Learning to Navigate Through Complex Dynamic Environment With Modular Deep Reinforcement Learning

Yuanda Wang, Haibo He, *Fellow, IEEE*, and Changyin Sun

***Abstract*—In this paper, we propose an end-to-end modular reinforcement learning architecture for a navigation task in complex dynamic environments with rapidly moving obstacles. In this architecture, the main task is divided into two subtasks: local obstacle avoidance and global navigation. For obstacle avoidance, we develop a two-stream Q-network, which processes spatial and temporal information separately and generates action values. The global navigation subtask is resolved by a conventional Q-network framework. An online learning network and an action scheduler are introduced to first combine two pretrained policies, and then continue exploring and optimizing until a stable policy is obtained. The two-stream Q-network obtains better performance than the conventional deep Q-learning approach in the obstacle avoidance subtask. Experiments on the main task demonstrate that the proposed architecture can efficiently avoid moving obstacles and complete the navigation task at a high success rate. The modular architecture enables parallel training and also demonstrates good generalization capability in different environments.**

***Index Terms*—Deep learning, navigation, reinforcement learning, two-stream Q-network.**

I. Introduction

RECENT advances in reinforcement learning have shown notable capability in many complex tasks, such as playing video games [1]–[8], controlling high-dimensional robots [9], [10], and even defeating a world-class player in the game of Go [11]. Unlike many of the previous successful reinforcement learning applications using low-dimensional hand-crafted feature representations [12], [13], scientists and engineers recently demonstrated that it is possible for a reinforcement learning agent to directly learn policies from a high-dimensional raw sensory input and work in an end-to-end fashion. This great success partially credits to the recent breakthroughs in deep learning [14], where deep neural networks are built to extract rich and effective representations from raw data for different learning tasks. For example, a newly developed deep Q-learning (DQL) algorithm achieved human-level control in the Atari game challenge [1], which employed a deep convolutional neural network (CNN) to process raw pixel input and approximate the optimal action-value function.

Among a great variety of tasks that can be formalized as reinforcement learning problems, learning to navigate in a complex dynamic environment is one of the most attractive tasks. It is also considered as an important milestone in developing artificial intelligence. A huge amount of real-life applications can be formalized as navigation problems such as self-driving cars and domestic robots.

In this paper, we study the navigation problem in the dynamic environment with rapidly moving obstacles with deep reinforcement learning (DRL). The goal is to find a control policy for an agent to navigate through a simulated indoor environment and reach the destination without colliding with walls or rapidly moving obstacles. We propose a modular DRL architecture, which takes raw range sensor data and relative position data as input and generates discrete control commands. Our architecture has three main modules: avoidance module, navigation module, and action scheduler. The avoidance module and the offline part of the navigation module are pretrained and give action alternatives that relate to avoiding collisions and to approaching the destination, respectively. The online part of the navigation module performs an online off-policy learning and also offers an action alternative. The action scheduler then finalizes the action to be executed based on the current situation and the learning progress.

The modular design enables the modules related to different skills to be trained and tested independently. The trained models can be easily generalized and reused in different environments. For example, obstacle avoidance is one of the basic skills in robotics. Once an avoidance module is trained, we can directly plug it in different agents in different environments, as long as the sensations and actions are compatible. With the help of online learning and action scheduling, the architecture can quickly adapt to new environments and complete tasks.

Manuscript received May 29, 2017; revised November 14, 2017, March 5, 2018, and May 27, 2018; accepted June 1, 2018. Date of publication June 25, 2018; date of current version December 13, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61520106009 and Grant U1713209 and in part by the National Science Foundation under Grant CMMI 1526835. *(Corresponding author: Haibo He.)*

Y. Wang and C. Sun are with the School of Automation, Southeast University, Nanjing 210096, China, and also with the Key Laboratory of Measurement and Control of Complex System of Engineering, Ministry of Education, Southeast University, Nanjing 210096, China (e-mail: yuanda_wang@uri.edu; cysun@seu.edu.cn).

H. He is with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI 02881 USA (e-mail: haibohe@uri.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TG.2018.2849942

Experiments in four different environments show that our architecture is more flexible and efficient than an integrated model from the conventional DRL algorithm.

The main contributions of this paper are summarized as follows.

1) A modular architecture for navigation problems is proposed to separately resolve local obstacle avoidance and global navigation, which enables modularized training and promotes generalization ability.
2) A novel two-stream Q-network is developed for obstacle avoidance. By separating spatial and temporal information from raw input and processing them individually, this new approach supplements temporal integration of agent observations and surpasses the conventional DQL approach in moving obstacle avoidance tasks.
3) We proposed an action scheduling method, which combines and exploits the pretrained policies for efficient exploration and online learning in unknown environments.

The rest of this paper is organized as follows. In Section II, we review some related studies on navigation problems. The proposed architecture and implementation details are presented in Section III. In Section IV, we describe the training, evaluation, and testing details; analyses and discussions about the results are also given. Finally, we state conclusions and future works in Section V.

II. Related Work

A. Traditional Robotics

The navigation topic has been extensively studied over decades in the traditional robotics field. Numerous algorithms have been developed for obstacle avoidance and path planning for mobile robots in different situations. The Potential Field Method (PFM) [15] models the environment by building an artificial potential field. The robot is repulsed away from the obstacles and attracted to the goal position. Many variants are developed for different kinds of environments and sensors [16]–[18]. However, the PFM may get trapped in local minima or oscillate in narrow spaces, which are the major weaknesses. To solve this problem, another method named vector field histogram (VFH) [19] is developed. It gives the desired moving directions of the robot by looking for gaps in the locally established polar histogram. It also has many variants, such as VFH+ [20] and VFH* [21]. Simultaneous localization and mapping (SLAM) [22] is another famous approach for robot navigation. It deals with the navigation problem by continuously constructing and updating a map of the environment while localizing itself using the same map. This approach is first implemented on range-sensor-based agents using extended Kalman filter [23] and particle filter [24]. Later, many vision-based SLAM approaches have been developed, either with a monocular camera [25] or a stereo camera [26]. Although SLAM has been successful in many commercial applications, it still suffers from a large cost of memory and computation. Besides, most existing SLAM algorithms are unable to deal with dynamic environments. Almost all traditional robotics approaches require accurate models of the robots and the sensors to implement the algorithm, which could be another difficulty in the application.

B. Deep Learning

Recently, it is popular to study the navigation problems with supervised learning and deep learning. Tai *et al.* presented a solution to obstacle avoidance by using a deep CNN [27]. The network takes raw depth image as the input and generates control commands for a mobile robot. The training set is established by recording human operations. Pfeiffer *et al.* developed an end-to-end navigation model that drives a robot to approach a target [28]. The model consists of a deep CNN, which uses 2-D laser scan data and relative target positions to determine steering commands. The model is trained on expert demonstrations generated in simulations. Compared with the conventional robotics methods that rely on accurate models and hand-crafted features, the deep learning based navigation approaches can directly take the raw sensory input, which is more concise and flexible. However, the performances of these supervised learning methods are highly related to the training sets, which are mostly collected from human demonstrations. These data collections could be very time-consuming and also have other defects like overfitting.

C. Reinforcement Learning

Instead of learning from training samples, the reinforcement learning can directly learn from interacting with the environment. Over the past few decades, many techniques and architectures have been developed to improve the efficiency and performance of reinforcement learning in various tasks including navigation.

Seeking a good policy over a very large state space is always a challenging problem for reinforcement learning. An example is to find the optimal navigation policy in a large-scale unknown environment. Although we can use function approximators to generalize limited experiences across the whole state space, the exploration with sparse feedback is still a big challenge. One of the important solutions is hierarchical reinforcement learning [29], which addresses large-scale planning problems by introducing multilevel spatio–temporal abstractions. The *option* framework [30] introduces the concept of *option*, which is a partial policy for a subgoal of the main goal. Achieving the subgoals can facilitate achieving the main goal. At each step, the agent either selects a single step action or an *option* to achieve a certain subgoal. Rohanimanesh and Mahadevan [31] proposed a concurrent option framework for the agent that is able to execute options in parallel. Besides, there are many other hierarchical reinforcement learning frameworks such as HAM [32], MAXQ [33], and others.

Recently, researchers use deep neural networks as function approximators in reinforcement learning to process raw sensory inputs and develop end-to-end learning approaches known as DRL. Many works have been done on this topic: Double Q-learning [34] algorithm reduces overestimation in action value approximation by decomposing action selection and action evaluation. Dueling network architecture [35] explicitly separates the estimation of state values and action advantages to get more accurate policy evaluation. Both methods significantly boost the performance in several Atari games. The deep deterministic policy gradient method [9] extends the DQL into continuous action domain. By using an actor-critic structure, this method successfully resolves many simulated physical tasks with continuous control inputs. The asynchronous advantage actor-critic (A3C) method [36] enables asynchronous training of multiple agents in separated parallel threads, which significantly improves the training efficiency. Hierarchical-DQN [37] integrates hierarchical action-value functions, which operate at different temporal scales, with goal-driven intrinsically motivated DRL to improve the efficiency of exploration in complicated environments. The recurrent neural network (RNN) is also introduced to deal with partially observable environments [38].

In the navigation domain, DRL has shown great effectiveness. Tai and Liu [39] trained a deep Q-network-based agent for obstacle avoidance in a simulated indoor environment. The agent takes raw RGB-D sensor input and generates discrete control commands. This method demonstrates the capability of navigating a mobile robot in a real-world environment. They also extended this paper to a range-sensor-based robot with continuous control using asynchronous deep deterministic policy gradient [40]. The navigation performance of networks with external memory is investigated in the Minecraft 3-D block world environment [3]. Tessler *et al.* [41] developed a hierarchical DRL system that has the ability to transfer knowledge from previously learned skills. The proposed system exhibits good navigation performance also in Minecraft environment. RNNs and their long short-term memory (LSTM) version are very popular in first-person view navigation problems due to its ability to capture long-term dependencies in the sequential information from history. Lample and Chaplot [42] developed an agent with deep recurrent Q-networks and outperformed the built-in AI agents in the ViZDoom FPS game using only the raw pixel input. Remarkably, Mirowski *et al.* [43] proposed a visual navigation approach in complex 3-D maze environments with A3C method and auxiliary tasks, which significantly improved the performance. The A3C agent processes raw RGB inputs with convolutional encoders and LSTM layers. The agent is also simultaneously trained on two auxiliary tasks, which are depth prediction and loop closure classification.

III. Proposed Approaches

To find the optimal policy to navigate through a complex dynamic environment, we take the divide and conquer strategy and divide the main navigation task into two subtasks: local avoidance and global navigation. The avoidance task is to keep the agent away from walls and moving obstacles, whereas the navigation task is to drive the agent toward the destination. In our proposed approach, two subtasks are resolved by the avoidance module and the navigation module, respectively. The action scheduler is introduced to finalize the action to execute. The approach is similar to the *option* framework in hierarchical reinforcement learning. However, the skills learned from the subtasks are used to help the explorations at the early stage; the final policy directly selects actions to achieve the main goal instead of choosing from the *options*. A diagram is given in Fig. 1 to illustrate the modular DRL approach for navigation. In the rest of this section, an introduction to the simulated environment is given first. Then, we thoroughly describe the three main

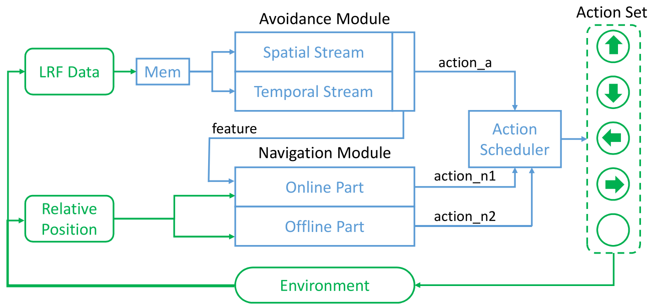


Fig. 1. Basic structure of the modular DRL approach for navigation. The components inside the reinforcement learning agent are marked in blue; the components related to external environment are marked in green.

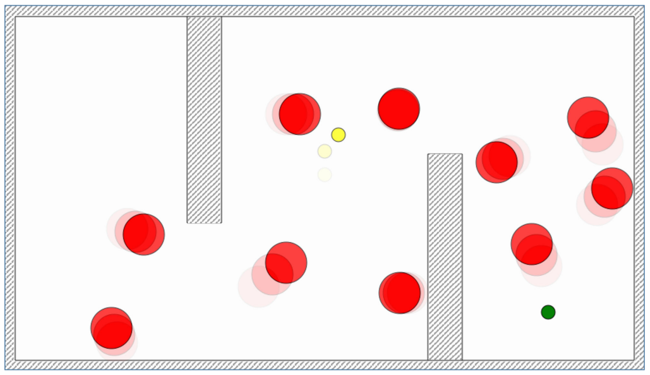


Fig. 2. Snapshot of an ongoing simulation in the simulated environment. The red circles represent moving obstacles and the yellow dot represents the agent. Their positions of several previous time steps are shown in faded shades to demonstrate the motions. The destination is shown by a green dot on the bottom right corner. The walls are depicted by dashed areas.

components and their cooperative working procedure according to Fig. 1.

A. *Simulated Environment*

To demonstrate and evaluate our proposed approach, and to compare with other navigation approaches, a simulated 2-D environment is developed. Fig. 2 is a snapshot of an ongoing navigation simulation in the simulated environment.

The agent senses its surroundings through a simulated laser range finder (LRF) with a field of view (FOV) of 360°. The distance measurements are made uniformly around the agent and the angular resolution is 1.8°. The output of the simulated LRF is a 200-D vector o_t, which is normalized by the maximum effective range $o_{\max}$. The first element of o_t always indicates the measurement in horizontal right direction, and is followed by the other measurements in counterclockwise direction. Additionally, the agent can also obtain a 2-D vector p_t, which indicates the relative coordinates of the destination. The dynamic of the moving agent is simulated by a second-order physical system defined by the following parameters: force F, mass m, and friction factor μ. The initial speed of the agent is zero, and the max speed is $v_{\max}$. The speed can be changed by taking actions. The action set consists of five commands, $\mathcal{A} = \{up, down, left, right, none\}$, which represents applying

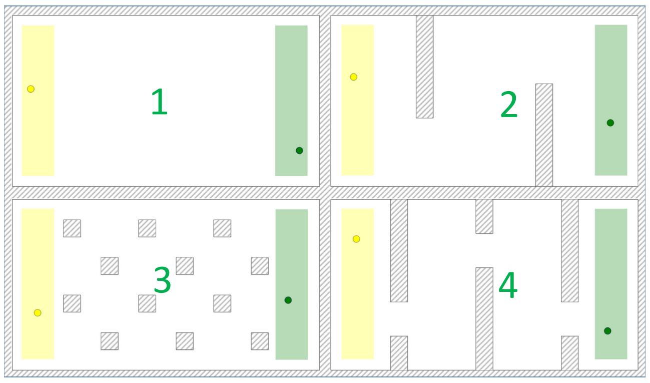


Fig. 3. Layouts of four testing rooms of the simulated environment. The start points and destination points are randomly initialized within the yellow and green shaded areas in each room, respectively. The walls are depicted by dashed areas. The moving obstacles are not shown in this figure.

TABLE I
PARAMETERS OF THE SIMULATED ENVIRONMENT

Parameter	Value
size	$9m \times 5m$
sampling time	$0.1s$
radius of agent	$0.05m$
radius of obstacles	$0.30m$
force F	$1N$
mass of agent m	$1kg$
friction factor μ	0.02
maximum sensing range o_{max}	$2m$
maximum speed v_{max}	$0.5m/s$
maximum steps in one episode T_{max}	$1,000$

forces to the agent in four directions, and the *none* means no force applied.

The obstacles always move straight at constant speed. The initial positions and speeds are randomly set at the beginning. When an obstacle collides with the wall, its moving direction changes according to physical laws but the speed stays constant. The moving obstacles can overlap with each other and the collisions between obstacles are not considered.

To test the proposed approach in a comprehensive manner, we have designed four different testing rooms and their layouts are shown in Fig. 3. *Room-1* is a simple blank layout. *Room-3* is filled with small square static obstacles. *Room-2* and *Room-4* both have long walls that block the way from the start point to the destination. The moving agent has to navigate through the gaps on the walls, which is very challenging, especially in *Room-4* where the gaps are very narrow.

The environment works in an episodic style. After an episode is terminated, all moving objects are reset. The three events that terminate the current episode are as follows:

1) agent collides with walls or moving obstacles;
2) simulation time step exceed $T_{\max}$;
3) agent reaches the destination.

The detailed parameters of the environment are listed in Table I. We assume that the maps of the rooms, the parameters of the environment, the dynamics of the moving agent, and the real-time motions of the moving obstacles are not known by any controller interacting with this environment, in other words, only the LRF data and the relative coordinates are available. All following learning, testing, and comparison processes are performed in this simulated environment.

B. Avoidance Module

Obstacle avoidance with LRF observation is a classical robotics problem. Many traditional methods such as the artificial potential field [15] method and the VFH method [19] use complex manually designed algorithms to solve this problem. The implementations require dynamic models of the robots and sensors. However, the dynamics are unknown in the simulated environment, and the moving obstacles make it even more challenging. In the avoidance module, we address the avoidance subtask with DRL and two-stream Q-network.

In reinforcement learning, the simulated environment is formalized as a Markov Decision Process (MDP) described by a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. At each time step t, an agent in state $s_t \in \mathcal{S}$, chooses and takes the action $a_t \in \mathcal{A}$ according to the policy π, then receives the reward $r_t \sim \mathcal{R}(s_t, a_t)$, and finally ends up in a new state s_{t+1} following the deterministic transition function $s_{t+1} = \mathcal{P}(s_t, a_t)$. By setting negative rewards for collision, we express the goal of obstacle avoidance in the form of the general goal of reinforcement learning, which is seeking a policy to maximize the total rewards. Q-learning algorithm [44] is one of most important breakthroughs in reinforcement learning for estimating the future discounted total reward, also known as return. The return is formalized as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $\gamma \in [0, 1]$ is the discount factor and T is the time step when the MDP comes to an end. The function that maps state-action pairs to their estimated expected return following the policy π is named action-value function or Q-function, which is defined as $Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a]$. We define the optimal Q-function $Q^{\star}(s, a)$ as the maximum return expected to achieve by following any policy, at state s then executing action a, that is, $Q^{\star}(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a]$. This optimal action-value function verifies the *Bellman optimality equation*: $Q^{\star}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^{\star}(s', a') | s, a]$. Then, the Q-function can be estimated by utilizing this equation as an iterative update, which is given by

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)].$$

Because the state space is continuous in the simulated environment, we need a function approximator to estimate the Q-function and generalize limited training experiences across the whole state-action space. The function approximator can be a simple linear function for low-dimensional observations or a complex nonlinear function approximator such as deep CNNs for high-dimensional images. For this avoidance task with LRF observations in a dynamic environment, we proposed a two-stream Q-network to approximate the action-value function.

The LRF data reflect the relative distances to obstacles within the sensing range, which convey the local spatial information. Besides, the relative motion information of obstacles or local temporal information is also critical for the agent to make correct decisions. To take both spatial and temporal information into consideration, we adopt the two-stream architecture. The

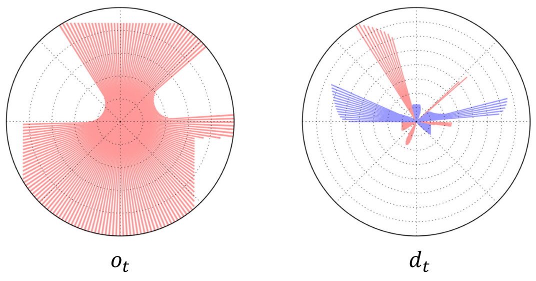


Fig. 4. Frame of raw laser scan (o_t) and its difference with the previous laser frame ($d_t = o_t - o_{t-1}$). Positive values are marked in red and negative values in blue.

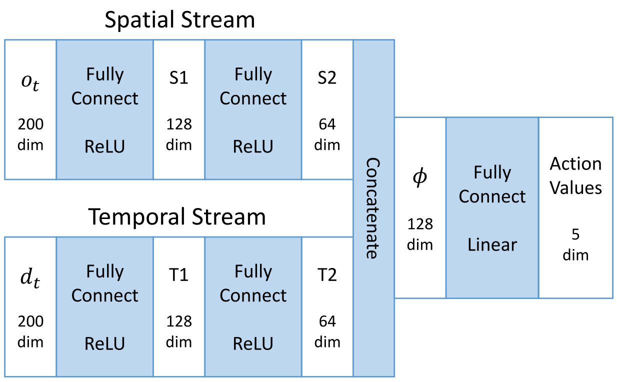


Fig. 5. Two-stream Q-network structure.

two-stream architecture is related to the two-stream hypothesis [45]. According to this biological hypothesis, the human visual cortex contains two pathways: one for object recognition and the other for motion recognition. This structure was first introduced for action recognition in videos, and achieved the state-of-the-art performance [46]. Inspired by this, we split the input into two parts. The spatial part is the raw laser scan o_t; the temporal part is the difference between the present and the previous laser scans, namely, $d_t = o_t - o_{t-1}$, which reflects the motion across successive observations. An example of the raw spatial data o_t and temporal data d_t is given in Fig. 4.

The observations in Fig. 4 were taken at the same moment as shown in Fig. 2 from the moving agent's view. From the raw spatial measurements o_t, we can only tell that there were two obstacles in the up-left direction and the up-right direction, respectively. However, after studying the temporal part d_t, we can find out the agent was relatively approaching the two obstacles. Because the two parts depict the surroundings from different aspects, the features are apparently different. Therefore, it is reasonable to process the two parts separately to decouple the feature extraction of spatial and temporal data. With both spatial and temporal information as input, especially the temporal part which explicitly conveys the relative movements, the agent is able to get a more comprehensive knowledge of the surrounding situations.

To extract useful features from the raw measurements, one may consider CNNs since they have been very popular in DRL to deal with raw inputs. Pfeiffer *et al.* [28] employed a network with five 1-D convolutional layers and three fully connected layers to process 1080-D vectors from an LRF in a real-time environment. However, for the relatively low-dimensional inputs, fully connected neural networks are more concise and efficient to implement. Tai *et al.* [40] employed a simple three-hidden-layer fully connected neural network to handle 10-D LRF input vectors. From a set of comparative experiments and analyses, we found that the fully connected network is a better structure for this avoidance subtask. The details are given in Section IV-A.

The two-stream Q-network structure is shown in Fig. 5. The spatial part o_t and the temporal part d_t are fed into the network in two streams separately. We use two fully connected layers to extract spatial and temporal features. The first layer outputs a 128-D vector and the second layer generates a 64-D vector. Then, the two vectors are concatenated as one feature vector ϕ, which is a direct combination of both spatial and temporal features. Each layer above is followed by a rectifier linear unit (ReLU). Finally, another fully connected linear layer projects the feature ϕ to the action values. The two-stream Q-network is present as $Q^\theta(s_t, a_t)$, where $s_t = (o_t, d_t)$ and θ represents the network weights.

The training procedure of the two-stream Q-network mainly follows the DQL algorithm in [34], and two main techniques are employed. The first one is *experience replay* [47]. The agent's transition experience $e = (s_t, a_t, r_t, s_{t+1})$ of each step is stored in a replay buffer $\mathcal{D} = (e_1, \ldots, e_N)$. The network weights are updated using a minibatch of experiences uniformly sampled from the replay buffer. In this way, the correlations between consecutive samples are significantly reduced, which leads to a smooth and stable learning process, and also improves the data efficiency. The second technique is *double Q-learning*. A separated network termed target network is introduced to estimate the target for gradient descent update of the main network. This method breaks the feedback resulting from updating toward the target generated by its own, which further eliminates the potential oscillation and divergence of network weights.

The weights of the main network at training iteration i are updated by minimizing the loss function

$$L_i(\theta_i) = \mathbb{E}_{e\sim\mathcal{D}}\left[(y_i - Q^{\theta_i}(s_t, a_t))^2\right] \tag{1}$$

where

$$y_i = r + \gamma \hat{Q}^{\theta^-}(s_{t+1}, \arg\max_{a'} Q^{\theta_i}(s_{t+1}, a')) \tag{2}$$

is the updating target from the separated target network $\hat{Q}$ with parameters θ^-. The update is performed by gradient descent

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{e\sim\mathcal{D}}\left[(y_i - Q^{\theta_i}(s_t, a_t))\nabla_{\theta_i} Q^{\theta_i}(s_t, a_t)\right] \tag{3}$$

$$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_{\theta_i} L_i(\theta_i) \tag{4}$$

where α is the learning rate. Instead of computing the accurate expectation over the whole replay buffer, we computationally take expectations over the sampled batch of experiences and use a minibatch gradient descent. The target network slowly synchronizes its weights to the main network by a small update factor τ in every iteration.

To ensure continual exploration, the action is selected by an ϵ-greedy strategy: with a probability ϵ to choose an action at random and with a probability of $1 - \epsilon$ to choose the action with the best action value. After adequate training, the learned Q-function approximated by the two-stream Q-network will finally converge near the optimal Q-function. Then, the learned avoidance control policy is simply acting greedily to the learned action-value function, that is $\pi^\theta(s) = \arg\max_a Q^\theta(s,a)$. The agent equipped with this pretrained avoidance module will then be competent to avoid moving obstacles.

C. Navigation Module

The navigation module has two parts: the offline part and the online part. The offline part is designed to solve the simple navigation subtask, which is seeking the fastest policy to drive the agent directly to the destination only based on the relative coordinates in a blank environment without any moving obstacles. Although there are many efficient path-finding or control approaches off the shelf for similar navigation problems, they cannot be directly employed for this subtask. The simulated environment has a continuous state space instead of a graph-based model, so it cannot be solved by graph searching algorithms like A* [48]. Furthermore, according to our assumption on the simulated environment, the system dynamics of the moving agent are not available, therefore feedback control methods like optimal control [49] are not applicable for their requirement of an accurate system model. Finally, we also use the Q-learning algorithm to solve this subtask. Comparing with the DQL algorithm for obstacle avoidance, this navigation task is much easier, so we use the standard Q-learning algorithm [50] with a single-hidden-layer network as the function approximator. The input is the 2-D vector p_t. To describe the object policy, we set a large positive reward for reaching the destination and a small negative reward for every time step to urge the agent to get the destination in the fastest way. After training, the offline part of navigation module is able to give the best moving action toward the destination in a blank environment.

The online part of the navigation module provides the ultimate policy for solving the main task by online learning, which is also performed following the DQL algorithm. As shown in Fig. 1, the online part has two inputs: 1) a 2-D relative coordinates vector of the destination; and 2) a 128-D feature ϕ from the avoidance module, which conveys both spatial and temporal information of the surrounding situation. With these inputs, the online part of the navigation module is able to obtain both local obstacle information and global navigation information. The rewards from the testing environment, which indicate collision, reaching, and time-consuming, give an immediate feedback to improve the policy.

We use another fully connected neural network to approximate the action-value function for online learning. The feature ϕ and relative coordinates are concatenated as a 130-D vector as the input of the online learning network. We set a positive reward for reaching the destination and a negative reward for collision and also a small time punishment in each step. The training procedure follows the DQL algorithm, which is the same as that used for training the two-stream Q-network. We also employed experience replay and double Q-learning for the online training. However, the action selection is different: instead of applying ϵ-greedy to the action-value function, we designed an action scheduler to pick up the actions from three action alternatives.

D. Action Scheduler

The action scheduler is designed to perform efficient exploration and speed up the training for the online part of the navigation module. The action scheduler first exploits the learned policies from the avoidance module and the offline part of the navigation module to accumulate sufficient high-quality experiences in the replay buffer for training. Over time, the scheduler gradually transfers from the static learned policy to the online learning policy. The online exploring and optimizing continues until a stable policy is obtained.

Before interacting with a new environment, the policy from the online part is random, because the online network is not pretrained. Learning from the random policy is a typical approach in classical reinforcement learning. Because the local obstacle and global destination information are all available to the online part, a good policy will eventually be found after sufficient training iterations. However, in a fairly complex room layout like *Room-4*, the agent has great difficulty to find the destination and get positive rewards, because almost all experiences are about collisions. Therefore, the training progress is very slow, meanwhile, the performance in the early stage of testing is unsatisfactory.

Fortunately, we have two pretrained modules, from which the action alternatives are corresponding to the best actions to avoid obstacles and to directly approach the destination. Although these two standalone policies are not able to solve the main task, we develop a simple switching method based on the real-time collision risk to combine two policies into a heuristic policy. Among all obstacles perceptible to the agent, the nearest and the fastest approaching one is the biggest threat; therefore, we combine the relative position and relative speed to evaluate the collision risk. The real-time risk factor η is defined as

$$\eta = \frac{1}{\min(o_t - \kappa d_t)}$$

where $\kappa \in [0,1]$ is a parameter that indicates the sensitivity to relative speed. Obviously, larger η indicates the more risk of collision. Based on the risk factor, we define a switching threshold λ, and the switching strategy is given as

$$a = \begin{cases} \text{action\_n2} & \eta < \lambda \\ \text{action\_a} & \eta \geq \lambda \end{cases}$$

where action_a and action_n2 are the action alternatives offered by the avoidance module and the offline part of the navigation module, respectively. If the risk factor exceeds the switching threshold, the agent takes the avoidance action, otherwise takes the navigation action. Apparently, the performance of this simple switching policy highly depends on the switching threshold, which is difficult to determine without environment dynamics. If we are fortunate to have a good threshold in one room, it may not perform well in another room, because it is a static policy without any adaptive or learning ability. However, it is enough to perform as a heuristic policy to accumulate high-quality experiences including both collisions and successes for the early stage training of the online part of the navigation module.

To guarantee a smooth transfer, we introduce a self-decaying transfer probability φ. In each step, the agent follows the heuris-

Algorithm 1: Online Training Workflow of Our Proposed Architecture.

```
1:  Initialize:
      Initialize online part of navigation module $Q^{n1}$
      Load offline part of navigation module $Q^{n2}$
      Load avoidance module $Q^a$
      Load simulated environment
      Set switching threshold $\lambda$
      Set transfer probability $\varphi = 1$
2:  for episode=1 to MaxEpisode do
3:    Set/reset simulated environment
4:    Observe $o_1$ and $p_1$, set $d_1 = [o_1, \mathbf{0}]$
5:    Get feature $\phi_1$ from avoidance module
6:    for timestep $t = 1$ to $MaxStep$ do
7:      if $rand() < \varphi$ then
8:        Evaluate risk factor $\eta$
9:        if $\eta < \lambda$ then
10:         Select action by offline part
            $a = \arg\max_a Q^{n2}(p_t, a)$
11:       else
12:         Select action by avoidance module
            $a = \arg\max_a Q^a([o_t, d_t], a)$
13:       end if
14:     else
15:       if $rand() < \epsilon$ {Generate a new random
          number} then
16:         $a =$ random action
17:       else
18:         $a = \arg\max_a Q^{n1}([\phi_t, p_t], a)$
19:       end if
20:     end if
21:     Execute action $a$
22:     Observe $o_{t+1}, d_{t+1}, p_{t+1}$ and reward $r$
23:     Get feature $\phi_{t+1}$ from avoidance module
24:     Save experience $e_t = ([\phi_t, p_t], a, r, [\phi_{t+1}, p_{t+1}])$
25:     Train $Q^{n1}$ using mini-batch stochastic gradient
26:     Decay $\varphi$
27:   end for
28: end for
```

tic policy with the probability φ for heuristic exploration, with probability $(1-\varphi)(1-\epsilon)$ to exploit its online learning policy. Besides, we still keep a small probability ϵ for random exploration to further promote the performance. The φ starts from 1 and decays in each step utill 0. At the early stage, the executed actions are mostly from the heuristic policy, and the online part learns from these experiences and quickly develops the capability for avoidance and approaching the destination. At the late stage, the online part takes over and implements its own policy and continues optimizing the performance. The workflow of the proposed approach for online learning is shown in Algorithm 1.

IV. Experiments and Results

A. Pretrain and Evaluation

The pretrain of the avoidance module is performed in the blank environment *Room-1* with 20 moving obstacles. We disable the destination and enlarge the start area to the whole room for training. Before training, the two-stream Q-network is initialized with random weights as the main network, and then a duplicate network is made as the target network. In the beginning of every episode, the positions of 20 moving obstacles and the agent are randomly initialized. The agent is guaranteed to be collision-free in the first step. The speeds of obstacles are also randomly assigned. In each episode, once the agent collides with the walls or other moving obstacles, the current episode terminates, then a negative reward r_{coli} is given. However, the reward is extremely delayed because it is given only when the collision occurs, which always leads to slow learning progress. Therefore, we use the reward shaping method suggested in [51] and give a small intermediate positive reward r_{step} for each collision-free step, which effectively speeds up the learning process. The rewards r_{coli} and r_{step} are listed in Table II.

TABLE II
Training Parameters and Reward Settings of DQL

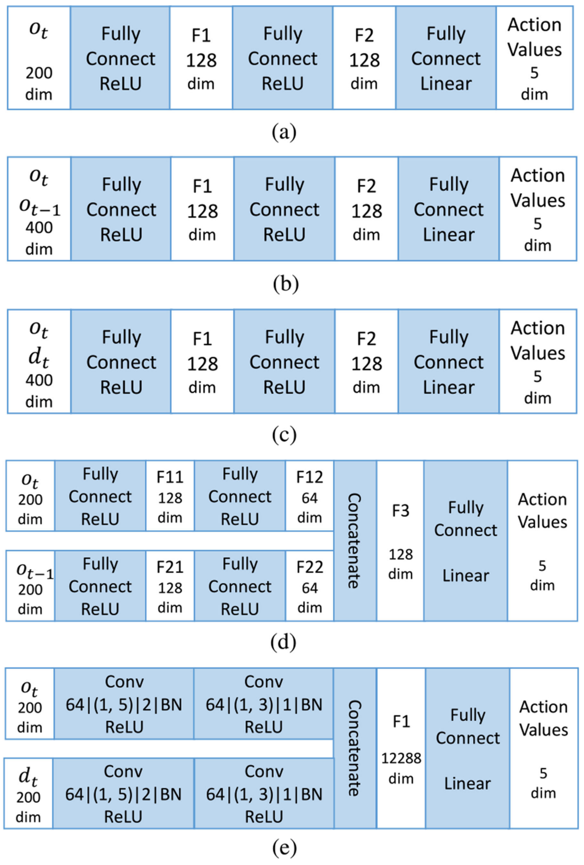
Parameter	Value	Reward	Value
Discount factor γ	0.99	Collision r_{coli}	-100
Initial learning rate α	0.0001	Collision-free r_{step}	1
Soft update factor τ	0.001	Time punishment r_{time}	-0.1
Replay buffer size N	$50,000$	Reach destination r_{dest}	100
Batch size	32		

For the first 1000 steps, we set $\epsilon = 1$ for random exploration. Then, in the next 10 000 steps, ϵ linearly reduces to 0.1. The experience e of every time step is stored in the replay buffer $\mathcal{D}$. The state is expressed as $s_t = (o_t, d_t)$ except $s_1 = (o_1, \mathbf{0})$. The actions are selected following the ϵ-greedy strategy. After the first 1000 steps, we start training the network in every step. In training iteration i, a batch of experience is uniformly sampled from the replay buffer $\mathcal{D}$ and performs minibatch gradient descent following (3) and (4). The training parameters are given in Table II.

Our network models are built with Tensorflow [52] and implemented on a single Nvidia TITAN Xp GPU. The simulated environment runs on an Intel i7-6800K CPU. The networks are trained using the Adam optimizer [53]. All the following trainings and tests are performed on this platform.

To highlight the performance of our proposed two-stream Q-network, we introduced five Q-networks with different inputs and structures from related works for comparison. Their structures are shown in Fig. 6. A brief introduction of the five networks is given as follows.

1) *One-frame Q-network:* Only one frame of the laser observation o_t is fed into the network. It goes through two fully connected layers with ReLU activation functions and then maps to action values by a linear fully connected layer. This is a very common network structure for value function approximation for low-dimensional inputs. Obviously, this network only considers spatial information. The structure is shown in Fig. 6(a).
2) *Two-frame-1 Q-network:* Two consecutive frames of observations, o_t and o_{t-1}, are connected into one vector as the input of this network. The following processes are the same as the *one-frame Q-network*. This structure is similar



o_t 200 dim | Fully Connect ReLU | F1 128 dim | Fully Connect ReLU | F2 128 dim | Fully Connect Linear | Action Values 5 dim
(a)
o_t o_{t-1} 400 dim | Fully Connect ReLU | F1 128 dim | Fully Connect ReLU | F2 128 dim | Fully Connect Linear | Action Values 5 dim
(b)
o_t d_t 400 dim | Fully Connect ReLU | F1 128 dim | Fully Connect ReLU | F2 128 dim | Fully Connect Linear | Action Values 5 dim
(c)
o_t 200 dim | Fully Connect ReLU | F11 128 dim | Fully Connect ReLU | F12 64 dim
o_{t-1} 200 dim | Fully Connect ReLU | F21 128 dim | Fully Connect ReLU | F22 64 dim
Concatenate | F3 128 dim | Fully Connect Linear | Action Values 5 dim
(d)
o_t 200 dim | Conv 64|(1, 5)|2|BN ReLU | Conv 64|(1, 3)|1|BN ReLU
d_t 200 dim | Conv 64|(1, 5)|2|BN ReLU | Conv 64|(1, 3)|1|BN ReLU
Concatenate | F1 12288 dim | Fully Connect Linear | Action Values 5 dim
(e)

Fig. 6. Five other network structures for comparison. The parameters of convolutional layers in (e) are described in this format: filter number | filter size | stride length | BN = batch normalization. (a) One-frame Q-network. (b) Two-frame-1 Q-network. (c) Two-frame-2 Q-network. (d) Two-stream-1 Q-network. (e) Two-stream-CNN Q-network.

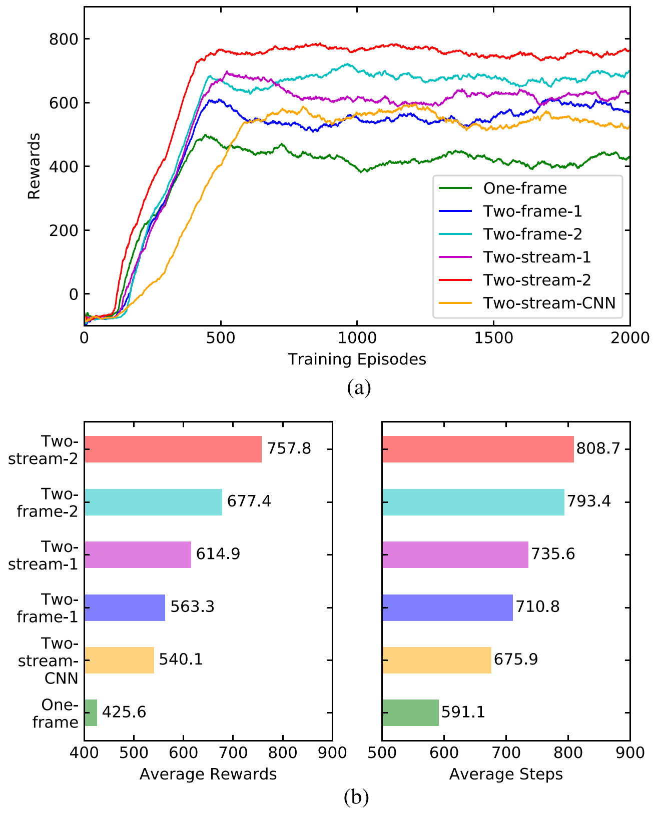


Fig. 7. Training processes and evaluation results of six Q-networks in obstacle avoidance subtask. (a) Sum of reward of each Q-network along the training episodes. The curves are filtered by moving average along the raw sum of reward series with sample length 200. For the first 199 points, the average values are taken from itself back to the first point. And starting from the 200 point, each point indicates the average value over itself and the previous 199 points. (b) Average sum of rewards and average moving steps of each Q-network structure over 1000 evaluation episodes after training.

to the deep Q-network in [1], which stacks four consecutive video game frames as the network input. Both spatial and temporal information is implicit in the input vector. The structure is shown in Fig. 6(b).

3) *Two-frame-2 Q-network:* The current frame and the difference with the previous frame, o_t and d_t, are connected into one vector as the input of this network. The structure is identical to the *two-frame-1 Q-network*. The input is very similar to our proposed two-stream Q-network. The only difference is that the temporal and spatial parts are processed together in one-stream style. The structure is shown in Fig. 6(c).
4) *Two-stream-1 Q-network:* The inputs are two consecutive frames of laser data, o_t and o_{t-1}. They are separately fed into the network whose structure is identical to our proposed two-stream Q-network. The structure is shown in Fig. 6(d).
5) *Two-stream-CNN Q-network:* The inputs are o_t and d_t. They are separately processed in two-stream style by two 1-D convolutional layers. Then, all output feature maps are concatenated into one vector and map to action values by a linear fully connected layer. This structure is modified from [28]. The structure and details of the convolutional layers are shown in Fig. 6(e).

Our proposed two-stream Q-network is denoted by *two-stream-2 Q-network* in the following discussions to distinguish it from other similar Q-networks, especially the *two-stream-1 Q-network* and *two-stream-CNN Q-network*, which also use two-stream structure.

To guarantee a fair comparison, those five networks are built and implemented with the same software and hardware platform. They are trained and tested in the same simulated environment following the same training procedures as our *two-stream-2 Q-network*. We also specially designed the parameters of each layer so that the numbers of network weights of all six networks are proximate except the *one-frame Q-network*, which is less than the others. After 2000 episodes of training, we set the $\epsilon = 0$ and evaluate them for another 1000 episodes in the same environment.

The performances of these Q-networks are measured by the sum of reward in each episode. Higher sum of reward means more collision-free moving steps, which indicates better policy. The sums of reward of six networks through the training progress are shown in Fig. 7(a). The average sum of rewards and average moving steps of 1000 episodes of evaluation after training are given in Fig. 7(b). The raw curves of reward summation in each episode show dramatic variance, even after the policies are learned. That is mostly because of the random movement of obstacles, which could corner the moving agent sometimes and results in inevitable collisions. A good policy can significantly reduce the possibility of being cornered, but it cannot be completely eliminated. The noisy learning curves make it difficult to observe the progress of each policy. To smooth the curves, we take moving average over the raw reward sum series and the sample length is selected as 200. The filtered curves show that the *two-stream-2 Q-network* is superior to the other five networks in both learning speed and final performance. The evaluation results in Fig. 7(b) also verify the same conclusion.

As discussed in Section III-B, the temporal information is critical for the agent to make correct decisions. For lack of temporal information, *one-frame Q-network* performs worse among all networks. Both *two-frame-1 Q-network* and *two-stream-1 Q-network* use two consecutive frames as the input and the temporal information is implicit in the input. The networks are able to extract useful representations of the temporal information. Therefore, the performance is better than *one-frame Q-network*. However, two consecutive frames do not clearly carry the idea of motion, and it may need more training steps or larger networks to get accurate temporal representations.

In *two-frame-2 Q-network*, although the input directly conveys both spatial and temporal information, these two parts are simply connected as one vector. As a result, it is highly likely that many irrelevant connections between the two parts are established in network, which could cause confusion in the training. This might be a reason why the two-frame networks perform worse than their corresponding two-stream versions. For example, *two-stream-1* obtains better performance than *two-frame-1* and *two-stream-2* is better than *two-frame-2*. Our proposed *two-stream-2 Q-network* not only uses the input that directly conveys both spatial and temporal information, but also separately processes them to avoid irrelevant coupling and thus get better performance.

Many recent approaches in navigation use RNN/LSTM to extract temporal information from sequential observations [38], [42]. However, in this avoidance task, the obstacles move at constant speed, and the relative speed can be directly acquired from the difference of two consecutive observations, which is enough for avoidance. Using LSTM to process a longer observation sequence may reveal some hidden states outside the sensory range, which are far less important. Introducing LSTM also makes the network structure and training procedure more complex. Therefore, in this avoidance task, the two-stream structure is more concise and efficient in dealing with temporal information.

For the structure of *two-stream-CNN Q-network*, we use a 1-D convolutional network instead of a fully connected network to extract features from the raw laser data. The 1-D convolutional networks have been used to process 1-D signals in speech recognition [54], electrocardiogram classification [55], and mechanical fault diagnosis [56]. It has also been used for processing the laser data in mobile robot navigation [28]. In the simulated environment, the moving obstacles could approach the agent in any direction. Therefore, the laser sensor must have a 360° FOV. From the example shown in Fig. 4, the laser measurements are in circular structures. However, they are actually stored and processed as a normal 1-D vector. As a consequence, it could be problematic to extract features from the head and tail parts of the vector. The convolutional layers are locally connected and neglect the fact that the head and tail parts are connected in nature. This leaves a "blind spot" at the direction where the circular structure is separated, which results in the unsatisfying performance of the *two-stream-CNN Q-network*. However, the "blind spot" does not exist in the fully connected networks, because all distance measurements are equally mapped to the next layer despite their positions in the raw vector. In [28], the testing platform is a two-wheeled robot with an LRF that has 270° FOV and 1080-D output data. The robot always moves forward in a static environment. As a result, the "blind spot" does not exist in that environment.

The pretrain of the offline part of navigation network is performed in *Room-1* without any moving obstacles. The observation is the 2-D relative coordinate of the destination. The Q-network has 32 hidden nodes and is initialized with random weights before training. At the beginning of each episode, the start point and the destination point are randomly initialized within the whole room. We employ the ϵ-greedy strategy for exploration. The random exploration probability ϵ starts from 1 and decreases by 0.02 if the agent reaches the destination until $\epsilon = 0.1$. The reward for arriving the destination is r_{dest}, and the time punishment is r_{time}, but no reward is assigned for collision. The values of the rewards r_{dest} and r_{time} can be found in Table II. The episode ends if the agent reaches the destination or collides with the walls. The weights update is performed at every step using stochastic gradient descent. After 1000 training episodes, we run 100 evaluation episodes to check the performance of the learned policy. The evaluation episodes run on the same environment and the ϵ is set to zero. Different from the training episode, we initialize the start and destination point within certain areas shown in Fig. 3. The results show that the agent successfully reaches the destination in all 100 episodes. It takes 67.2 steps on average, and the average steps of the best 10 and the worst 10 episodes are 41.8 and 83.7, respectively.

B. Testing and Results

After the pretrain and evaluation of the avoidance module and the offline part of the navigation module, we test the proposed modular DRL approach in the main navigation task. We also compare our approach with two other approaches. One is the conventional DQL approach. The other is the PFM. To evaluate the generalization capability of these approaches, we carried out the navigation experiments on four different rooms, which are described in Section III-A. In the following, we first briefly describe the configurations of all three approaches, and then present the testing results and discussions.

For our proposed approach, the configurations of the avoidance module and the offline part of the navigation module are given in Section IV-A and they are pretrained before testing. We use a one-hidden-layer fully connected Q-network to approximate the action-value function for the online part. The structure is 130(input)–128(hidden)–5(output). The hidden nodes are activated by ReLU and then are linearly mapped to the output

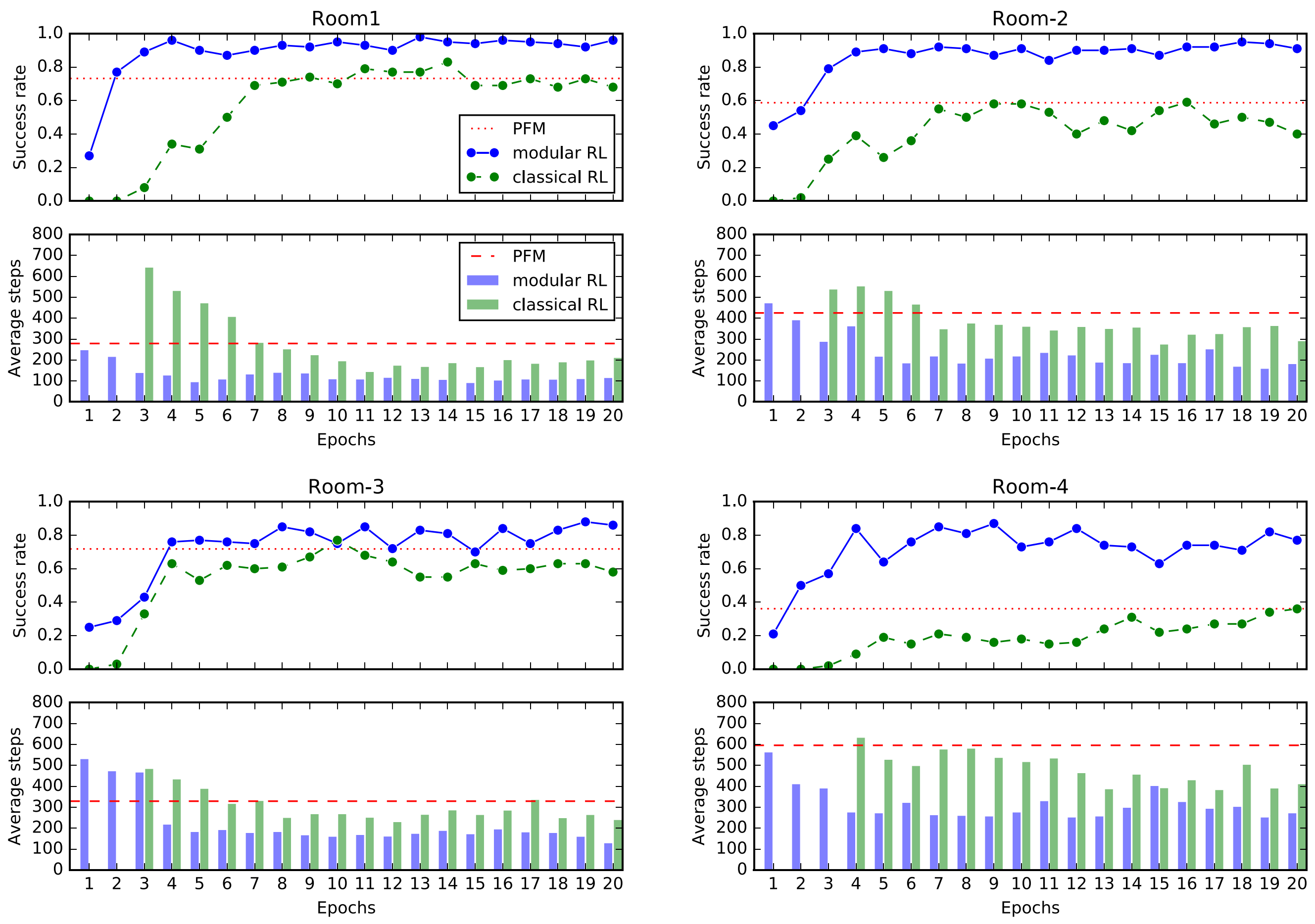


Fig. 8. Success rate and the average step of our modular DRL approach, the conventional DQL approach (classical DQL), and PFM of the main navigation tasks in four rooms. Each subfigure shows the results in one room. Each x-tick in the subfigure indicates an epoch which equals 100 consecutive episodes. The upper polyline charts show the success rate in each epoch. The lower bar charts demonstrate the average step in each epoch. The red dot lines and dash lines show the average success rate and average step of PFM, respectively. In the lower bar charts, the height of a bar is set to 0 if the corresponding success rate is lower than 0.05, because the average step cannot be accurately estimated if the number of successful episode in an epoch is zero or too few.

layer. The weights of all layers are randomly initialized. The rewards for collision, time cost of each step, and reaching the destination are r_{coli}, r_{time}, and r_{dest}, respectively, which are all listed in Table II. For the action scheduler, we set $\kappa = 1$ and $\lambda = 0.3$. The transfer probability φ is linearly decayed from 1 to 0 in the first 100 000 steps. The workflow of the action scheduler and online learning is listed in Algorithm 1. For the test, we first perform the online learning for 2000 episodes with the random exploration probability $\epsilon = 0.1$. Then, the trained model is evaluated for another 1000 episodes with $\epsilon = 0$.

The conventional DQL approach for the main navigation task is based on [1]. For the Q-network, we also use the proposed two-stream structure. The input of local observation is o_t and d_t. The inputs first separately go through two streams and then concatenated into a 128-D feature vector. This part of network is the same as the *two-stream-2* except the last fully connected layer. Then, we connect this feature vector with the global relative position input p_t and form a 130-D vector. Next, this vector goes through a one-hidden-layer network, which has the same structure as the online learning network described above and finally outputs the action values. The overall structure for the original approach is the same as our approach. We used the same reward setting and training procedure as that used for the online training of our proposed approach. The ϵ is set to 1 for the first 1000 steps and then linearly reduced to 0.1 in the next 100 000 steps. However, instead of learning with the help of the pretrained avoidance and navigation modules and the action scheduler, the conventional approach has to learn from scratch with random initial weights. To guarantee a fair comparison, this approach was trained for 5000 episodes instead of 2000 episodes, because our approach had performed 2000 episodes pretrain on the avoidance module and 1000 episodes on the navigation module. The trained network is evaluated for two times: the first time is after 2000 training episodes, and the second one is made after 5000 episodes. Each contains 1000 evaluation episodes with $\epsilon = 0$.

The PFM is a widely used approach for navigation in robotics. The basic concept is using an artificial potential field to demonstrate the environment. A mobile robot in the field is repulsed away from the obstacles and attracted to its target position. We followed the approaches in [15] and established a time-varying potential field around the agent, based on the agent's laser observation and relative positions. However, in the original algorithm, the output is the desired speed vector indicated by the negative

TABLE III
EVALUATION RESULTS IN FOUR ROOMS

	Ours	DQL2000	DQL5000	PFM
Room-1	0.953 (99)	0.781 (264)	0.792 (163)	0.732 (279)
Room-2	0.879 (191)	0.479 (329)	0.506 (286)	0.587 (425)
Room-3	0.840 (150)	0.626 (216)	0.711 (217)	0.718 (329)
Room-4	0.827 (333)	0.283 (368)	0.324 (347)	0.361 (596)

Note: DQL2000 and DQL5000 are the evaluation results of the conventional deep Q-learning approach after $2,000$ and $5,000$ training episodes. The numbers in parentheses are average steps; the numbers outside of parentheses are success rates.

gradient of the field, which is continuous. To make it compatible with the discrete action space in our environment, we use the following strategies:

1) get the current speed vector of the agent;
2) calculate five expected speed vectors for the execution of five actions;
3) choose the action resulting in minimum distance between the expected and desired speed vector.

In other words, we select the most effective action that turns the current speed to the desired speed. Furthermore, the parameters are carefully adjusted in each testing environment to the best of our capability. The PFM is evaluated in four rooms for 1000 repeated episodes.

The performance is evaluated by the success rate and the average step. In one episode, if the agent reaches the destination within $T_{\max}$ steps, the episode is a successful episode. The success rate refers to the percentage of successful episodes among all testing episodes. The success rate is the primary metric because higher success rate always means better policy. The average step is the average of count of steps before the agent successfully reaches the destination in each episode. Lower average step indicates higher efficiency of the policy. To guarantee a reasonable difficulty in each room, we put 20 moving obstacles in *Room-1*, and 15 in *Room-2* and *Room-3*, which are relatively complex. For *Room-4*, we only put 10 moving obstacles, because the layout is very narrow.

The success rate and average step of our approach, the conventional Q-learning approach, and PFM are shown in Fig. 8. For the first two learning-based approaches, the performance is summarized every 100 episodes or 1 epoch to demonstrate the online learning progress. Although the conventional Q-learning approach was trained for 5000 episodes (50 epochs), we only show the results of the first 20 epochs for comparison. For PFM, the average performances are marked by horizontal constant lines. The final evaluation results are given in Table III.

In Fig. 8, from the success rate curves in the upper subfigures, our approach demonstrates faster learning speed than the conventional DQL approach. After at most four epochs, our approach can generate stable policies in all four rooms. From the average step bar charts in the lower subfigures, we can find that both our approach and the conventional approach take fewer steps to achieve the destination as the training continues. However, the average step of the proposed approach decreases faster and is always less than the conventional approach. This indicates that our method is more efficient. The final evaluation results also show that the policies generated by the proposed approach get better performance than the policies generated by the conventional DQL and PFM with fine-tuned parameters. The stable and high-performance results in four different rooms, even in the very complex *Room-4*, reveal the great generalization capability and stability of the proposed approach.

The results also demonstrate the effectiveness of the modular design. The heuristic policy generated by the pretrained modules and the action scheduler significantly promotes the online learning speed compared with the conventional approach. That is because the training of the online part is based on exploration experiences of the heuristic policy, which gives many useful demonstrations on approaching the destination in the early stage. As a comparison, the conventional Q-learning approach uses random exploration at the early stage. Therefore, frequent collisions are inevitable. These collision experiences can help the agent to develop avoidance skills, which can be verified by the high average steps in the early stage. Also, the low success rates exposed its weakness in approaching the destination. In simple layouts like *Room-1*, the agent with only avoidance skills may reach the destination by accident and get a high positive reward, and then gradually develop the skill of reaching the destination as well as avoiding the obstacles. However, in a relatively complex environment, the probability of reaching the destination by accident is much lower. Therefore, it could be relatively hard to obtain an efficient policy due to the lack of success experiences. The learning process in *Room-4* is a clear example: our approach with the heuristic policy finds a good policy after three epochs, whereas the conventional approach only makes very small progress during the whole training process.

Learning from features instead of raw inputs is another important factor for the good performance. The combined spatial and temporal feature extracted by the avoidance module significantly reduces the dimension of local information for online training. Therefore, a smaller network can be used to approximate the action-value function, which facilitates the training speed.

Both our approach and the conventional approach use ϵ-greedy strategy for exploration, and the ϵ is set to 0.1 at the late stage. It is possible that two approaches finally have very close performances. However, from the evaluation results shown in Table III, the conventional approach only makes very small improvement from 2000 episode to 5000 episode, and it is expected to cost much longer time to achieve the performance like ours.

The PFM is not a learning-based algorithm, therefore, the performance partially depends on the parameters. The major difficulty of tuning the parameters for PFM is balancing the intensity and scope of the repulse field and the attraction field. This also leads to a tradeoff between success rate and average step. To avoid collisions and improve the success rate, one has to intensify the repulse field, which makes the agent perform more cautiously and leads to more steps. We manually tune the parameters for every layout only seeking higher success rate. However, we cannot get comparable success rate and average step with our approach. One major advantage of the PFM is no training required. It can obtain a good performance from the

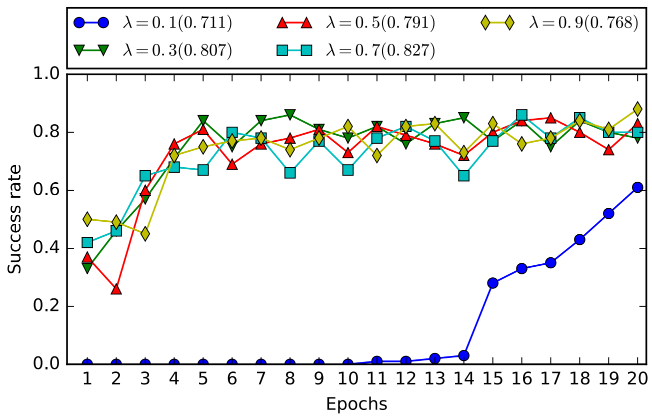


Fig. 9. Learning progress in *Room-4* with different switching threshold λ. The numbers in parentheses are the success rate of 1000 testing episodes after the training.

first attempt in a new environment, while our approach has to adapt for several episodes. In exchange, the model of the agent (at least the directional relationship between actions and laser measurements) is required, which violates the assumption of the simulated environment. Besides, there are many other conventional robotics navigation algorithms, such as VFH, VFH+, to name a few. Although they all claim better performance than PFM, the implementations demand accurate models of mobile robots, which is apparently incompatible with the assumption.

Our approach has two parameters about the action scheduler that potentially affect the final performance. We estimate the risk by predicting one step ahead using the current speed, thus κ is set to 1. λ affects the manner of the heuristic policy: larger λ makes the agent very sensitive to the risk and always makes safe movements, whereas smaller λ leads to bold actions toward the destination. To study how λ affects the learning progress, we conduct a set of experiments in *Room-4* using our approach with different switching thresholds. The learning progresses are shown in Fig. 9. Except for $\lambda = 0.1$, all other λ settings demonstrate similar learning progress. The curves and the success rates in testing episodes show that the performance of our approach is not very sensitive to λ as long as it is not too small.

V. Conclusion

In this paper, we develop a modular DRL architecture to address the navigation problem in complex dynamic environments. Specifically, built on top of the conventional DQL algorithm, we propose a new two-stream Q-network architecture, a spatial stream and a temporal stream, to solve the complex navigation problem. This structure achieved significantly improved performance compared to the conventional DQL approach in the moving obstacle avoidance task. With the help of the heuristic policy from two separately pretrained network modules and the action scheduling method, our approach can quickly obtain a high-performance navigation policy in new environments by online learning. Our approach also demonstrates high generalization capability, stability, and efficiency in experiments. For future work, we consider improving our approach with the hierarchical reinforcement learning structures and deterministic policy gradient algorithm and address more general and realistic navigation problems such as continuous action space and concurrent activities. Furthermore, we will consider testing our approach in real-world dynamic complex environments like a crowded shopping mall in the future.

References

[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[2] T. Barron, M. Whitehead, and A. Yeung, "Deep reinforcement learning in a 3-D blockworld environment," in *Proc. Deep Reinforcement Learn., Frontiers Challenges, Int. Joint Conf. Artif. Intell.*, 2016, vol. 2016, p. 16.
[3] J.Oh *et al.*, "Control of memory, active perception, and action in Minecraft," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2790–2799.
[4] M. Dann, F. Zambetta, and J. Thangarajah, "Integrating skills and simulation to solve complex navigation tasks in Infinite Mario," *IEEE Trans. Comput. Intell. AI Games*, vol. 10, no. 1, pp. 101–106, Mar. 2018.
[5] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3357–3364.
[6] M. S. Emigh, E. G. Kriminger, A. J. Brockmeier, J. C. Príncipe, and P. M. Pardalos, "Reinforcement learning in video games using nearest neighbor interpolation and metric learning," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 1, pp. 56–66, Mar. 2016.
[7] M. McPartland and M. Gallagher, "Reinforcement learning in first person shooter games," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 1, pp. 43–56, Mar. 2011.
[8] D. Wang and A.-H. Tan, "Creating autonomous adaptive agents in a real-time first-person shooter computer game," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 2, pp. 123–138, Jun. 2015.
[9] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," U.S. Patent 15/217,758, Jan. 26, 2017.
[10] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.
[11] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[12] A. Ng *et al.*, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Berlin, Germany: Springer-Verlag, 2006, pp. 363–372.
[13] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 263–279, 2013.
[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[15] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.
[16] P. A. Veatch and L. S. Davis, "Efficient algorithms for obstacle detection using range data," *Comput. Vis., Graph., Image Process.*, vol. 50, no. 1, pp. 50–74, 1990.
[17] W. Tianmiao and Z. Bo, "Time-varying potential field based 'perception-action' behaviors of mobile robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1992, pp. 2549–2554.
[18] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonom. Robots*, vol. 13, no. 3, pp. 207–222, 2002.
[19] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, Jun. 1991.
[20] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1998, vol. 2, pp. 1572–1577.
[21] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, vol. 3, pp. 2505–2511.
[22] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.
[23] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 229–241, Jun. 2001.

[24] M. Montemerlo *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. Nat. Conf. Artif. Intell.*, 2002, pp. 593–598.
[25] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.
[26] I.-K. Jung and S. Lacroix, "High resolution terrain mapping using low attitude aerial stereo imagery," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2003, pp. 946–951.
[27] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 2759–2764.
[28] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1527–1533.
[29] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, 2003.
[30] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1–2, pp. 181–211, 1999.
[31] K. Rohanimanesh and S. Mahadevan, "Learning to take concurrent actions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 1651–1658.
[32] R. Parr and S. J. Russell, "Reinforcement learning with hierarchies of machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 1998, pp. 1043–1049.
[33] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.
[34] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[35] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, vol. 48, pp. 1995–2003.
[36] V. Mnih *et al.*"Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[37] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.
[38] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 29–37.
[39] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," arXiv:1610.01733, 2016.
[40] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 31–36.
[41] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in Minecraft," in *Proc. AAAI*, 2017, vol. 3, pp. 1553–1561.
[42] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. AAAI*, 2017, pp. 2140–2146.
[43] P.Mirowski *et al.*, "Learning to navigate in complex environments," arXiv:1611.03673, 2016.
[44] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.
[45] M. A. Goodale and A. D. Milner, "Separate visual pathways for perception and action," *Trends Neurosci.*, vol. 15, no. 1, pp. 20–25, 1992.
[46] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 568–576.
[47] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Fujitsu Lab., Ltd., Kawasaki-shi, Japan, 1993.
[48] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[49] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems*. New York, NY, USA: Wiley-Interscience, 1972.
[50] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[51] A. Y. Ng, "Shaping and policy search in reinforcement learning," Ph.D. dissertation, Univ. California, Berkeley, Berkeley, CA, USA, 2003.
[52] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," *OSDI*, vol. 16, pp. 265–283, 2016.
[53] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980, 2014.
[54] P. Golik, Z. Tüske, R. Schlüter, and H. Ney, "Convolutional neural networks for acoustic modeling of raw time signal in LVCSR," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc.*, 2015, pp. 26–30.
[55] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 3, pp. 664–675, Mar. 2016.
[56] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-D convolutional neural networks," *IEEE Trans. Ind. Electron.*, vol. 63, no. 11, pp. 7067–7075, Nov. 2016.

Yuanda Wang received the B.S. degree in automation from Nanjing University of Information Science and Technology, Nanjing, China, in 2014. He is currently working toward the Ph.D. degree in control science and engineering at the School of Automation, Southeast University, Nanjing, China.

Since 2016, he has been a joint Ph.D. student with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, USA. His research interests include reinforcement learning, neural network, and multi-agent systems.

Haibo He (SM'11–F'18) received the B.S. and M.S. degrees from Huazhong University of Science and Technology, Wuhan, China, in 1999 and 2002, respectively, and the Ph.D. degree from Ohio University, Athens, OH, USA, in 2006, all in electrical engineering.

From 2006 to 2009, he was an Assistant Professor with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA. He is currently the Robert Haas Endowed Chair Professor with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, USA. His research interests include adaptive learning and control, computational intelligence, machine learning and data mining, and various applications. He has authored or coauthored one sole-author research book (Wiley), edited one book (Wiley-IEEE) and six conference proceedings (Springer), and also authored and coauthored more than 300 peer-reviewed journal and conference papers.

Dr. He was the General Chair of the IEEE Symposium Series on Computational Intelligence (2014). He was the recipient of the IEEE International Conference on Communications Best Paper Award (2014), the IEEE Computational Intelligence Society Outstanding Early Career Award (2014), the National Science Foundation CAREER Award (2011), and the Providence Business News "Rising Star Innovator Award" (2011). He is currently the Editor-in-Chief for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Changyin Sun received the B.S. degree in applied mathematics from the College of Mathematics, Sichuan University, Chengdu, China, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Southeast University, Nanjing, China, in 2001 and 2003, respectively.

He is currently a Professor with the School of Automation, Southeast University. His research interests include intelligent control, flight control, pattern recognition, optimal theory, etc.

Dr. Sun is an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, *Neural Processing Letters*, and the IEEE/CAA JOURNAL OF AUTOMATICA SINICA.