimization. In addition, it is more general and efficient especially for non-stationary environments, and it is robust to measurement noise in absolute positions. It does not rely on localization with partial maps or extra sensors recognizing the stationary or non-stationary obstacles, but only a value function approximator.

This paper focuses on the systematic environment decomposition and intrinsic reward definition in online guidance and navigation tasks in non-stationary environments. Although the numerical experiments

are undertaken in discrete 2-dimensional maze environments, it is indicated that the proposed method is generally applicable to both discrete and continuous state spaces and both 2-dimensional and 3-dimensional guidance and navigation environments. Future research should therefore concentrate on the implementation to different applications (1) to validate this algorithm in continuous, non-stationary environments, (2) to apply this method to a real-world ground robot by extending a lower level control policy, and (3) to test this method in 3-dimensional guidance and navigation problems with flying robots.

CRediT authorship contribution statement

Ye Zhou: Conceptualization, Methodology, Validation, Writing – original draft. **Hann Woei Ho:** Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Malaysian Ministry of Higher Education for providing the Fundamental Research Grant Scheme (FRGS), Malaysia (Grant number: FRGS/1/2020/TK0/USM/03/11).

References

Abbaszadeh Shahri, A., Maghsoudi Moud, F., 2021. Landslide susceptibility mapping using hybridized block modular intelligence model. Bull. Eng. Geol. Environ. 80 (1), 267–284.

Barto, A.G., Mahadevan, S., 2003. Recent advances in hierarchical reinforcement learning. Discrete Event Dyn. Syst. 13 (1–2), 41–77.

Bellemare, M.G., Candido, S., Castro, P.S., Gong, J., Machado, M.C., Moitra, S., Ponda, S.S., Wang, Z., 2020. Autonomous navigation of stratospheric balloons using reinforcement learning. Nature 588 (7836), 77–82.

Bellman, R., 1957. Dynamic Programming. Princeton University Press.

Brooks, A., Makarenko, A., Williams, S., Durrant-Whyte, H., 2006. Parametric POMDPs for planning in continuous state spaces. Robot. Auton. Syst. 54 (11), 887–897.

de Oliveira, T.H.F., de Souza Medeiros, L.P., Neto, A.D.D., Melo, J.D., 2021. Q-Managed: A new algorithm for a multiobjective reinforcement learning. Expert Syst. Appl. 168, 114228.

Dietterich, T.G., 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. (JAIR) 13, 227–303.

Eppe, M., Gumbsch, C., Kerzel, M., Nguyen, P.D., Butz, M.V., Wermter, S., 2022. Intelligent problem-solving as integrated hierarchical reinforcement learning. Nat. Mach. Intell. 1–10.

Foka, A., Trahanias, P., 2007. Real-time hierarchical POMDPs for autonomous robot navigation. Robot. Auton. Syst. 55 (7), 561–571.

Fujimoto, S., Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning. PMLR, pp. 1587–1596.

Ghavamzadeh, M., Mahadevan, S., Makar, R., 2006. Hierarchical multi-agent reinforcement learning. Auton. Agents Multi-Agent Syst. 13 (2), 197–229.

He, R., Brunskill, E., Roy, N., 2011. Efficient planning under uncertainty with macro-actions. J. Artificial Intelligence Res. 40 (1), 523–570.

Hengst, B., 2002. Discovering hierarchy in reinforcement learning with HEXQ. In: International Conference on Machine Learning (ICML). 2, pp. 243–250.

Hoey, J., Schröder, T., Alhothali, A., 2016. Affect control processes: Intelligent affective interaction using a partially observable Markov decision process. Artificial Intelligence 230, 134–172.

Khan, S.G., Herrmann, G., Lewis, F.L., Pipe, T., Melhuish, C., 2012. Reinforcement learning and optimal adaptive control: An overview and implementation examples. Annu. Rev. Control 36 (1), 42–59.

Kim, I.Y., De Weck, O., 2006. Adaptive weighted sum method for multiobjective optimization: a new method for Pareto front generation. Struct. Multidiscip. Optim. 31 (2), 105–116.

Kobayashi, T., Sugino, T., 2020. Reinforcement learning for quadrupedal locomotion with design of continual–hierarchical curriculum. Eng. Appl. Artif. Intell. 95, 103869.

Lieck, R., Toussaint, M., 2016. Temporally extended features in model-based reinforcement learning with partial observability. Neurocomputing 192, 49–60.

Lin, J.G., 2005. On min-norm and min-max methods of multi-objective optimization. Math. Program. 103 (1), 1–33.

Liu, C., Xu, X., Hu, D., 2015. Multiobjective reinforcement learning: A comprehensive overview. IEEE Trans. Syst. Man Cybern.: Syst. 45 (3), 385–398.

Ma, A., Ouimet, M., Cortés, J., 2020. Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning. Auton. Robots 44 (3), 485–503.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Nachum, O., Gu, S.S., Lee, H., Levine, S., 2018. Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3303–3313.

Ni, Z., He, H., Wen, J., Xu, X., 2013. Goal representation heuristic dynamic programming on maze navigation. IEEE Trans. Neural Netw. Learn. Syst. 24 (12), 2038–2050.

Parr, R.E., Russell, S., 1998a. Hierarchical Control and Learning for Markov Decision Processes. University of California, Berkeley Berkeley, CA.

Parr, R., Russell, S., 1998b. Reinforcement learning with hierarchies of machines. Adv. Neural Inf. Process. Syst. 1043–1049.

Ragi, S., Chong, E.K.P., 2013. UAV path planning in a dynamic environment via partially observable Markov decision process. IEEE Trans. Aerosp. Electron. Syst. 49 (4), 2397–2412.

Scott A. Miller, Z.A.H., Chong, E.K.P., 2009. A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. EURASIP J. Adv. Signal Process..

Si, J., 2004. Handbook of Learning and Approximate Dynamic Programming, Vol. 2. John Wiley & Sons.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587), 484–489.

Sridharan, M., Wyatt, J., Dearden, R., 2010. Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. Artificial Intelligence 174 (11), 704–725.

Sutton, R.S., Barto, A.G., 1998. Introduction to Reinforcement Learning. MIT Press.

Sutton, R.S., Precup, D., Singh, S., 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112 (1–2), 181–211.

Takamuku, S., Arkin, R.C., 2007. Multi-method learning and assimilation. Robot. Auton. Syst. 55 (8), 618–627.

Tan, A.-H., Ong, Y.-S., Tapanuj, A., 2011. A hybrid agent architecture integrating desire, intention and reinforcement learning. Expert Syst. Appl. 38 (7), 8477–8487.

Theile, M., Bayerlein, H., Nai, R., Gesbert, D., Caccamo, M., 2020. UAV Path planning using global and local map information with deep reinforcement learning. arXiv preprint arXiv:2010.06917.

Theocharous, G., Mahadevan, S., 2002. Approximate planning with hierarchical partially observable Markov decision process models for robot navigation. In: IEEE International Conference on Robotics and Automation, ICRA'02, Vol. 2. IEEE, pp. 1347–1352.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E., 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach. Learn. 84 (1), 51–80.

Van Moffaert, K., Nowé, A., 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. J. Mach. Learn. Res. 15 (1), 3483–3512.

Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K., 2017. Feudal networks for hierarchical reinforcement learning. In: International Conference on Machine Learning. PMLR, pp. 3540–3549.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1 (1), 67–82.

Zhou, Y., 2022. Efficient online globalized dual heuristic programming with an associated dual network. IEEE Trans. Neural Netw. Learn. Syst..

Zhou, Y., van Kampen, E., Chu, Q.P., 2016. Autonomous navigation in partially observable environments using hierarchical Q-learning. In: Proceedings of the International Micro Air Vehicles Conference and Competition 2016, Beijing, China.

Zhou, Y., van Kampen, E.-J., Chu, Q., 2018. Incremental approximate dynamic programming for nonlinear adaptive tracking control with partial observability. J. Guid. Control Dyn. 41 (12), 2554–2567.

Zhou, Y., van Kampen, E.-J., Chu, Q., 2019. Hybrid hierarchical reinforcement learning for online guidance and navigation with partial observability. Neurocomputing 331, 443–457.

Zhou, Y., van Kampen, E.-J., Chu, Q., 2020. Incremental model based online heuristic dynamic programming for nonlinear adaptive tracking control with partial observability. Aerosp. Sci. Technol. 105, 106013.