Artificial Intelligence for Engineering Design, Analysis and Manufacturing

cambridge.org/aie

Research Article

Cite this article: Liu X, Jin Y (2020). Reinforcement learning-based collision avoidance: impact of reward function and knowledge transfer. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 1–16. https://doi.org/10.1017/S0890060420000141

Received: 19 March 2019
Revised: 7 January 2020
Accepted: 11 January 2020

Key words:
Agent-based systems; autonomous vehicle; collision avoidance; deep reinforcement learning; machine learning

Author for correspondence:
Yan Jin, E-mail: yjin@usc.edu

Reinforcement learning-based collision avoidance: impact of reward function and knowledge transfer

Xiongqing Liu and Yan Jin

Department of Aerospace and Mechanical Engineering, University of Southern California, 3650 McClintock Avenue, OHE-430, Los Angeles, CA 90089-1453, USA

Abstract

Collision avoidance for robots and vehicles in unpredictable environments is a challenging task. Various control strategies have been developed for the agent (i.e., robots or vehicles) to sense the environment, assess the situation, and select the optimal actions to avoid collision and accomplish its mission. In our research on autonomous ships, we take a machine learning approach to collision avoidance. The lack of available ship steering data of human ship masters has made it necessary to acquire collision avoidance knowledge through reinforcement learning (RL). Given that the learned neural network tends to be a black box, it is desirable that a method is available which can be used to design an agent's behavior so that the desired knowledge can be captured. Furthermore, RL with complex tasks can be either time consuming or unfeasible. A multi-stage learning method is needed in which agents can learn from simple tasks and then transfer their learned knowledge to closely related but more complex tasks. In this paper, we explore the ways of designing agent behaviors through tuning reward functions and devise a transfer RL method for multi-stage knowledge acquisition. The computer simulation-based agent training results have shown that it is important to understand the roles of each component in a reward function and the various design parameters in transfer RL. The settings of these parameters are all dependent on the complexity of the tasks and the similarities between them.

Introduction

In the recent years, the development of autonomous vehicles, especially autonomous cars, has been a common topic reported frequently in newspapers and television programs, thanks to the advance in control systems and the proliferation of machine learning techniques. Our research on collision avoidance for ships at sea aims to develop technologies that will eventually lead to the future autonomous ships that can not only steer in open waters and less congested areas but also are capable of avoiding collisions in congested harbors and reaching alongside berths without direct human involvement.

There have been various approaches to solving vehicle collision avoidance problems which can be divided into two large categories. One is *vehicle control* system development and the other *traffic control* system development, with the former being more relevant to this research. Vehicle control can be further categorized into the *dynamical systems* approach (e.g., Machado *et al.*, 2016), which relies on traditional control theories, and the *intelligent systems* approach (e.g., Yang *et al.*, 2017), which employs knowledge-based systems and machine learning techniques to make collision avoidance decisions. Although the dynamical systems approach can be effectively applied in mostly predictable circumstances, when the uncertainty level becomes high and exceptions happen, the intelligent systems approach will be needed. Traditional knowledge-based systems have been applied to collision avoidance (e.g., Jin and Koyama, 1987). However, the issues of knowledge acquisition, formalization, and management have remained to be practically challenging.

The recent progress in machine learning, especially the deep learning (LeCun *et al.*, 2015), has opened the ways to developing systems that can learn from humans' operation experiences (e.g., through supervised deep learning) and from machines' own experiences [e.g., through reinforcement learning (RL)]. The RL approach allows an agent to learn from its own experience. By interacting with the environment, the agent learns to select actions at any state to maximize its total reward. In case of deep learning, for example, AlphaGo (Chen, 2016), the agent learns from the experience of human experts and apply the learned skills to solving the problems in the same domain of the experts. Furthermore, computer agents can learn to play games, for example, Go, by themselves through RL without any input from humans. AlphaGo-Zero is such an example that even won the human experience based AlphaGo in game matches (Silver *et al.*, 2017).

In this research, the attempts have been made to apply the human experience-based supervised learning approach to train neural networks, but the effectiveness has been limited, largely because it was a nontrivial process to acquire sufficient human steering data to train the computer agents to a high skill level. Only rarely could we have opportunities to obtain meaningful data for complex ship encounters and the amount of such data is often too small for the agent training purpose. Therefore, we adopted the RL approach to acquire collision avoidance knowledge in complex ship encounter situations.

From a system point of view, the development of RL agents needs to satisfy two requirements. The first requirement relates to *ship behavior design*. Although through deep RL, ship agents can learn their optimal policies given the *state* and *action spaces*, the *state transition*, and *reward functions*, the learned *neural network* is still a black box that provides little identifiable knowledge about agents' action mechanisms. To make agents' behavior more transparent, it is important to understand how the agents' behavior changes in response to the *change of reward functions*. In this paper, an initial effort in turning the reward functions for ship agent behavior design is presented.

Second, training in RL takes time, and the capability of *transferring previously learned knowledge* (i.e., neural network) in the new learning situations effectively is needed not only for speeding up the learning processes but also for expanding the range of situations by including the ones covered by other agents. This requirement calls for transfer learning in the reinforcement learning context. In this paper, a *belief-based transfer RL* method is introduced, and its design features are investigated and presented.

To summarize, in this research we address two research questions: (1) *how the agent behavior is determined by reward functions in RL*? and (2) *what are the important parameters that impact both the learning and task behaviors of the agent in a transfer RL context*?

The rest of the paper is organized as follows. The related work is reviewed in the section "Related work", and then, the task of ship collision avoidance is described in the section "Tasks of ship collision avoidance". "Machine learning based knowledge capture" provides details of a machine learning approach for collision avoidance, and section "Case study design" presents the design of case studies, and the results and discussion are described in the section "Results and discussion". The conclusions are drawn in the section "Conclusions" together with the directions of future work.

Related work

Collision avoidance

Over the years, collision avoidance has been a common research topic in many industrial domains, including robotics, shipping, air traffic control, and self-driving cars. In the area of robotics, research has focused on issues related to how vehicle robots avoid obstacles and collisions with each other (Brunn, 1996; Alonso-Mora *et al.*, 2013; Shiomi *et al.*, 2014) and how assembly robots, or manipulators, avoid interferences among its own arms or with those of others (Hourtash *et al.*, 2016; Hameed and Hasan, 2016). In the sea and river shipping industry, collision avoidance can be highly difficult, when the water areas are becoming congested, due to the large inertia of ships causing immovability when movement is needed (Goerlandt and Kujala, 2014). Once a collision happens at sea, the loss can be tremendous (Eleftheria *et al.*, 2016). Airplane collision avoidance (Zou *et al.*, 2016) and even the collision with debris in space (Casanova *et al.*, 2014) have become issues due to the increasing level of congestion. The self-driving industry is a rapidly growing field in the recent years, and fully autonomous cars are expected to become available soon. A self-driving system is considered to be more engineered in the sense that each car agent is equipped with more intelligence, which is at least able to (a) process human-level perceptions, that is, detecting lanes, traffic signs, pedestrians, other vehicles/obstacles, etc.; (b) predict trajectories of pedestrians and other vehicles; and (c) deal with complex social interactions with other vehicles and pedestrians. After all, self-driving cars must be able to avoid obstacles and other vehicles in various situations (Mukhtar *et al.*, 2015; Kuderer *et al.*, 2015).

Collision avoidance problems have always attracted the attention of researchers in various research fields such as artificial intelligence, control theory, robotics, multi-agent system, and so on. The traditional practice to achieve real-time obstacle avoidance was to create an artificial potential field (Khatib, 1986). Fahimi *et al.* (2009) proposed harmonic potential functions and the panel method to address multi-robot obstacle avoidance problem in the presence of both static and dynamic obstacles. Mastellone *et al.* (2008) designed a controller for collision avoidance based on the Lyapunov-type approach and demonstrated the robustness of the system when the communication between robots was unreliable. Keller *et al.* (2016) designed a path planner for unmanned aircraft systems to provide surveillance by combining graph search and B-spline parametric curve construction, which could successfully navigate around obstacles and provide sufficient coverage. Tang and Kumar (2015) proposed the OMP + CHOP algorithm for a centralized multi-robot system, which was shown to be safe and complete, but at the cost of optimality.

Machine learning

In order for collision avoidance algorithms to be more adaptive and flexible in the real-world complex environment, learning capabilities of a multi-agent system have been developed. In the recent years, deep learning has achieved tremendous success in various areas such as image recognition (Krizhevsky *et al.*, 2012; Le *et al.*, 2012), speech recognition (Hinton *et al.*, 2015), automatic game playing (Mnih *et al.*, 2013), self-driving (Bojarski *et al.*, 2016; Ohn-bar and Trivedi, 2016), and so on. Deep learning algorithms can extract high-level features by utilizing deep neural networks, such as convolutional neural networks (CNNs) (Krizhevsky *et al.*, 2012), multi-layer perceptrons, and recurrent neural networks (RNNs) (LeCun *et al.*, 2015). Scaling up deep learning algorithms is able to discover high-level features in a complex task. Dean *et al.* (2012) constructed a very large system, which was able to train 1 billion parameters using 16,000 CPU cores. Coates *et al.* (2013) scaled to networks with over 11 billion parameters using a cluster of GPU servers.

RL is about an agent interacting with the environment, learning an optimal-policy, by trial-and-error, for sequential decision-making problems in a wide range of fields in both natural and social sciences, and engineering (Sutton and Barto, 2018; Bertsekas and Tsitsiklis, 1996; Matarić, 1997; Szepesvari, 2010). Mnih *et al.* (2013) introduced a deep learning algorithm using experience replay and CNNs to learn a Q-function, which can play various Atari 2600 games better than human players. Experience replay allows an online learning agent to random sample batches from past experiences to update Q-values, thus breaking the correlations between consecutive frames. By combining the supervised learning and RL approach, the group at DeepMind has further proven that their deep learning algorithm can outperform a world champion in the most challenging classic game Go (Silver *et al.*, 2016, 2017; Churchland and Sejnowski, 2016; Wang *et al.*, 2016a), which has extremely large number of possible configurations, and is difficult to evaluate board positions. Schaul *et al.* (2016) further developed a prioritized experience replay framework to sample more important transitions and learn more efficiently. For reward function studies, there have been ways to do inverse RL (Ng and Russell, 2000), reward strategy analysis (Manju and Punithavalli, 2011), and reward function generation (Mericli *et al.*, 2010).

Chen *et al.* (2016) developed a decentralized multi-agent collision avoidance algorithm based on deep RL. Two agents were simulated to navigate toward their own goal positions and learn a value network which encodes the expected time to goal, and the solution was then generalized in multi-agent scenarios. Deep learning algorithms have been successful in achieving end-to-end learning. Dieleman and Schrauwen (2014) investigated whether it is possible to apply feature learning directly to raw audio signals by training CNNs. Traditionally, content-based music information retrieval tasks are resolved based on engineered features and shallow processing architectures, which relies on mid-level representations of music audio, for example, spectrograms. The results showed that even though the end-to-end learning does not outperform the spectrogram-based approach, the system is able to learn automatically frequency decompositions and feature representations from raw audio.

Self-driving cars mentioned above is a promising field which took off in the last few years and heavily relies on the advances in deep learning. Since self-driving cars always require a great deal of expensive and complex hardware, Yu *et al.* (2016) implemented a deep Q-learning algorithm using dataset (images) from real-time play of the game JavaScript Racer. In a recent published paper (Bojarski *et al.*, 2016), a CNN is trained to map steering commands directly from raw pixels from camera input. The system automatically learned internal processing steps such as detecting useful road features with only the human steering angle as the training signal. This end-to-end learning approach is challenging in that it requires a huge number of inputs and the advantage is that it releases the rely on the designer's prior domain knowledge.

Transfer learning

Given a complicated task which is difficult to learn directly, transfer learning is a commonly used technique which can generalize previously learned experience and apply these experiences into new tasks (Arnold *et al.*, 2007; Pan and Yang, 2010; Bahadori *et al.*, 2014). Transfer learning refers to utilizing knowledge gained from source tasks to solve a target task. It is believed that in an RL context, transfer learning can speed up the learning agent to learn a new but related task (i.e., target task) by learning source tasks first. Taylor and Stone (2007) introduced a transfer algorithm called *Rule Transfer*, which summarizes source task policy, modifies the decision list, and generates a policy for the target task. Rule learning is well understood and human readable. The agent benefits from the decision list initially and continues to refine its policy through target task training. It was shown that Rule Transfer could significantly improve learning in robot soccer using learned policy from a grid-world task.

Fernandez and Veloso (2006) proposed two algorithms to address the challenges of *Policy Reuse* in an RL agent. The major components include an exploration strategy and a similarity function to estimate the similarity between past policies and new ones. The PRQ-learning algorithm probabilistically bias an exploration learning process by using a *Policy Library*. In the second algorithm called PLPR, the Policy Library is created when learning new policies and reusing past policies.

Torrey *et al.* (2006) introduced the induction logic programming for analyzing the previous experience of source task and transferred rules for when to take actions. Through an advice-taking algorithm, the target task learner could benefit from outside imperfect guidance. A system AI^2 (Advice via Induction and Instruction) for transfer learning in RL was built, which creates relational *transfer advice* using inductive logic programming. Based on a human-provided mapping from source tasks to target tasks, the system was able to speed up RL.

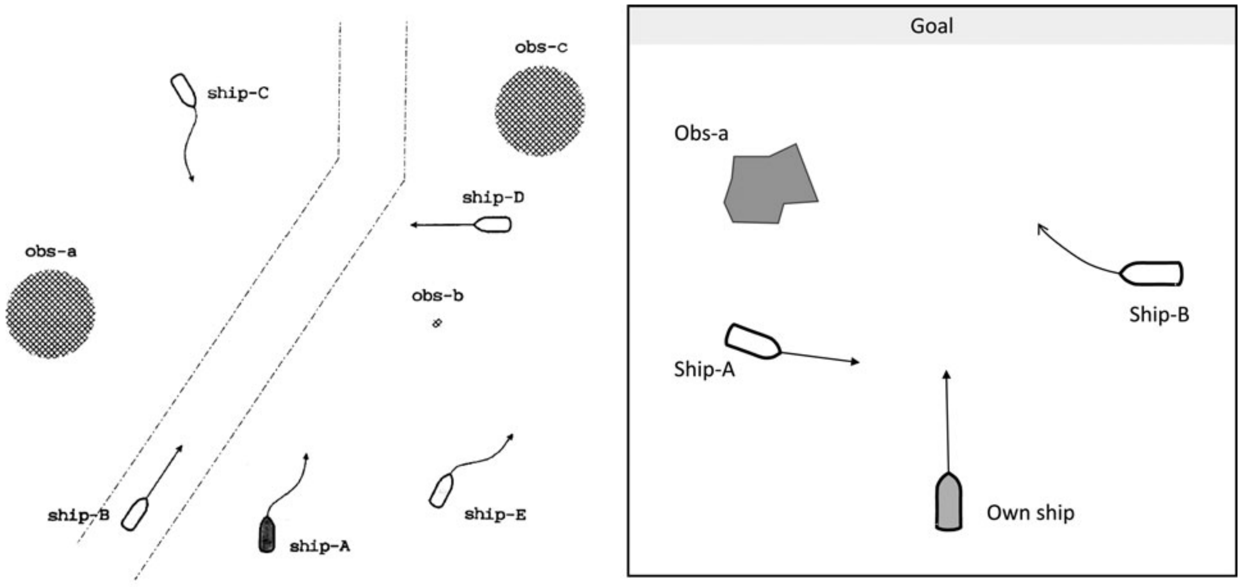
In transfer learning within deep neural networks, a base network on a base dataset and task is first trained, and the learned features are then transferred to the target network to be trained on a target dataset and task. Transfer learning can be particularly useful to train a large target network without overfitting if the target dataset is much smaller than the base/source dataset. One common approach is to copy the first n layers of the base network to the first n layers of the target network, while the remaining layers of the target network are randomly initialized and trained. Yosinski *et al.* (2014) presented a way to measure the degree to which a certain layer is general or specific and found that initializing a network with transferred features from almost any layers could boost the performance after fine-tuning to a new dataset. A task-driven deep transfer learning framework for image classification was designed (Ding *et al.*, 2016), where the features and classifiers are obtained at the same time. Through pseudo labels for target domain, the system could transfer more discriminative information to the target domain. Parisotto *et al.* (2016) proposed a transfer reinforcement learning approach (Actor-Mimic) to mimic expert decisions for multi-task learning, which adopts the concept of *policy distillation* (Hinton *et al.*, 2015).

To date, there has been little literature aiming to combine deep RL and transfer learning to solve robotic collision avoidance problems, because (a) it is difficult to directly learn from raw pixel or distance sensory inputs and (b) it requires large amount of training data, which is not easy to generate in real life. This research combines transfer and RL approaches for ship agents to acquire collision avoidance knowledge more efficiently.

Tasks of ship collision avoidance

Collision avoidance at sea

Collision avoidance at sea involves a ship agent, called own ship, a number of target ships and obstacles, and the designated routes for different types (e.g., different size) of ships. Figure 1a illustrates a typical ship encounter situation, where Ship-A is the own ship, Ships B, C, D, and E are target ships of the own ship. Each ship has its own destinations such as a specific berth when moving into a port or an exit direction when moving out. There are also obstacles, such as Obs-a, Obs-b, and Obs-c, which can be islands, fixtures, or buoys. The routes are often indicated on the hardcopy or electronic maps, called charts. In addition, there are regulations, dictated by COLREG rules (Ford, 2009), by which the ships will be considered as having the "right of the road" or should "give way" to other ships depending



(a) An illustration of ship collision avoidance

(b) A simplified ship collision avoidance task

Fig. 1. Ship collision avoidance at sea.

on their position in the encounter situation. Although the regulations are supposed to be followed by all the ships, there is no guarantee that a target ship will follow the rules, especially when the situation can be interpreted in different ways by different ships.

In this research, since our focus is to investigate the issues related to RL-based collision avoidance knowledge capture, we define a simplified version of the tasks of collision avoidance. As shown in Figure 1b, the own ship is on track moving north (or upward). The destination of the own ship is any point on the upper edge of the rectangle, indicated as “Goal”. So the “success” of the own ship in this task is defined as reaching the goal without hitting any other ships or obstacles, and the “efficiency” of the own ship in a *successful run* is defined by the normalized inverse of the total number of units of distance (or steps) traveled by the ship. The efficiency of *unsuccessful runs* is defined as 0 (zero). There can be stationary obstacles and moving ships in the water area. The objective of RL based knowledge capture is to train ship agents, so that they can achieve successful runs with high efficiency for various encounter situations.

Task complexity and similarity

The collision avoidance tasks indicated in Figure 1 can be simple or more complex depending on how many target ships and obstacles are involved and how difficult the encounter situation can be. From a knowledge capture perspective, it will be ideal that the ship agents can be trained in simple situations and the knowledge (i.e., neural networks) obtained from the simpler situations can be applied and extended through further learning in more complex situations. It is conceivable that knowledge transfer between similar cases tends to be easier than that between dissimilar cases. To facilitate the knowledge transfer study, simple measures of task *complexity* and *similarity* are introduced as follows.

The complexity of a collision avoidance task can be generally considered to have two major components, namely *internal complexity* and *external complexity*. The internal complexity stems from *how difficult the own ship can be maneuvered* to avoid collision (e.g., large ships are hard to move, and smaller ones easier to move), while the external complexity captures *how difficult the encounter situation can be* to avoid collision. Since moving targets are harder to avoid than stationary obstacles and more targets or obstacles are harder to deal with, the external complexity can be further decomposed into *object complexity* – measured by whether an object is moving (ship) or stationary (obstacle) and *aggregate complexity* – measured by the total number of target ships and obstacles. Figure 2 illustrates such a complexity hierarchy.

In this research, the similarity between two collision avoidance tasks is measured by the inverse of the “distance” between the two tasks in the three-dimensional task complexity space, as indicated in Figure 3a. The three axes in the figure are x-axis is the obstacle dynamics plane (red), y-axis is the number of obstacles plane (green), and z-axis is the vehicle dynamics plane (blue). Figure 3b shows a specific contextualization of this task complexity model, with the three dimensions having specific value ranges specified as *Vehicle dynamics* = {*large ship, small ship*}, *Obstacle dynamics* = {*static, moving*}, and *Number of obstacles* = {*single, multiple*}.

For any two tasks i and j in Figure 3b, their task complexity can be determined by two vectors: $\overrightarrow{\text{complexity}_i} = (\text{OD}_i, \text{NO}_i, \text{VD}_i)$ and $\overrightarrow{\text{complexity}_j} = (\text{OD}_j, \text{NO}_j, \text{VD}_j)$, respectively, where *OD* is *obstacle dynamics*, *NO* is the *number of obstacles*, *VD* is *vehicle dynamics*. The weighted length of the vectors from the origin can be used as a scaler measure for the task complexity, that is,

$$||\overrightarrow{\text{complexity}_i}|| = \sqrt{\lambda_1 \text{OD}_i^2 + \lambda_2 \text{NO}_i^2 + \lambda_3 \text{VD}_i^2}. \quad (1)$$

In this research, the *inter-task similarity* of the two tasks, i and j is defined as the distance between the two points in the complexity space given as follows:

$$\text{similarity}_{i,j} = \frac{1}{1 + \sqrt{(\text{OD}_i - \text{OD}_j)^2 + (\text{NO}_i - \text{NO}_j)^2 + (\text{VD}_i - \text{VD}_j)^2}}. \quad (2)$$

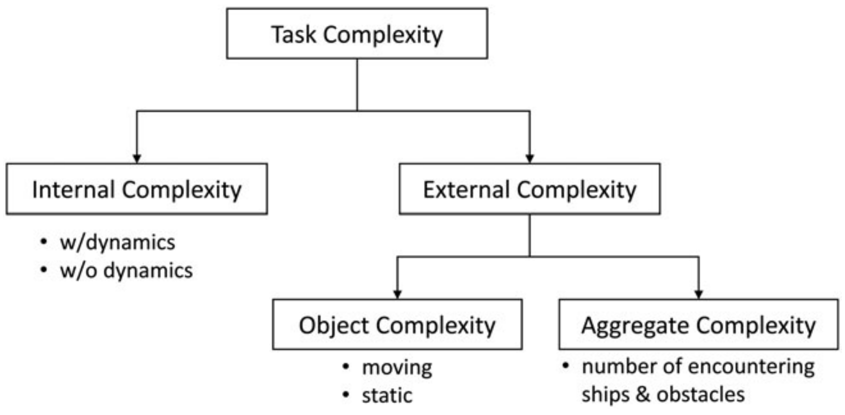


Fig. 2. Collision avoidance task complexity hierarchy.

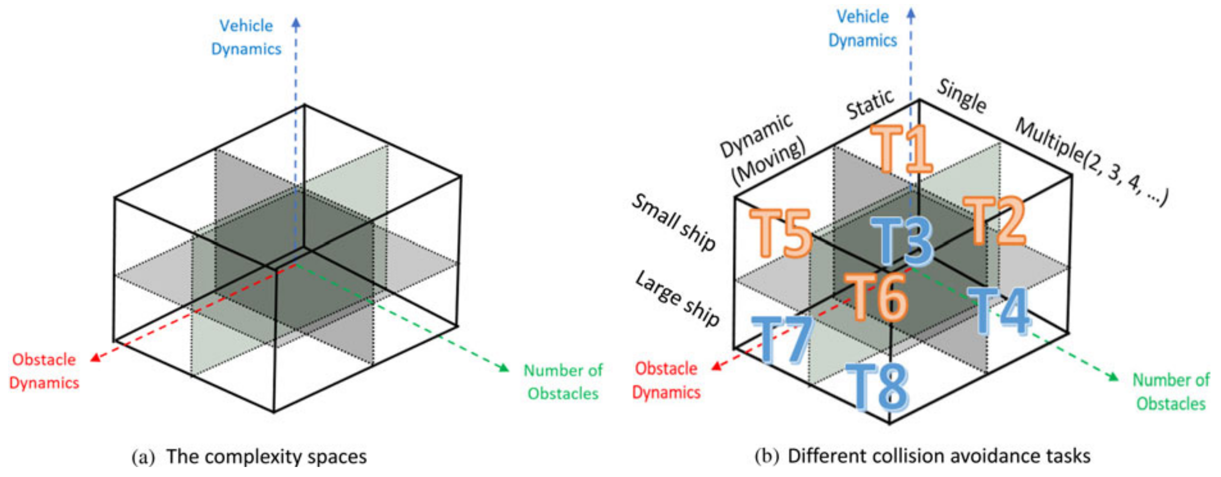


Fig. 3. Complexity spaces and task locations.

In general, the three factors may not be equally important in determining the inter-task similarity. Thus, three additional weights can be added, as we did in Eq (1):

$$\text{similarity}_{i,j} = \frac{1}{1+\sqrt{\lambda_1(\mathrm{OD}_i-\mathrm{OD}_j)^2+\lambda_2(\mathrm{NO}_i-\mathrm{NO}_j)^2+\lambda_3(\mathrm{VD}_i-\mathrm{VD}_j)^2}}. \quad (3)$$

Determining the values of the relative weights of λ_1, λ_2, and λ_3 require a deep understanding of the collision avoidance tasks in corresponding domains. In the case study section of this paper, a simplified measuring scheme is adopted for simulation and training purposes.

Machine learning based knowledge capture

For the collision avoidance tasks and the similarity definition introduced above, an RL approach is adopted to (1) capture the collision avoidance knowledge through trial-and-error and (2) transfer the captured knowledge to closely related but more complex tasks. Following subsections describe the principles and algorithms of the methods introduced.

Knowledge capture: deep RL

RL for collision avoidance involves sensory data and collection of sufficient data through trial-and-error processes. However, it can be difficult to collect a sufficient amount of data as the training input by only relying on sensory inputs. In addition, the state/action space is always continuous which makes it impractical to build a look-up Q-table. To overcome the curse of dimensionality, deep neural networks are used as functional approximators to replace the Q-table and approximate Q-values.

We began this research by implementing the deep RL algorithm with experience replay as proposed in Mnih *et al.* (2013). First, standard Q-learning (Watkins, 1989) is considered, which can be formulated as a tuple of

$$\text{Q-learning} : \langle S, A, P, R, \gamma \rangle, \quad (4)$$

where $S = \{s_1, \ldots\}$: state space; $A = \{a_1, \ldots\}$: action space; P : transition matrix; $R_t = \sum_{t'}^{T} \gamma^{t'-t} r_{t'}$: reward function; and γ : discount rate.

In Eq. (4), S is the state space, which consists of agent's all possible states in the environment. A is the action space consisting of all the possible actions that the agent can take. P is the transition matrix (usually unknown in a model-free environment), R is the reward function, and γ is the discount factor. At any given time t, the agent's goal is to maximize its future discounted return $R_t = \sum_{t'}^{T} \gamma^{t'-t} r_{t'}$, where T is the time when the game or episode terminates. Like many other RL algorithms, the agent estimates at each time step the action value function $Q(s, a)$, using a Bellman equation, Eq. (5), as an update. Such value iteration algorithms converge to the optimal value function:

$$Q_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]. \tag{5}$$

In order to adapt to tasks involving infinitely large state/action space where building the Q-table is impractical, deep Q-learning with experience replay uses a neural network as a function approximator (Q-network). A Q-network with weights θ_i can be trained by minimizing the loss function $L_i(\theta_i)$ at each iteration i,

$$L_i(\theta_i) = \mathbb{E}[(y_i - Q(s, a; \theta_i))^2], \tag{6}$$

where $y_i = \mathbb{E}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \right]$ is the target Q-value for iteration i. The gradient is calculated by the following:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]. \tag{7}$$

The deep Q-learning algorithm utilizes a technique called *experience replay* where the agents' experiences, $e_t = (s_t, a_t, r_t, s_{t+1})$, are stored into a *replay memory*, $D = \{e_1, e_2, \ldots, e_N\}$ (N is the capacity of the replay memory). Then, mini-batches are randomly sampled from D and applied to Q-learning updates. The agent selects an action according to the ε-greedy policy.

Various approaches have been proposed to stabilize learning process such as deep Q-network (DQN) (Mnih *et al.*, 2013), double DQN (van Hasselt *et al.*, 2015), and dueling DQN (Wang *et al.*, 2016*b*). In this research, our base network is built by combining these three approaches. The double DQN algorithm is able to solve the problem of overoptimistic value estimates, by separating the target network, which is used for action evaluation, from the current network which is used for action selection. The agent's experience is randomly assigned to update one of the two networks.

In standard DQN, at each update of the Q-values, only the value for one of the actions is updated, whereas others remain untouched. Dueling DQN separates the Q-value into a state value and action advantages, so that the state value is updated more frequently.

Agent learning behavior

In this research, a computer game environment was created to conduct case studies, as shown in Figure 4. The game environment consists of a learning agent (the green ball), static and moving obstacles (the red balls), and a goal area (the orange rectangle). The dark slice on the ball indicates the direction of the ball's movement. An end-to-end deep learning approach (Mnih *et al.*, 2013) is taken and the game window images are captured as input state information. Therefore, a state $s_i \in S$ in Eq. (4) for this study is defined as the *pixel values* of the game window. Figure 4 shows an example of the game window.

The action space is composed of seven actions, $A = \{a_1, \ldots, a_7\}$, as indicated in Table 1. Each action is defined by a pair of forward velocity v, and angular velocity ω, that is, $a_i = (v_i, \omega_i)$, $i = 1, 2, \ldots, 7$.

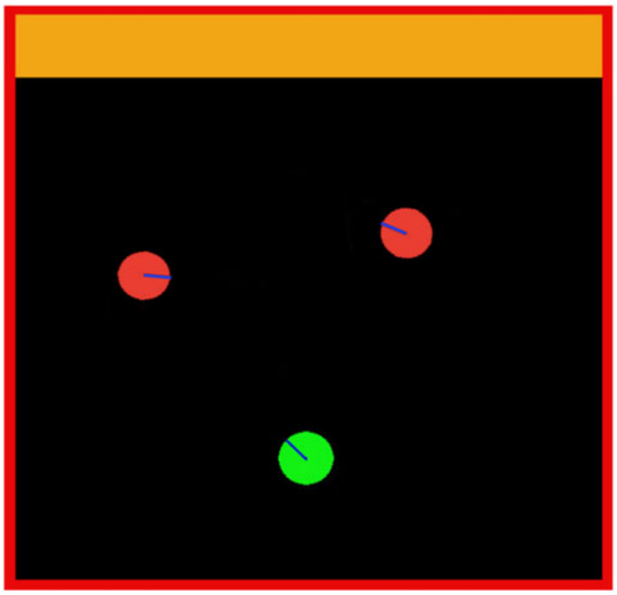


Fig. 4. A screen shoot of the game system for case studies.

Table 1. Agent actions

Action	v	ω
a_1	5	0.35
a_2	5	0.2
a_3	5	0.1
a_4	10	0
a_5	5	−0.1
a_6	5	−0.2
a_7	5	−0.35

Reward functions and knowledge capture

For tasks like ship collision avoidance, usually the reward function is composed of two major items. One is *ending reward*, or *terminal reward*, that rewards the agent when the game or episode is completed (hence *ending* or *terminal*). Typically, a large positive reward is given for success and a large negative reward for failure. The other reward item is *shaping reward*, which signifies how the agent will be rewarded during the process of the task execution. Typically, a small negative reward is given for every step of action or every unit time because staying in the game (or process) should be penalized, so that the agent will attempt to complete the task as soon as possible. Based on this consideration, the following *Typical Reward Function* is applied as a baseline design:

$$r^t = \begin{cases} 200, & \text{if reach goal} \\ -200, & \text{if hit obstacle}. \\ -1, & \text{else} \end{cases} \tag{8}$$

Since one of the research questions mentioned above relates to assessing how variations of reward function can influence the

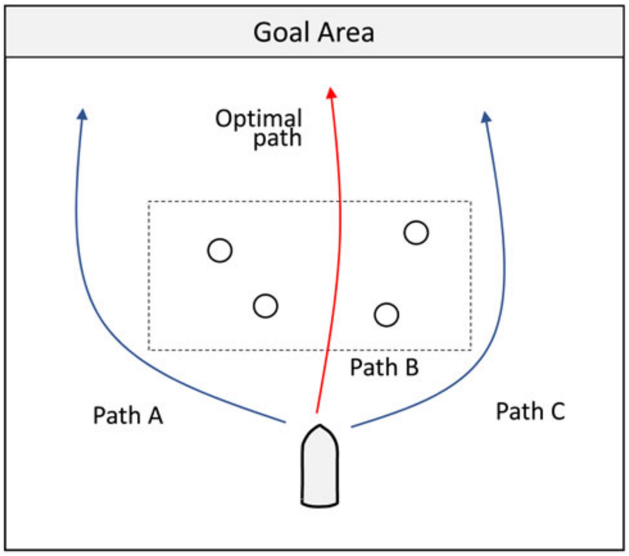


Fig. 5. Agent task performing behavior in collision avoidance.

agents' learning and performing behavior, we need to introduce *untypical* or *variable* reward functions. Because the agents' task performing behavior occurs during the process of task execution, rather than at the ending point of the process, the focus of varying reward functions can be put on the *shaping reward* item. More specifically, we attempt to shape the agents' behavior so that the efficiency of collision avoidance, as defined in the section "Collision avoidance at sea", can be increased.

Figure 5 illustrates an example of the efficiency difference between different task performing behaviors of a ship agent. Given the obstacles in the center area, the agent can go either the two side paths, A and C, or the center path, B. Although all three paths can lead to a successful run, the efficiency of path B is much better since it is the shortest path for reaching the goal.

In order for the agent to learn explicitly the most efficient behavior, for every step from start to end, the agent receives an additional reward, which is computed by a *goal reward* r_g and a *deviation reward* r_{dev}. Thus, at any time step t, the reward function is defined as the following *Enhanced Reward Function*:

$$r^t = \begin{cases} 200, & \text{if reach goal} \\ -200, & \text{if hit obstacle} \\ -(\omega_g \cdot r_g + \omega_{dev} \cdot r_{dev}), & \text{else} \end{cases}. \quad (9)$$

The goal reward, r_g, is the reward of moving toward goal direction, and the deviation reward, r_{dev}, is the negative reward of deviating from the goal direction. Currently, r_g and r_{dev} are computed by a linear function (10) of agent's current location coordinates, y and x_{dev}, where x_{dev} is the agent's deviation from the horizontal center of the window: $x_{dev} = \text{abs}(x/\text{width} - 1/2)$, and *width* is the horizontal length of the game window, k_g, b_g, k_{dev} and b_{dev} are constant coefficients. To make the learning more stable, r_g and r_{dev} are clipped between [−1, 1] and [0, 0.5], respectively. ω_g and ω_{dev} are the relative weight on r_g and r_{dev}:

$$\begin{cases} r_g = k_g y + b_g & \in [-1, 1] \\ r_{dev} = k_{dev} x_{dev} + b_{dev} & \in [0, 0.5] \end{cases}. \quad (10)$$

It is worth mentioning that the shaping reward designed this way can be used to "design" the agent's behaviors as will be discussed later.

Knowledge transfer: a belief-based approach

As mentioned above, RL has been successfully applied to capture domain knowledge for various tasks including the game of Go (Silver *et al.*, 2016, 2017). However, RL can be expensive in terms of the time for training and that for exploring various possible situations. Furthermore, for highly complex task situations, RL may not be able to converge at the desired level of total reward. To deal with these issues, transfer RL has been explored to expand the reach of RL.

Expert agent and student agent

In this research, we consider that two agents are involved in transfer RL, an *expert* agent and a *student* agent. The expert agent is first trained through ordinary RL with a relatively simple task, called *source task*, and becomes an "expert" in that task. The knowledge (i.e., the neural network) of the expert can then be transferred to the student agent that is completely know-nothing at the beginning but can be "taught" by the expert agent through (1) receiving part or all of the expert's knowledge as initialization so that the student holds part or all of the knowledge for the source task and (2) asking the expert agent for help from time to time when the student agent ventures to learning closely related but more complex new tasks, called *target tasks*. "What knowledge and how much" should be passed from the expert to the student, and "what help and how often" should be provided by the expert to the student are two important questions.

Belief-based transfer learning

The goal of transfer learning is to transfer the expert's knowledge to the student agent, so that the student can applied the expert's knowledge to engage in learning with the closely related but more complex tasks more effectively – that is, converge to a high level of total reward – and efficiently – that is, fast ramping-up and fast convergence. In this research, we consider three phases of transfer learning. First is "copy expert" knowledge (neural network) as the student's initial neural network, as shown in Figure 6. If the student agent plans to work only with the same tasks as the expert did, then "copy expert" alone will be sufficient. However, the student agent wants to engage in learning in the closely related but more complex new target tasks. Therefore, the second phase "consult expert" is needed in which the student occasionally asks the expert for suggested actions for given states. As the student's learning progresses, the expert's help decreases until being phased out completely, leading to the last phase, called "learn by self", where the student performs ordinary RL without further influence from the expert. Figure 6 shows an illustration of the three phases of the transfer RL together of the starting and ending points.

As shown in Figure 6, "consult expert" is the phase where the expert's "helping" or "coaching" happens. In this phase, we differentiate between three types of student actions based on the mechanism that produces the actions for the given state information.

a) Transfer action: With probability p_1 in Eq. (11a), the student agent seeks the help from the expert and selects the transfer action which is produced by directly using the expert neural

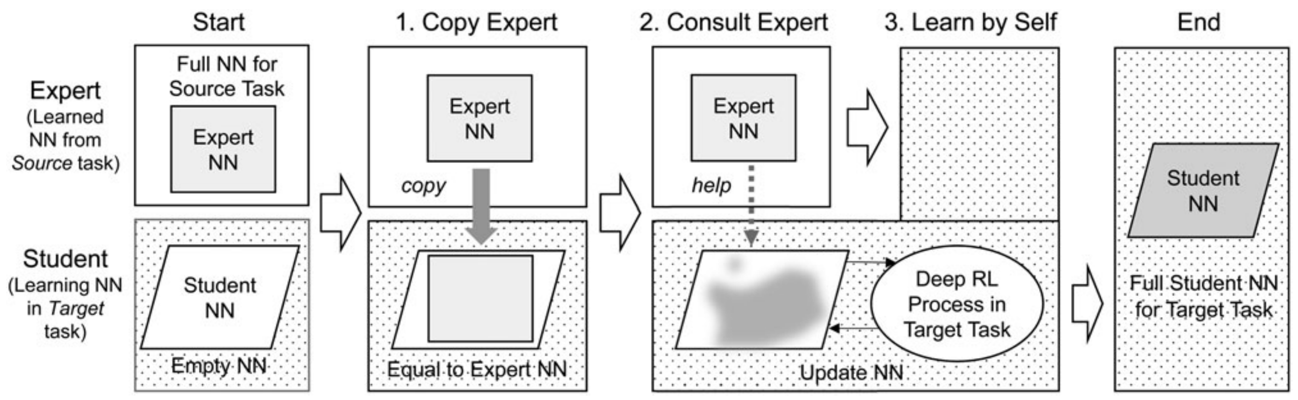


Fig. 6. Three phases of transfer RL.

network (see Fig. 6). T_{trans} is a hyperparameter, called transfer period, during which the student is "helped" by the expert, and the frequency of help decreases linearly from β_0 at the beginning to zero at T_{trans}. Thus, the transfer period T_{trans} equals the duration of the "consult expert" phase in Figure 6. β_0 is another hyperparameter, called initial transfer belief, which indicates how much confidence the student puts in the expert knowledge. The higher the confidence, the more often the student tends to ask the expert for help.

$$p_1 = \begin{cases} \beta_0\left(1 - \dfrac{t}{T_{\text{trans}}}\right), & t \leq T_{\text{trans}} \\ 0, & t > T_{\text{trans}} \end{cases} \tag{11a}$$

$$p_2 = \varepsilon(1 - p_1) \tag{11b}$$

$$p_3 = (1 - \varepsilon)(1 - p_1) \tag{11c}$$

$$p_1 + p_2 + p_3 = 1 \tag{11d}$$

b) *Exploration action*: With probability p_2 of Eq. (11b), the student agent selects a random action. In this research, an *ε-greedy* method is adopted for ordinary RL, in which the exploration probability usually decreases linearly from 1 to a fixed small number, say 0.01, over a period of time T_{expl}, $T_{expl} \geq T_{\text{trans}}$. In this research, in order to leave rooms for "transfer actions", the decreasing of the ε-value adopts a nonlinear concave shape rather than a linear shape. Therefore, the exploration probability decreases nonlinearly splitting a large portion from "exploitation" until T_{trans}. Our experiment results have shown that without such a large portion of exploration, the student agent will not learn about the new task effectively. The reason is partly that the initial student network is the same as the transfer network, so the "exploitation" is the same as, or similar to "transfer". Too little exploration will cause insufficient expansion to try the new tasks, leading to ineffective learning.

c) *Exploitation action*: With probability p_3 of Eq. (11c), the student agent selects the current *best* action produced by its own learned neural network, as in the ordinary RL. It should be noted that the expert network (the square NN in Fig. 6) is not updated during the student training process. Only the student network (the trapezoid NN in Fig. 6) is updated in the target task domain. As the learning converges and transfer period T_{trans} passes, the expert network will be completely phased out.

Case study design

As indicated in Figure 6, there are different learning paths for a student agent to move from the "start" to the "end". First, the student can learn from scratch by moving from "start" directly to phase 3 "learn by self". Second, the student can select phase 1 "copy expert" and then move to phase 3 "learn by self". Lastly, the student agent can take the whole course of "copy expert", "consult expert", and then "learn by self". In the following, these three leaning paths are called L-Path1, L-Path2, and L-Path3, respectively.

Table 2 categorizes the case studies that are performed in this research including the cases, the parameters, and used parameter values. The reward function exploration is carried out only for L-Path1, since mixing the knowledge transfer may confound the effect of reward function. Both L-Path2 and L-Path3 are explored to assess the impact of pure "copy expert" and the inclusion of "consult expert". The cases shown in Table 2 are carried out against various tasks and task transfer situations, which are discussed in detail in the following section "Tasks for training and transfer".

Training parameters

As mentioned in the section "Agent learning behavior", an end-to-end RL approach (Mnih *et al.*, 2013) is employed in this research. The game window images, such as the one shown in Figure 4, are captured as input state information. The size of the game window is 400 × 400 pixels. During preprocessing, the window is first scaled down to 84 × 84 pixels, and then converted to grayscale since the color should not influence the agent's decision-making. At each step, the agent stacks 4 previous frames together, which are then fed into the neural network. The network structure is the same as the original DQN paper (Mnih *et al.*, 2013) with 84 × 84 pixels input and an output of 7 actions. The experience replay size is 50,000. At every step, a mini-batch of size 32 is randomly sampled from the experience replay.

All the case studies were trained using Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.0001. The discount factor γ is set to be 0.99. The agent follows ϵ-greedy policy, with ϵ being annealed down from 1 to 0.01 during the first 1

Table 2. Cases, parameters, and their values

	Knowledge capture from scratch (L-Path1)	Knowledge capture with transfer from expert		
		Copy expert (L-Path2)	Copy expert + Consult expert (L-Path3)	
			Transfer duration	Belief level
Typical reward function	Baseline	NNs(t_0) = NNe(t_0) (Baseline for KT)	T_{trans} (150 K/300 K/700 K/1 M)	β_0 (0/0.5/1.0/1.5/2.0)
Enhanced reward function	Change ω_{dev} (0/0.5/1.0/1.5/2.0)	X		

Table 3. System training parameters

System design parameters	Parameter values	Remarks
Experience replay size	50,000	N = 50 K in $D = \{e_1, e_2, \ldots, e_N\}$, where $e_t = (s_t, a_t, r_t, s_{t+1})$
Mini-batch size	32	$Be_t = \{e_{t1}, e_{t2}, \ldots, e_{t32}\}$, e_{ti} $(1 \leq i \leq 32)$ is randomly selected
Discount factor γ	0.99	See Eq. (4)
Learning rate	0.0001	
Total training episodes	50,000	One complete training = 50 K episodes
Annealing frames	1 million	1 M frames for ϵ to decreases from 1 to 0.01
ϵ	$1 \rightarrow 0.01$	Linearly decrease from 1 to 0.01 over T_{expl}
ϵ_T	$\beta_0 \rightarrow 0.01$	Concavely decrease from β_0 to 0.0 over T_{tran}
Goal weight ω_{g}	1	See Eq. (9)
Deviation weight ω_{dev}	0/0.5/1.0/1.5/2.0	Values explored; see Eq. (9)
Transfer duration (frames) T_{trans}	150 K/300 K/700 K/1 M	Frames explored; see Eq. (11a)
Initial transfer belief β_0	0.1/0.3/0.5/0.9	Values explored; see Eq. (11a)

million frames. The weight on goal reward ω_{g} is 1. The weight on deviation reward ω_{dev} can be 0, 0.5, 1.0, 1.5, or 2.0 for this study. Table 3 shows major system parameters and their values of the case studies.

Tasks for training and transfer

In this case study, four different tasks are considered, as shown in Figure 7. Task A is a *simple* task, in which there is only one static obstacle in a small restricted area. The position of the obstacle is randomly set within the restricted area for each episode during training.

Task B is more complex than Task A in that there are two static obstacles randomly generated for each episode within a larger area. Task C and Task D involve moving obstacles (i.e., target ships) with the constant moving speed at 20 pixels per time step. Again, the initial position and moving direction of the target ships in Task C and Task D are set randomly for each episode.

Task B is used for reward function study, in which both *typical reward function* and *enhanced reward function* are tried, and their training results compared.

For the knowledge transfer studies, Task A is used as the *source task*, and the rest of the tasks are used as *target tasks*. To assess the complexity of each task and the similarity level between the tasks, we make the following assumptions.

1. The *own ship* is a small ship with desirable movability. Hence, for all tasks, the vehicle dynamics VD = 1.0.
2. The number of obstacles and/or target ships is either 1 (NO = 1.0) or 2 (NO = 2.0) depending on how many of them are involved.
3. For stationary obstacles, the object dynamics is 1 (OD = 1.0); for moving target ships, it is 2 (OD = 2.0).
4. Given that moving target ships are much more difficult to avoid and the own ship vehicle dynamics is minimum (for small ships), we set up the weight coefficients as follows: $\lambda_1 = 3$, $\lambda_2 = 1.5$, and $\lambda_3 = 1$.

Based on the above, we have the complexity vector for each task in Figure 7 represented as follows: $\overrightarrow{\text{complexity}}_{\text{A}} = (1.0, 1.0, 1.0)$, $\overrightarrow{\text{complexity}}_{\text{B}} = (1.0, 2.0, 1.0)$, $\overrightarrow{\text{complexity}}_{\text{C}} = (2.0, 1.0, 1.0)$, and $\overrightarrow{\text{complexity}}_{\text{D}} = (2.0, 2.0, 2.0)$. Based on these vectors and the Eqs (1) and (3), the complexity values of each task and the similarity values between the tasks can be calculated, as shown in Table 4.

Results and discussion

In this research, four case studies are performed to investigate the RL based knowledge capture. They are *reward function* for behavior design, *copy expert* for knowledge transfer, *adjust transfer period,* and *adjust transfer belief* for knowledge transfer. In the following, we first discuss the results of reward function study and then provide details of how knowledge transfer strategies interact with the tasks of different similarities.

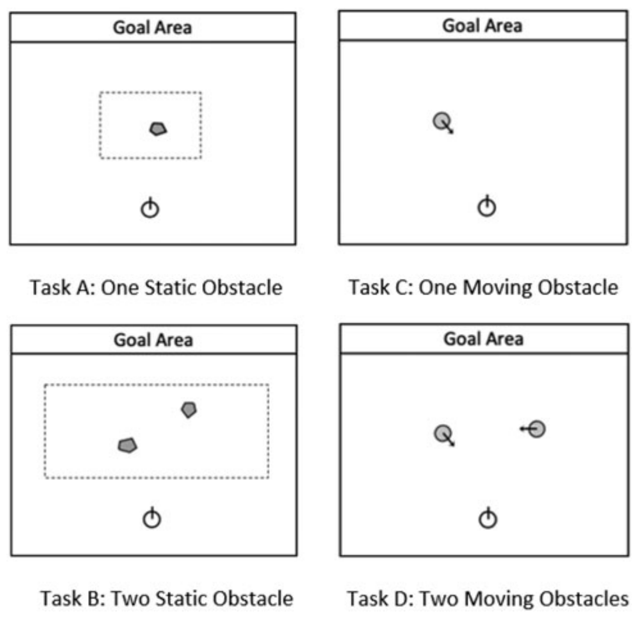

Goal Area
Goal Area
Task A: One Static Obstacle
Task C: One Moving Obstacle
Goal Area
Goal Area
Task B: Two Static Obstacle
Task D: Two Moving Obstacles

Fig. 7. Collision avoidance Tasks A through D used for case studies.

Table 4. Task complexities and similarities

	Task A	Task B	Task C	Task D
$\overrightarrow{\text{complexity}}$	2.345	3.162	3.808	4.359
Task A	1.000			
Task B	0.449	1.000		
Task C	0.366	0.320	1.000	
Task D	0.320	0.366	0.499	1.000

Reward functions for designing system behavior

In this study, Task B in Figure 7 is used as the learning task and L-Path2 is used for agent training. The ship agent starts from scratch and captures collision avoidance knowledge through ordinary RL (i.e., there is no knowledge transfer involved). The goal of this study is to explore the effect of applying the *enhanced reward function* of Eqs (9) and (10) and investigate the behavioral response of the ship agent to the changing reward function variables. The *typical reward function* [Eq. (8)] is used as a baseline case. Training of ship agents is carried out with varying values of the reward function parameter: *deviation weight* ω_{dev}. Five values of ω_{dev} were tried: 0, 0.5, 1.0, 1.5, and 2.0. The agent's learning speed and the rate of the successful runs (see "Collision avoidance at sea" section) are used as the performance measurements.

Figure 8 shows the average reward of each case. The x-axis is the number of episodes into the training process, and the y-axis is the average reward. The different colored lines correspond to different deviation weights. The deviation weight $\omega_{\text{dev}} = 0$, that is, Eq. (8) is used as reward function, is treated as the baseline case.

From Figure 8, it can be seen that adding a small (negative) deviation reward can boost learning speed. The reward plots of $\omega_{\text{dev}} = 0.5$, 1.0, and 1.5 (orange, green, and red curves, respectively) ramp up much faster than the baseline. At 100×100 episodes, it is already more than three times (3×) better, and at 150×100, about one and half times (1.5×) better.

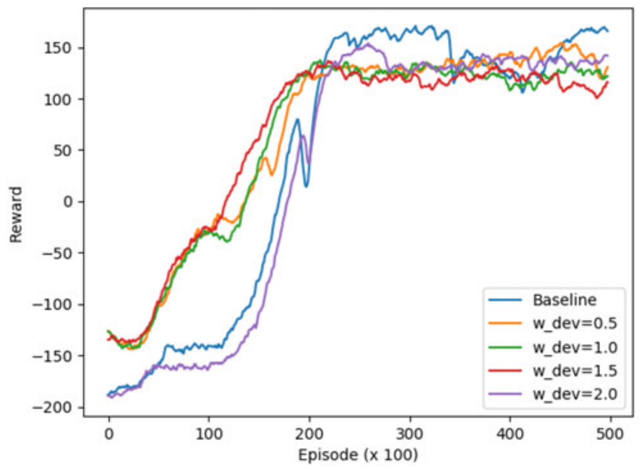


Fig. 8. Learning performance – average reward for different ω_{dev} values.

Since the shaping rewards in these cases are all different, it is unreasonable to compare the agent's average reward after the convergence. In fact, they all converge to a similar range of reward (100–150).

To further evaluate the agent's learned behavior in different cases, 500 random tests were carried out for each ω_{dev} setting, where the success rate and the average time of reaching the goal were recorded.

Figure 9 summarizes the average time of reaching the goal and the success rate of different cases with varying deviation weights. The baseline uses the *typical reward function*. All the statistics of average goal reaching time have been converted to relative percentage of the baseline. As can be seen, after adding deviation reward (penalty) to the reward function, the agent saves 5%, 14%, 11%, and 11% of time, as well as energy, in cases where $\omega_{\text{dev}} = 0.5$, 1.0, 1.5, and 2.0, respectively. When $\omega_{\text{dev}} = 1.0$ the goal reaching time is the best, only 86% of the baseline time. However, in all these cases, the success rate drops slightly. When ω_{dev} keeps increasing to 1.5 or 2.0, the success rate drops to 0.91 and 0.89.

The simulation results of the baseline case (typical reward function) and various reward tuning cases (enhanced reward function with varying ω_{dev}) have demonstrated the impact of reward functions on the agent's learning and working behavior. Such an impact can be very sensitive to small changes in the values of the reward function parameters. Following are some insights drawn from the results.

Ending reward and interim reward: The baseline is based on the *typical reward function* in Eq. (8). The major feature is that it has two important step-function like "ending-rewards", one for reaching the goal +200, and the other for hitting an obstacle −200, signifying two opposite endings of the episode. The "−1" in (8) and the added $-(\omega_g \cdot r_g + \omega_{\text{dev}} \cdot r_{\text{dev}})$ in (9) are "interim awards" that are devised to guide the way for the agent to reach its ending point. A close look at the work behavior of the agent trained based on Eq. (8) reveals that the agent always tries to avoid the whole "congested" water area and takes the side ways to reach the goal no matter how the obstacles in the area are positioned, as shown as Path A and Path C in Figure 5. By adding the penalty for horizontal deviation $\omega_{\text{dev}} \cdot r_{\text{dev}}$ the agent's behavior changed: it seeks both the middle opportunities, for example, Path B, as well as the safer sideways, Path A and Path C. The

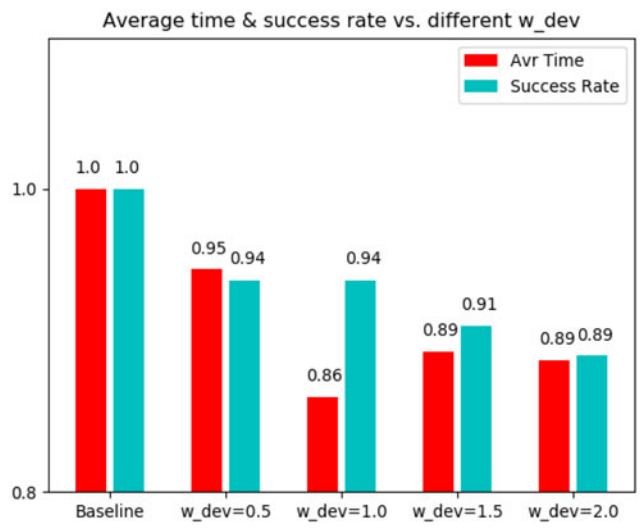


Fig. 9. Task performance – goal reach time and success rate for different ω_{dev} values.

added negative reward encouraged the agent to explore more the middle ways despite the collision risks. The result is that the average goal reaching time is shortened, but the success rate suffers a bit too. The interim reward provides an effective means to "design" the agent's behavior and embedding the rules, regulations and heuristic knowledge.

Sparse reward versus shaped reward: The reward functions with only "ending reward" items are sparse. Although the baseline function Eq. (8) had "−1" as its interim reward, it is sparser compared to function (9) which has more interim items involved. Sparse rewards slow down learning because the agent needs to take many actions before getting any significant reward. To avoid sparse rewards, the interim reward items are used to create shaped reward function (hence also called *shaping rewards*). As shown in Figure 8, shaped reward function (9) with more shaping items has led to better learning speed, thanks to the reward gradient created by the shaping items. When the gradient's effect expands to the point where *the reward field is distorted*, the success rate falls significantly through a threshold 0.9 (as shown in Fig. 9), the learning speed drops (Fig. 8 with $\omega_{\text{dev}} = 2.0$). One can set an upper limit for shaping the reward function that is determined by threshold 0.9 (or another desired number) of the success rate.

Copy expert for knowledge transfer

When the given domain task is too complex and the sensory information limited, RL may become ineffective and/or inefficient for knowledge capture. In this case study, we investigate how directly copying the knowledge (i.e., neural network) from the expert agent who learned from simpler tasks can help the student agent learn in the closely related but more complex tasks. It is assumed that the expert agent has been fully trained using the source Task A with *complexity* (A) = 2.345 (Fig. 7 and Table 3), and L-Path2 (Table 2) is employed for student training.

In this study, the first *target task* is Task B: *complexity* (B) = 3.126, *similarity* (A,B) = 0.449. This transfer case can be considered as a "high similarity" case. The second *target task* is Task D: *complexity* (D) = 4.359, *similarity* (A,D) = 0.320. Comparing with Task B, this case can be considered as a "low similarity" case. For the purpose of comparison, two "*bootstrap*" scenarios are introduced, in which the student agent learns from scratch in the target tasks B and D without receiving any information from the expert agent.

Figure 10 illustrates the results of this case study, with Figure 10a showing the effect of "copy expert" for a *high similarity* case and Figure 10b for a *low similarity* one. The plots are all the average values of 10 simulation runs. The spreads in Figure 10b show the large deviation of the 10 simulations runs due to the higher task complexity and lower task similarity. Deviations for simpler tasks A and B are relatively small and are not plotted.

As shown in Figure 10, "copy expert" helped the student learn much faster compared with "bootstrap". When the target task similarity is high (Fig. 10a), the "bootstrap" learner can eventually reach the desired reward level, but "copy expert" transfer improved the learning efficiency significantly. When the similarity is low (Fig. 10b), however, the "bootstrap" learner can hardly reach a desired level of reward or it may take endless time to do so. In this case, "copy expert" played a significant role to increase the learning effectiveness and efficiency for the student agent. While the learning variance is relatively low for low complexity and high similarity target tasks (Liu and Jin, 2018), significant learning variance can be seen in high complexity and low similarity tasks, meaning many student agents could not achieve a higher reward score, especially for the bootstrap agents, as shown in Figure 10b.

The results in Figure 10 indicate that "copy expert" can help increase the learning efficiency and the improvement can be significant for low similarity tasks. However, when the target task becomes more complex and the similarity is low, "copy expert" alone may not be effective enough in guiding the learning process of student agents, as shown in Figure 10b. In the next two case studies, we explore whether and how adding *a transfer period* to "copy expert" may help learning in more complex and unsimilar target tasks.

Adjust transfer period

In this study, the belief-based transfer RL approach described in the section "Knowledge transfer: a belief-based approach" is applied to train the student agents through L-Path3 (Table 2). Both Task B and Task D are used as target tasks, so that both high similarity and low similarity tasks are covered. The goal here is to investigate how changing transfer duration length may influence the student agent's learning behavior while keeping the transfer belief at $\beta_0 = 0.9$. Since the student agent takes learning path L-Path3, it has already copied the neural network from the expert agent that was trained in source task Task A.

Figure 11 illustrates the learning performance of varying transfer period from 150 K to 300 K, 700 K, and 1 M frames with yellow, blue, green, and pink colors, respectively. The target task is Task B, a high similarity task. As described in the section "Knowledge transfer: a belief-based approach", shorter transfer period T_{tran} means the shorter period of *expert help* (Fig. 6). From a learning speed point of view, the results in Figure 11 show that longer transfer periods lead to better learning performance, with the effect diminishing as it becomes excessively long (after 700 K frames). When the transfer period is getting closer to 1 million frames, which is also the annealing time of ε decreasing to 0.1, the performance decreases. Comparing with the "copy expert" case (the red line in Fig. 11), the positive impact of *expert help* is considerably large, especially until the 20 K episodes range.

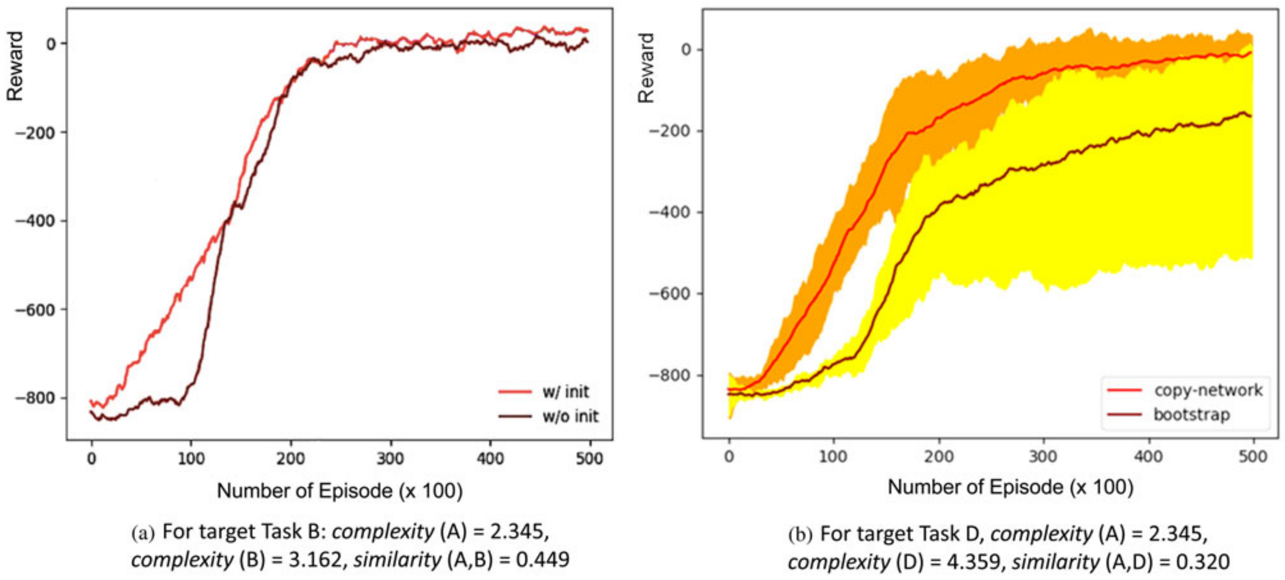


Fig. 10. Comparison of "copy expert" transfer learning and "bootstrap" learning from scratch.

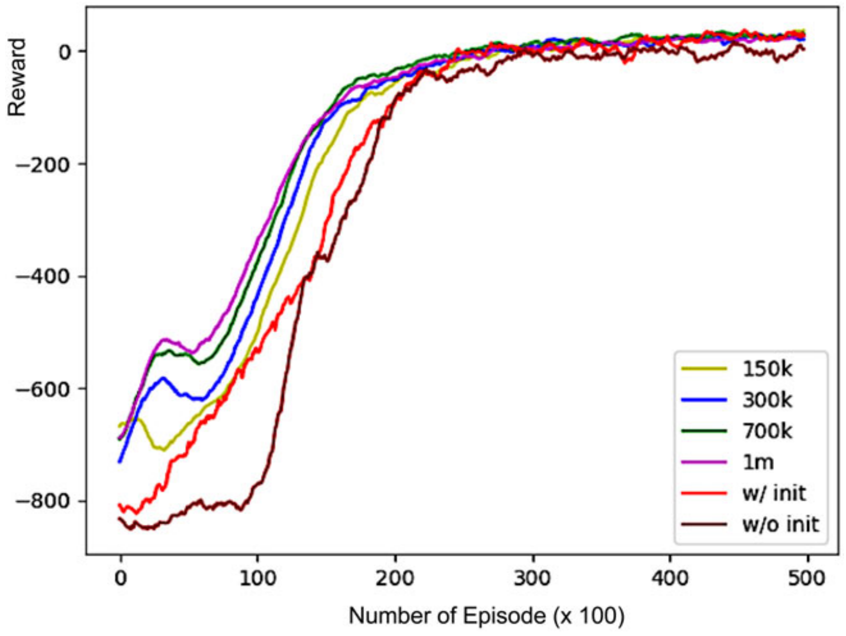


Fig. 11. For target Task B (high similarity): average learning performance.

Since the transfer period of 700 K turned out as the best arrangement for transfer learning in the similar target task situations, we applied the same parameter values $\beta_0 = 0.9$ and $T_{\text{trans}} =$ 700 K to target Task D, a low similarity task. Figure 12a shows the results of the simulation runs, where the baseline case is in blue average and green spread and is based on learning path L-Path2 "copy expert and then learn by self", while the transfer learning is in red average and brown spread. Unlike the high similarity cases, the effect of adding "transfer period" did not have much effect on the learning behavior of the student agent.

For $T_{\text{trans}} = 700$ K, during the early stage, the performance is better than the baseline. However, after the transfer period, the learning variance starts to grow. Although the maximum performance (the upper edge of the brown spread) is still better than baseline, many students perform worse than the baseline (lower edge of the brown spread) due to the high level of variance of learning. The average performance is slightly higher than baseline but can hardly be considered as improvement.

For $T_{\text{trans}} = 300$ K, again, during the early stage, the performance is better than the baseline. However, because the transfer period is rather short, the learning speed did not pick up after the transfer period. The learning variance also starts to grow. Although the maximum performance is still better than baseline (shown as the upper edge of the orange spread), many student agents perform worse than the baseline, as indicated by the lower part of the orange spread. The average performance is slightly lower than baseline but not by a large amount. To a certain extent, the short transfer period with high belief level interfered the normal learning and led to the inferior performance due to the dissimilarity of the target task.

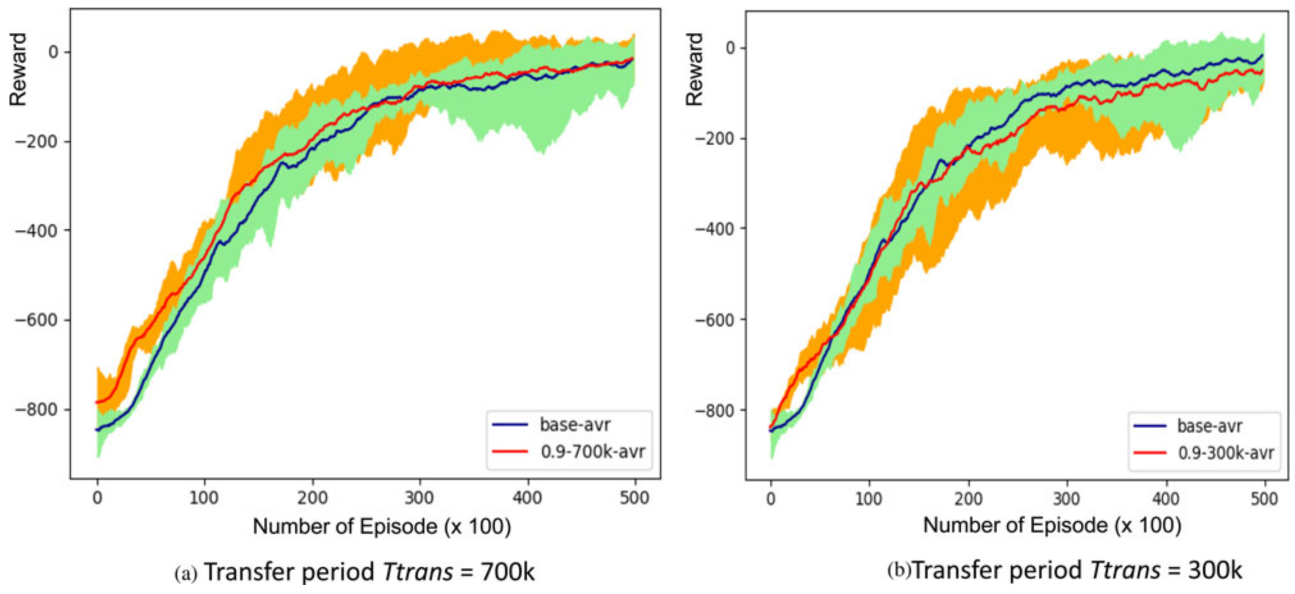


Fig. 12. Performance learning for target Task D, low similarity, with $\beta_0 = 0.9$.

Adjust transfer belief level

Because the student transfer learning in more complex and low similarity target tasks is not sensitive to the change of transfer period in the last case study (given $\beta_0 = 0.9$), we turn to varying the transfer belief β_0 in this case study. The target task here is Task D. The transfer period T_{trans} is fixed at 700 K, and the baseline case is the same as the one described in the last case. The initial transfer belief is varied from 0.9 to 0.5, 0.3, and 0.1.

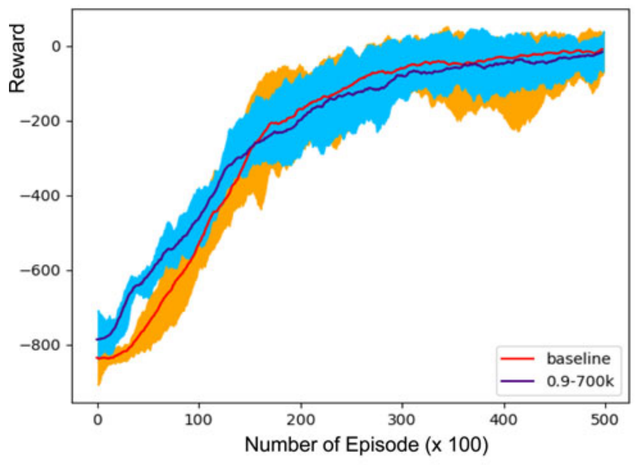
Figure 13 illustrates the results of simulation runs of varying initial transfer beliefs, with β_0 being set to 0.9, 0.5, 0.3, and 0.1, shown in Figure 13a–d, respectively. β_0 measures the initial probability of the student agent picking transfer action suggested by the expert network (this probability is linearly decreasing to 0 at the end of transfer period). As shown in Figure 13b–d, the jump-start effect of lower initial transfer belief settings is less obvious compared to $\beta_0 = 0.9$. Additionally, when $\beta_0 = 0.5$, as shown in Figure 13b, many students perform much better than the baseline after the transfer period, and the learning variance is very low. This pattern is unique and cannot be found in other transfer beliefs (i.e., $\beta_0 = 0.9$, 0.3, and 0.1). Another interesting pattern is that higher transfer belief ($\beta_0 = 0.9$) helps the student perform much better in the early learning stage for frequently trying transfer actions; however, this does not guarantee a better performance in the later stage. As can be seen in Figure 13a, the average convergence time of $\beta_0 = 0.9$ is almost the same as the baseline. On the other hand, though the early stage performance is almost overlapping with the baseline in the cases with smaller transfer beliefs, as time proceeds, the learning starts to differentiate from the baseline. It appears that the value of the initial belief level β_0 has a profound impact on the later part, rather than start-up, of students' learning process.

Conclusions

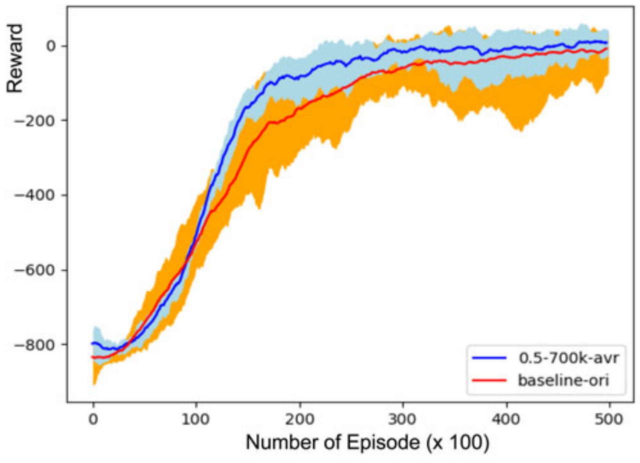
Both robotic research and transportation industries have dealt with collision avoidance problems for years. In addition to system control methods, the intelligent collision avoidance support is often needed for unforeseeable situations where human-like intelligence is demanded. Recent progress in machine learning, especially RL, has made it possible to train agents to acquire collision avoidance knowledge in the way closely as humans do. The knowledge capture based on RL is still domain and task-dependent. A better understanding is needed on how to shape agents' learning and task performance, and how to make use of the machine knowledge that has already been acquired through previous machine learning processes.

In this research, both the design of reward functions and the use of previously learned knowledge are investigated in the context of knowledge capture. The exploration of the reward functions revealed the trade-offs between types of shaping functions, and our proposed belief-based transfer RL approach provided useful design parameters to transfer experts' knowledge in different target task situations. The following are some conclusions.

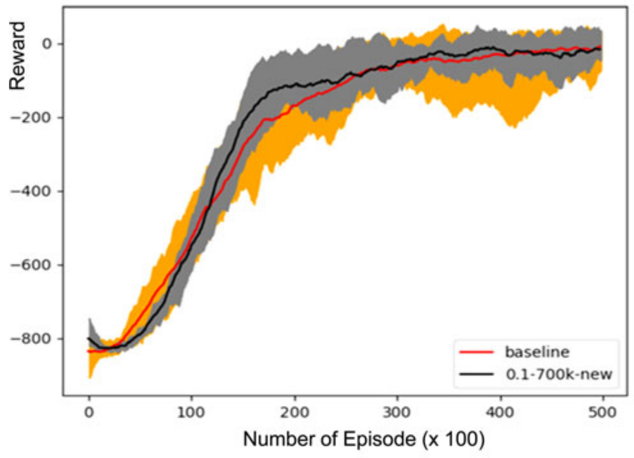
- Reward function is a useful instrument for designing agent behaviors. It is important to understand the effect of *ending/terminal rewards* and *interim/shaping rewards*. Designing interim rewards is an effective way to shape the agent's behavior, provided the sparsity and shape of the reward function are recognized and the trade-off between the performance measures clarified, since unexpected effects may happen.
- "Copy expert" can be a very effective strategy for multi-step learning in approaching complex RL tasks. For high similar target tasks, copy expert can significantly improve the initial learning stage and provide needed jump-start of learning. For more complex and low similarity cases, copy expert can significantly reduce the learning deviation and orient the learning process into a right direction reaching to a much higher level of reward.
- "Consult expert" by adding a transfer period to copy expert can be very helpful for the student agent in transfer learning. For high similarity target tasks, longer transfer period tends to be more effective in ramping learning early, and there appears to be an optimal length of the transfer period, beyond which the effect diminishes and turns negative. For more complex target



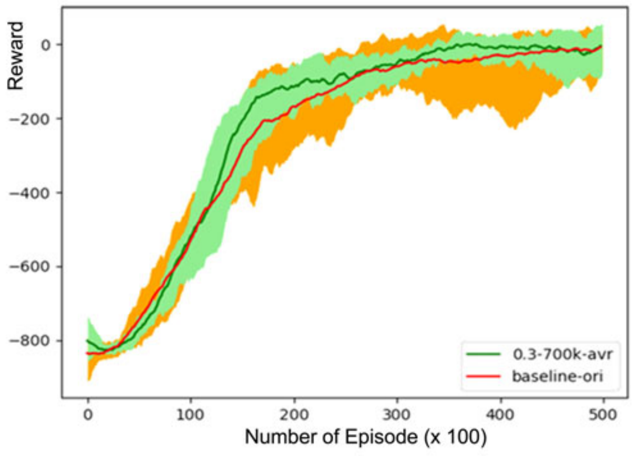
(a) $\beta_0 = 0.9$, $T_{trans} = 700k$



(b) $\beta_0 = 0.5$, $T_{trans} = 700k$



(d) $\beta_0 = 0.1$, $T_{trans} = 700k$



(c) $\beta_0 = 0.3$, $T_{trans} = 700k$

Fig. 13. Learning performance in response to changing initial beliefs for low similarity tasks.

tasks with low similarity, consult expert does not have a significant effect in providing a further improvement on top of copy expert.

- For more complex and low similarity target tasks, adjusting initial transfer belief is a useful way to gain further improvement. While higher transfer belief level helps high similarity target tasks, for low similarity target tasks, lower belief level works better since the expert's old knowledge is less relevant in the new target task. Instead of having a "jump-start" effect, reducing the level of transfer belief helps accelerate learning at a later time after the transfer period is passed. There appears to be a best value of transfer belief where the learning is most effective, and the variation the least.

It should be noted that the complexity and similarity measures, the simulation results and the insights discussed above are limited to the types of the tasks explored in this research. Further research is needed in order to draw more general conclusions. Our ongoing research attempts to expand the generality of this work by exploring more design parameters and more case studies. Another direction along this research line is to explore multi-agent transfer reinforcement learning in the domains of autonomous collision avoidance for ships, cars, and industrial robots.

Acknowledgments

The authors are grateful to the Monohakobi Research Institute (MTI) team, led by Dr. Hideyuki Ando, for their support and collaboration on this research. Thanks are also extended to Mr. Edwin Williams for his valuable comments and suggestions during the course of this research.

Funding. This paper was based on the work supported by MTI and Nippon Yusen Kaisha (NYK). The authors are grateful to MTI and NYK for their kind support.

References

Alonso-Mora J, Breitenmoser A, Rufli M, Beardsley P and Siegwart R (2013) Optimal reciprocal collision avoidance for multiple non-holonomic robots. In Martinoli A. et al. (eds) *Distributed Autonomous Robotic Systems*. Berlin, Heidelberg: Springer, pp. 203–216.

Arnold A, Nallapati R and Cohen W (2007) A comparative study of methods for transductive transfer learning. *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, 31 March 2008. Omaha, NE, USA: IEEE Computer Society.

Bahadori M, Liu Y and Zhang D (2014) A general framework for scalable transductive transfer learning. *Knowledge and Information Systems* **38**, 61–83.

Bertsekas DP and Tsitsiklis JN (1996) *Neuro-Dynamic Programming*. MIT Press.

Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, Zhang X, Zhao J and Zieba K (2016) End to end learning for self-driving cars. *arXiv*: 1604.07316 [cs.LG].

Brunn P (1996) Robot collision avoidance. *Industrial Robot: An International Journal* **23**, 27–33.

Casanova D, Tardioli C and Lemaître A (2014) Space debris collision avoidance using a three-filter sequence. *Monthly Notices of the Royal Astronomical Society* **442**, 3235–3242.

Chen JX (2016) The evolution of computing: AlphaGo. *Computing in Science & Engineering* **18**, 4–7.

Chen YF, Liu M, Everett M and How JP (2016) Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. *arXiv*: 1609.07845 [cs.MA].

Churchland PS and Sejnowski TJ (2016) *The Computational Brain*. Chambridge, MA, USA: MIT Press.

Coates A, Huval B, Wang T, Wu D and Ng A (2013) Deep learning with COTS HPC systems. *Proceedings of the 30th International Conference on Machine Learning*. PMLR, Vol. **28**. pp. 1337–1345.

Dean J, Corrado G, Monga R, Kai C, Devin M, Mao M, Ranzato M, Senior A, Tucker P, Yang K, Le QV and Ng AY (2012) Large scale distributed deep networks. NIPS'12: Proceedings of the 25th *International Conference on Neural Information Processing Systems*, Vol. **1**. Red Hook, NY, USA: Curran Associates Inc.

Dieleman S and Schrauwen B (2014) End-to-end learning for music audio. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, New York, NY USA.

Ding Z, Nasrabadi N and Fu Y (2016) Task-driven deep transfer learning for image classification. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 4–9 May 2014. Florence, Italy: IEEE.

Eleftheria E, Apostolos P and Markos V (2016) Statistical analysis of ship accidents and review of safety level. *Safety Science* **85**, 282–292.

Fahimi F, Nataraj C and Ashrafiuon H (2009) Real-time obstacle avoidance for multiple mobile robots. *Robotica* **27**, 189–198.

Fernandez F and Veloso M (2006) Probabilistic policy reuse in a reinforcement learning agent. *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Vol. **58**, 8–12 May 2006. Hakodate, Japan, pp. 720–727.

Ford JWW (2009) *A Seaman's Guide to the Rule of the Road*. Gloucestershire, UK: Morgans Technical Books Limited.

Goerlandt F and Kujala P (2014) On the reliability and validity of ship–ship collision risk analysis in light of different perspectives on risk. *Safety Science* **62**, 348–365.

Hameed S and Hasan O (2016) Towards autonomous collision avoidance in surgical robots using image segmentation and genetic algorithms. *2016 IEEE Region 10 Symposium (TENSYMP)*, 9–11 May 2016. Bali, Indonesia: IEEE, pp. 266–270.

Hinton G, Vinyals O and Dean J (2015) Distilling the Knowledge in a Neural Network. *arXiv*. 1503.02531v1 [stat.ML] 9 Mar.

Hourtash AM, Hingwe P, Schena BM and Devengenzo RL (2016) *U.S. Patent No. 9,492,235*. Washington, DC: U.S. Patent and Trademark Office.

Jin Y and Koyama T (1987) On the design of marine traffic control system (1st report). *Journal of the Society of Naval Architects of Japan* **162**, 183–192.

Keller J, Thakur D, Gallier J and Kumar V (2016) Obstacle avoidance and path intersection validation for UAS: a B-spline approach. *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. Arlington, VA, USA: IEEE, pp. 420–429.

Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research* **5**(1), 90–98.

Kingma DP and Ba J (2015) Adam: A method for stochastic optimization, in Proceedings of ICLR, 2015.

Krizhevsky A, Sutskever I and Hinton G (2012) ImageNet classification with deep convolutional neural networks. *Communications of the ACM* **60** (6), 84–90.

Kuderer M, Gulati S and Burgard W (2015) Learning driving styles for autonomous vehicles from demonstration. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 26–30 May 2015. Seattle, WA, USA: IEEE, pp. 2641–2646.

Le Q, Ranzato M, Monga R, Devin M, Chen K, Corrado G, Dean J and Ng A (2012) Building high-level features using large scale unsupervised learning. *International Conference on Machine Learning*: *arXiv*: 1112.6209v5 [cs.LG].

LeCun Y, Bengio Y and Hinton G (2015) Deep learning. *Nature* **521**, 436–444.

Liu X and Jin Y (2018) Design of transfer reinforcement learning under low task similarity. *ASME 2018 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC2018-86013*, 26–29 August 2018. Quebec City, Quebec, Canada: American Society of Mechanical Engineers Digital Collection.

Machado T, Malheiro T, Monteiro S, Erlhagen W and Bicho E (2016) Multi-constrained joint transportation tasks by teams of autonomous mobile robots using a dynamical systems approach. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 16–21 May 2016. Stockholm, Sweden: IEEE, pp. 3111–3117.

Mastellone S, Stipanovic D, Graunke C, Intlekofer K and Spong M (2008) Formation control and collision avoidance for multi-agent non-holonomic systems: theory and experiments. *The International Journal of Robotics Research* **27**, 107–126.

Matarić MJ (1997) Reinforcement learning in the multi-robot domain. In Arkin RC and Bekey GA (eds) *Robot Colonies*. Boston, MA, USA: Springer, pp. 73–83.

Mericli C, Mericli T and Akin HL (2010) A reward function generation method using genetic algorithms: a robot soccer case study (extended abstract). *Proceeding of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, Vol. **1–3**, 10–14 May 2010, Toronto, Canada.

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D and Riedmiller M (2013) Playing Atari with deep reinforcement learning. *arXiv*:1312.5602v1 [cs.LG].

Mukhtar A, Xia L and Tang TB (2015) Vehicle detection techniques for collision avoidance systems: a review. *IEEE Transactions on Intelligent Transportation Systems* **16**, 2318–2338.

Ng AY and Russell S (2000) Algorithms for inversereinforcement learning, in Proceedings of ICML 2000.

Ohn-Bar E and Trivedi MM (2016) Looking at humans in the age of self-driving and highly automated vehicles. *IEEE Transactions on Intelligent Vehicles* **1**, 90–104.

Pan SJ and Yang Q (2010) A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* **22**, 1345–1359.

Parisotto E, Ba J and Salakhutdinov R (2016) Actor-mimic: deep multitask and transfer reinforcement learning. *arXiv*:1511.06342v4 [cs.LG] 22 Feb 2016.

Schaul T, Quan J, Antonoglou I and Silver D (2016) Prioritized experience replay. *arXiv*:1511.05952v4 [cs.LG] 25 Feb 2016.

Shiomi M, Zanlungo F, Hayashi K and Kanda T (2014) Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. *International Journal of Social Robotics* **6**, 443–455.

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T and Hassabis D (2017) Mastering the game of Go without human knowledge. *Nature* **550**, 354–359.

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T and Hassabis D (2016) Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484

Sutton RS and Barto AG (2018) *Reinforcement Learning: An Introduction*. Cambridge MA, USA: MIT Press.

Szepesvari C (2010) *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers.

Tang S and Kumar V (2015) A complete algorithm for generating safe trajectories for multi-robot teams. In: Bicchi A., Burgard W. (eds) *Robotics Research. Springer Proceedings in Advanced Robotics*, vol 3. New York, NY, USA: Springer, pp 599–616.

Taylor M and Stone P (2007) Cross-domain transfer for reinforcement learning. *ICML '07: Proceedings of the 24th International Conference on Machine Learning*. ACM, pp. 879–886. June 2007, Corvalis, OR, USA

Torrey L, Shavlik J, Walker T and Maclin R (2006) Skill acquisition via transfer learning and advice taking. *European Conference on Machine Learning*. Berlin, Heidelberg: Springer, pp. 425–436.

van Hasselt H, Guez A and Silver D (2015) Deep reinforcement learning with double Q-learning. *arXiv*:1509.06461v3 [cs.LG].

Wang FY, Zhang JJ, Zheng X, Wang X, Yuan Y, Dai X, Zhang J and Yang L (2016*a*) Where does AlphaGo go: from Church-Turing thesis to AlphaGo thesis and beyond. *IEEE/CAA Journal of Automatica Sinica* **3**, 113–120.

Wang Z, School T, Hessel M, van Haselt H, Lanctot M and de Freitas N (2016*b*) Dueling network architectures for deep reinforcement learning. *arXiv*:1511.06581v3 [cs.LG] 5 Apr.

Watkins CJCH (1989) Learning from delayed rewards (Doctoral dissertation). Cambridge University, Cambridge University Press, Cambrdige, UK.

Yang IB, Na SG and Heo H (2017) Intelligent algorithm based on support vector data description for automotive collision avoidance system. *International Journal of Automotive Technology* **18**, 69–77.

Yosinski J, Clune J, Bengio Y and Lipson H (2014) How transferrable are features in deep neural networks? *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems*, Vol. **2**. Cambridge, MA, USA: MIT Press.

Yu A, Palefsky-Smith R and Bedi R (2016) *Deep Reinforcement Learning for Simulated Autonomous Vehicle Control. Course Project Reports: Winter 2016 (CS23 1n: Convolutional Neural Networks for Visual Recognition)*. Stanford University, pp. 1–7.

Zou X, Alexander R and McDermid J (2016) On the validation of a UAV collision avoidance system developed by model-based optimization: challenges and a tentative partial solution. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, 28 June–1 July 2016. Toulouse, France: IEEE, pp. 192–199.

Xiongqing Liu received his PhD degree in Mechanical Engineering from University of Southern California in the summer of 2019. His research was focused on developing transfer reinforcement learning algorithms for robotic and vehicle collision avoidance. By combining reinforcement learning with transfer learning, Dr. Liu introduced a belief-based approach to enable the transfer of knowledge obtained from reinforcement learning in one task domain to another similar task domain. The approach allows engineers to adjust the belief level and transfer period for the best transfer effect depending on the similarity levels of the source and target tasks. Dr. Liu received his BS degree from Shanghai Jiaotong University and is currently working as a system engineer in the machine learning related areas.

Yan Jin is Professor of Aerospace and Mechanical Engineering at University of Southern California. He received his Ph.D. degree in Naval Engineering from the University of Tokyo and did his post-doctoral research at Stanford University. Dr. Jin's research covers design theory & methods, multiagent and self-organizing systems. His current research interests include design cognition, machine learning and its application in engineering design, knowledge capturing, self-organizing and adaptive systems. Dr. Jin is a recipient of National Science Foundation CAREER Award and TRW Excellence in Teaching Award. He served as Editor-in-Chief of AIEDAM and Associate Editor of JMD and is currently Associate Editor of Design Science Journal. Dr. Jin is a Fellow of ASME.