Deep Reinforcement Learning Algorithms for Steering a Underactuated Ship

Pham Tuyen, L., Abu, L., Vien, N. A., & Chung, T. (2017). Deep Reinforcement Learning Algorithms for Steering a Underactuated Ship. In *2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2017)* https://doi.org/10.1109/MFI.2017.8170388

Published in:
2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2017)

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Deep Reinforcement Learning Algorithms for Steering an Underactuated Ship

Le Pham Tuyen[1], Abu Layek[1], Ngo Anh Vien[2], and TaeChoong Chung[1]

***Abstract*— Based on state-of-the-art deep reinforcement learning (Deep RL) algorithms, two controllers are proposed to pass a ship through a specified gate. Deep RL is a powerful approach to learn a complex controller which is expected to adapt to different situations of systems. This paper explains how to apply these algorithms to ship steering problem. The simulation results show advantages of these algorithms in reproducing reliable and stable controllers.**

I. Introduction

Ship steering problem [1] is a complicated task due to highly nonlinear and continuous control under the disturbances such as wave, wind, and current. Some studies have been focused on this domain to make controllers using some approaches such as PID control, fuzzy logic control and genetic algorithm [16][17][18]. Reinforcement Learning [13] is also one of the approaches to produce a controller for the ship. RL based methods interact with the environment and train the controller's parameters using a reward signal at each time step. However, traditional RL algorithms suffer some limits such as failing to deal with continuous control tasks or inefficiency in using data samples during the learning process.

Recent years, reinforcement learning takes advantages of neural network researches and produces a new generation of reinforcement learning algorithm called Deep Reinforcement Learning (Deep RL). The new algorithms employ neural networks inside and allow representing complex behaviors. Some Deep RL algorithms can deal with continuous control tasks such as Deep Deterministic Policy Gradient [2] and Normalized Advantage Function [4]. Moreover, Deep RL algorithms have a mechanism to use data samples efficiently, which makes the learning process more stable without being biased. This paper introduces two Deep RL algorithms, which can be applied to produce stable and reliable controller for ship steering task. The contributions are two-fold. First, we summarize the procedures of two Deep RL algorithms: Deep Deterministic Policy Gradient and Normalized Advantage Function. Second, we apply two Deep RL algorithms to produce controllers for steering the ship. The controllers are expected to help a underactuated ship pass through a specified gate.

The rest of the paper is organized as follows. Section II reviews underlying knowledge such as reinforcement learning and Markov decision process. Section III describes overall ship steering problem. Some Deep RL algorithms are described in Section IV. The performance of algorithms is reported in section V. Finally, section VI summarizes the content of the paper and suggests new ideas for future work.

[1]Department of Computer Science and Engineering, Kyung Hee University, Yongin, Gyeonggi, 446-701, South Korea {tuyenple,layek,tcchung}@khu.ac.kr

[2]EEECS/ECIT Queen's University Belfast, UK. v.ngo@qub.ac.uk

II. Reinforcement Learning and Markov Decision Process

Reinforcement Learning (RL) is a part of machine learning focused on interact with an environment, receives a reward from the environment and learns from the received reward. Usually, RL problems are modeled as a discrete-time Markov Decision Process (MDP) with a tuple of $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$. It includes a state space $\mathcal{S}$; an action space $\mathcal{A}$; a transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$ that measures the probability of obtaining the next state s_{t+1} given a current state-action pair (s_t, a_t); $r(s_t, a_t)$ defines the immediate reward achieved at each state-action pair, and $\gamma \in (0, 1)$ denotes a discount factor. A sequence of state-action pairs (s_t, a_t) creates a trajectory ξ_t (also called an episode) with discounted cumulative reward given by

$$R(\xi) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t).$$

A RL algorithm tries to find an optimal policy π^* in order to maximize the expected total discounted reward. Traditionally, the policy is represented by parameters follow some structures such as linear basis function and radial basis function [14]. Recent years, some new mechanisms allow representing the policy as deep neural networks (DNNs). As a result, DNN has become popular these days and a lot of studies have focused on this topic.

Q-Learning is a well-known approach to find the optimal policy. Q-Learning approach learns to maximize Q function $Q^\pi(s, a)$, which is the expected reward from state-action pair (s, a) and follow policy π:

$$Q^\pi(s_t, a_t) = E[R_t|s_t, a_t].$$

The policy in Q-Learning is a greedy deterministic policy:

$$\mu(s) = argmax_a Q(s, a).$$

In the case of continuous action domain, Q-Learning becomes difficult because it requires maximizing a continuous function. To overcome this problem, Deep Deterministic Policy Gradient (DDPG) uses the actor-critic framework [19] to generate continuous action while Normalized Advantage Function (NAF) has some tricks on Q-network to deal with continuous action. All algorithms are introduced in section IV-D and IV-E respectively.

III. Ship Steering Problem

Ship Steering Problem, introduced in [1], is a continuous state and action task. A ship starts at a randomly position, orientation and turning rate and is to be maneuvered at a constant speed through a gate placed at a fixed position. Equation 1 gives the motion equations of the ship, where $T = 5$ is the time constant for converging to desired turning rate, $V = 3$ m/sec is the constant speed of the ship, and $\Delta = 0.2$ sec is the sampling interval. There is a time lag between changes in the desired turning rate and the actual rate, modeling the effects of a real ship's inertia and the resistance of the water.

$$\begin{aligned} x[t+1] &= x[t] + \Delta V sin\theta[t] \\ y[t+1] &= y[t] + \Delta V cos\theta[t] \\ \theta[t+1] &= \theta[t] + \Delta\dot{\theta}[t] \\ \dot{\theta}[t+1] &= \dot{\theta}[t] + \Delta(r[t] - \dot{\theta}[t])/T \end{aligned} \tag{1}$$

At each time t, the state of the ship is given by its position $x[t]$ and $y[t]$, orientation $\theta[t]$ and actual turning rate $\dot{\theta}[t]$. The action $r[t]$ is the desired turning rate of the ship. All four state variables and also the action are continuous and their range is shown in Table I. The ship steering problem is episodic. In each episode, the goal is learning to generate sequences of actions that steer the center of the ship through the gate in the minimum amount of time. The sides of the gate are placed at coordinates $(850, 900)$ and $(950, 900)$. If the ship moves out of bound ($x < 0$ or $x > 1000$ or $y < 0$ or $y > 1000$), the episode terminates and is considered as a failure.

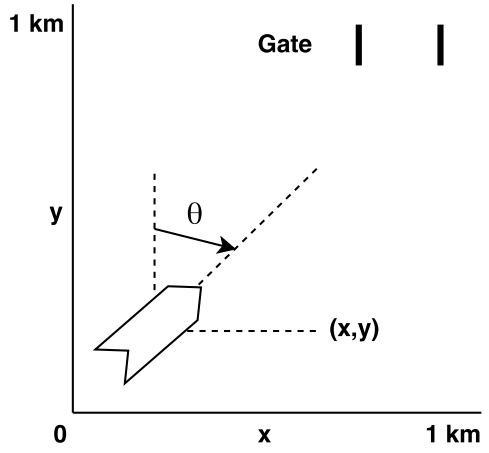


Fig. 1. Ship Steering Domain

TABLE I
Range of state and action for ship steering task

State	x	0 to 1000 meters
	y	0 to 1000 meters
	θ	-180 to 180 degrees
	$\dot{\theta}$	-15 to 15 degrees/sec
Action	r	-15 to 15 degrees/sec

IV. Deep Reinforcement Learning Algorithms for Learning Controller

In this section, we introduce two state-of-the-art Deep RL algorithms, Deep Deterministic Policy Gradient (DDPG) and Normalized Advantage Function (NAF), which are used to learn controllers for steering the ship. Even though they use different techniques to update the controller, they are employed DNNs with parameters, which are updated during the learning process and can represent complex behaviors. They use a mechanism called experience replay for efficiently reusing previous data samples without bias.

A. Deep Neural Network for Representing the Controller

Deep Deterministic Policy Gradient (DDPG) algorithm maintains two deep neural networks, one for a critic and one for an actor. As described in [2], the critic network is constructed from an input layer, two hidden layers, and one output layer. Besides, the critic network uses the rectified linearity (ReLU) layer for all hidden layers. The input layer receives a state and an action from the environment while the output layer generates a Q-value. The actions only include from the 2nd hidden layer. The Q-value is used to update parameters of the actor network.

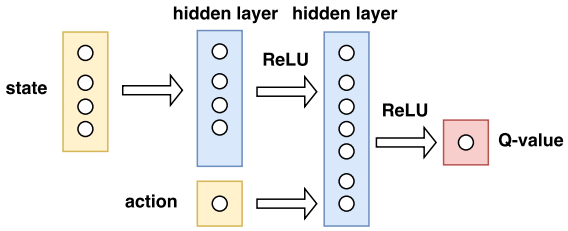


Fig. 2. Critic Network Model

The actor network is used to approximate an action from the current state. It is also constructed from two hidden layers and uses the rectified linearity (ReLU) layer for all hidden layers. The input layer receives states from environment and the output layer is a $\tanh$ layer to bound the actions.

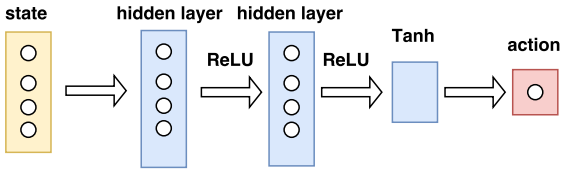


Fig. 3. Actor Network Model

In contrast to DDPG, Normalized Advantage Function (NAF) algorithm only uses a Q network. However, Q network in NAF is modified to deal with continuous control. Particularly, the output of the 2nd hidden layers is separated into a state value term V and an advantage term A. The intuition behind this model is described in the section IV-E.

As in DQN [3], a copy (called target network) of each network in algorithms is created to maintain the stability of

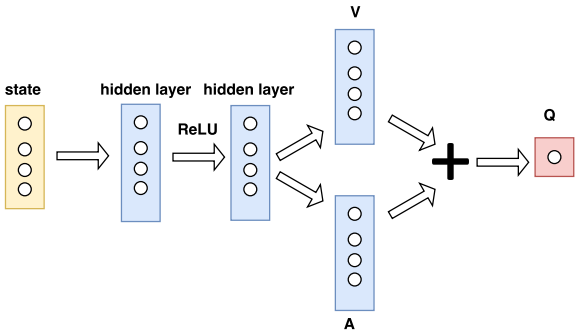


Fig. 4. Q network in NAF

the learning process. The weights of these target networks are updated by having them slowly follow the main network as follows:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

where θ' and θ is parameters of the target network and the main network respectively. τ is learning rate of the target network ($\tau \ll 1$). The update method makes sure that the target network does not suddenly change by the main networks

B. Learning Off-policy using Experience Replay Mechanism

For learning parameters of the neural networks, data samples are collected and feed to the neural network for doing some optimization tasks. However, an inefficient data selection method may lead the learning process to be biased. Experience replay is a mechanism to collect and select data samples efficiently. In DQN, experience replay is implemented by a cache R with a limited size. For each learning step, a tuple of (s_t, a_t, r_t, s_{t+1}) is stored in R. When the cache is full, the oldest samples will be discarded. At each learning step, a batch of tuples are randomly selected from the cache and is used to learn the parameters of the networks. The size of the cache is usually large to make the selected batch of independent and identically distributed (i.i.d) samples.

C. Exploration in continuous action spaces

For Deep RL in continuous action spaces, the policy ($\mu(s|\theta^\mu)$) is modeled by the neural network. Thus, to explore in the action space, we must add a noise signal to the policy:

$$\mu'(s) = \mu(s|\theta^\mu) + \mathcal{N}.$$

The added noise can be sampled from a uniform distribution or any random process. In this paper, we use a random process called Ornstein-Uhlenbeck [5], which is originally used in the original papers.

D. Deep Deterministic Policy Gradient Algorithm

Deep Deterministic Policy Gradient [2] inherits characteristics of a policy gradient method [7] and an actor-critic method [8][19]. As described in section IV-A, the algorithm employs two networks (actor network with parameter θ^μ and critic network with parameter θ^Q) together with their target networks (parameters is $\theta^{\mu'}$ and $\theta^{Q'}$ respectively). The actor network approximates the action from the current state and can deal with continuous action. The critic network approximates Q value from the current state-action pair. Parameters of the actor network are updated using Deterministic Policy Gradient [6] as follows:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a|\theta^Q\right)|_{s=s_i, a=\mu(s_i)} \times \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)|_{s_i}$$

Parameters of the critic network are learned by minimizing TD error as follows:

$$L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2.$$

y_i is approximated using target network of critic as follows:

$$y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)$$

DDPG algorithm is summarized in Algorithm 1.

Algorithm 1: Deep Deterministic Policy Gradient

Input: Main network and target network of critic $Q(s, a|\theta^Q)$ and $Q'(s, a|\theta^{Q'})$; main network and target network of actor $\mu(s, a|\theta^\mu)$ and $\mu'(s, a|\theta^{\mu'})$; Experience replay R.

Initialize: Randomly initialize Q and μ; Initialize target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$; Initialize R

for $episode = 1, 2, \ldots M$ **do**

- **Initialize:** a random process N for action exploration
- **Get** initial observation state s_1
- **for** $t = 1, 2, \ldots T$ **do**
  - **Get** $a_t = \mu(s_t|\theta^\mu) + N_t$
  - **Execute** a_t and observe r_t and s_{t+1}
  - **Store** tuple (s_t, a_t, r_t, s_{t+1}) in R
  - **Sample** a batch of N tuples from R
  - **Compute** $y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)$
  - **Update critic by minimizing the loss:** $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2$
  - **Update actor by using DPG:** $\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a|\theta^Q\right)|_{s=s_i, a=\mu(s_i)} \times \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)|_{s_i}$
  - **Update the target networks:** $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
- **end**

end

E. Normalized Advantage Function

Normalized Advantage Function (NAF) only uses one Q neural network. To adapt with continuous control task, Q network is decomposed into a state value term V and an advantage term A:

$$Q(s,a|\theta^Q) = A(s,a|\theta^A) + V(s|\theta^V)$$

The advantage A is parameterized as a quadratic function of nonlinear features of the state:

$$A(s,a|\theta^A) = -\frac{1}{2}(a-\mu(s|\theta^\mu))^T P(s|\theta^P)(a-\mu(s|\theta^\mu)).$$

$P(s|\theta^P)$ is a state-dependent, positive-definite square matrix, which is parametrized by $P(s|\theta^P) = L(s|\theta^P)L(s|\theta^P)^T$, where $L(s|\theta^P)$ is a lower-triangular matrix whose entries come from a linear output layer of a neural network, with the diagonal terms exponentiated. By this represent of Q-network, the action that maximizes the Q function is given by $\mu(s|\theta^\mu)$ and this represent can deal with continuous action task.

Algorithm 2: Normalized Advantage Function

Input: Main Q Network and Target Q Network $Q(s,a|\theta^Q)$ and $Q'(s,a|\theta^{Q'})$; Experience Replay R.

Initialize: Randomly initialize Q; Initialize target networks Q' with weights $\theta^{Q'} \leftarrow \theta^Q$; Initialize R

for $episode = 1, 2, \dots M$ **do**

  Initialize: a random process N for action exploration

  Get initial observation state s_1

  for $t = 1, 2, \dots T$ **do**

    Get $a_t = \mu(s_t|\theta^\mu) + N_t$

    Execute a_t and observe r_t and s_{t+1}

    Store tuple (s_t, a_t, r_t, s_{t+1}) in R

    Sample a batch of N tuples from R

    Compute $y_i = r_i + \gamma V'(s_{t+1}|\theta^{V'})$

    Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2$$

    Update the target network:

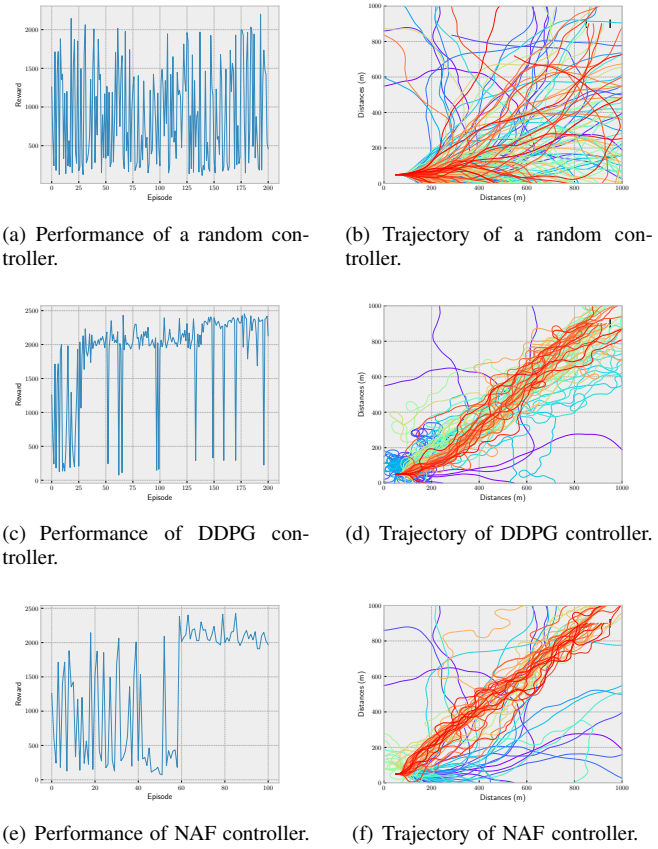
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

  end

end

V. Experiments

In this section, we use two algorithms introduced above to learn the ship's controllers. Two algorithms are implemented in Python and use tensorflow library [15] to build deep neural networks. Details about deep neural network follow original papers [2][4]. Particularly, two algorithms use ADAM [9] to learn the parameters of neural networks. The learning rate for the actor network and the critic network in DDPG are 0.0001 and 0.001 respectively. The learning rate for Q network in NAF is 0.001. The actor network has 200 units for each hidden layer and the critic network has 400 and 300 units for 1st and 2nd hidden layer respectively. Parameters of the actor network and the critic network are initialized from a uniform distribution $[3\times10^3, 3\times10^3]$ and $[3\times10^4, 3\times10^4]$. Parameters of Q network in NAF are also initialized from a uniform distribution $[3\times10^4, 3\times10^4]$. Two algorithms have some sharing parameters such as discount factor $\gamma = 0.99$, the rate of soft target updates $\tau = 0.001$, the batch size is $N = 256$ and the size of experience replay is 10^6. Exploration in action space follows Ornstein-Uhlenbeck process [5] with $\theta = 0.15$ and $\sigma = 1.0$.



(a) Performance of a random controller.

(b) Trajectory of a random controller.

(c) Performance of DDPG controller.

(d) Trajectory of DDPG controller.

(e) Performance of NAF controller.

(f) Trajectory of NAF controller.

Fig. 5. Ship trajectory and performance of algorithms during the learning process

The reward function is defined as follows:

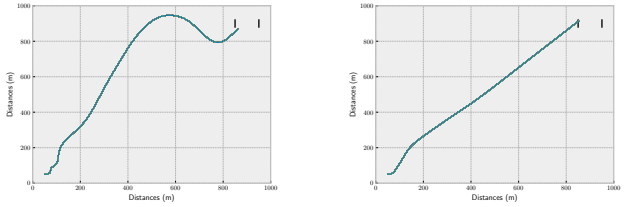
$$r(s,a) = exp(-(\frac{\Delta_{angle}}{\pi})^2) \quad (2)$$

Δ_{angle} is angle created by the ship and the gate center. If the ship towards the goal, Δ_{angle} will small and the algorithm receives a big reward. Otherwise, if the ship goes far away from the gate, Δ_{angle} is big and the algorithm receives a small reward. Δ_{angle} is in range of $[-\pi, \pi]$. Thus, at each time step, the algorithms receive a reward signal in range $[e^{-1}, 1]$

A. Fixed Initial State

The first simulation is made with a fixed position and orientation of the ship. Particularly, the ship starts at position $(50, 50)$ m and orientation $\theta = \pi/2$. Fig. 5 reports trajectories of the ship and the performance of algorithms during 100 episodes. The colors of trajectories are assigned from purple to red to indicate the process of learning. Purple color means the episodes at the beginning of the learning process and the red color means the episodes at the end of the learning process. With a random controller, the figure does not show the improvement of the controller to control the ship. However, with Deep RL controllers, almost latest trajectories (red trajectories) help the ship reach the gate. They mean that the Deep RL controllers have been learned to control the ship.

After the learning process finished, stable controllers have been made. Fig. 6 shows trajectory of the ship controlled by learned controllers. Without exploration noise added, the figure shows smooth trajectories towards the gate for two algorithms.



(a) Evaluate DDPG controller. (b) Evaluate NAF controller.

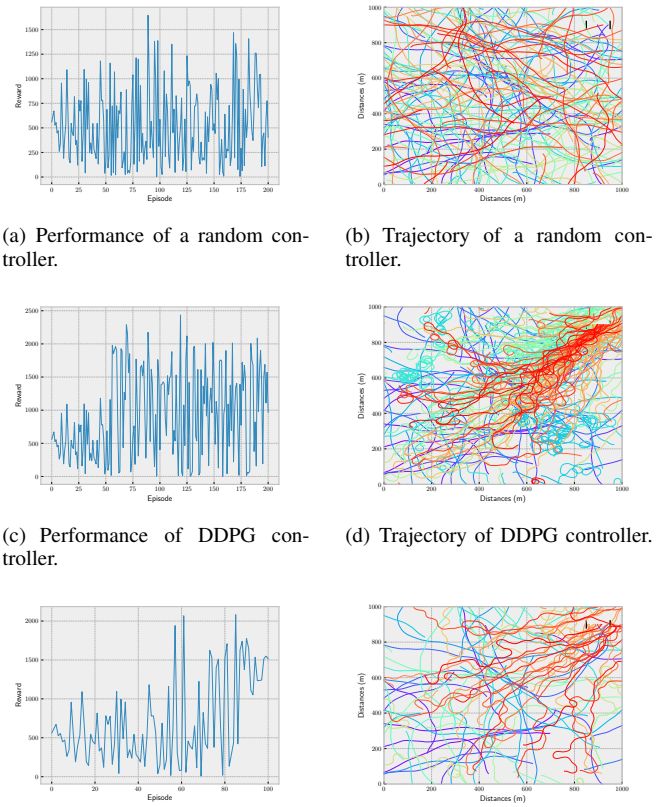
Fig. 6. Ship trajectory when using learned controllers

B. Random Initial State

This task challenges algorithms by allowing the ship to start at any position or orientation. Because of random initial state, the difficulty of this task has been increased. As a result, the number of episodes for learning an optimal controller is greater than the previous task. Performance and trajectories during the learning process are reported in Fig. 7. Similar to behaviors of the previous task, the ship controlled by a random controller move randomly on the whole map and never reach the goal intentionally. In contrast with the random controller, the Deep RL controllers are updated and can lead the ship towards the goal at the end of the learning process. DDPG controller needs more iterations than NAF controller to obtain a good controller (200 versus 100). In addition, the performance report is also shown that Deep RL algorithms achieve a higher cumulative reward than a random controller. Trajectories of the ship follow optimal controllers are shown in Fig. 8. Although the controllers are only trained for a short time, they can help the ship reach the gate quickly. When the controllers are trained for a long time, they can adapt to more and more situations.

VI. Conclusion

In this paper, the ship steering problem is solved by using some state-of-the-art deep reinforcement learning algorithms.

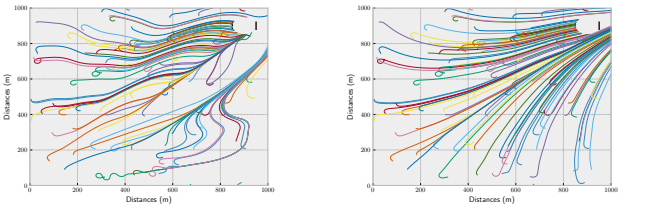


(a) Performance of a random controller. (b) Trajectory of a random controller.

(c) Performance of DDPG controller. (d) Trajectory of DDPG controller.

(e) Performance of NAF controller. (f) Trajectory of NAF controller.

Fig. 7. Ship trajectory and performance of algorithms during the learning process



(a) Evaluate DDPG controller. (b) Evaluate NAF controller.

Fig. 8. Ship trajectory when using learned controllers

Deep RL algorithms have been proven to help the ship to pass a specified gate. The ship can start at any position with any orientation and can reach the gate in a short time. There has a complex scenario, which the ship needs to sequentially pass some specified gates rather than one gate as setup in this paper. For this scenario, the learned controller needs to satisfy multiple objectives (pass multiple gates), which is hard to achieve for a primitive Deep RL. An approach to solve this scenario is using a hierarchical reinforcement learning algorithm as describe in [10][11][12]. We want to leave this scenario as a future work.

ACKNOWLEDGEMENT

The authors are grateful to the Basic Science Research Program through the National Research Foundation of Korea (NRF-2017R1D1A1B04036354) and Kyung Hee University (KHU-20160601).

REFERENCES

[1] Miller, W. Thomas, Paul J. Werbos, and Richard S. Sutton, eds. "Neural networks for control". MIT press, 1995.
[2] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[3] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.
[4] Gu, Shixiang, et al. "Continuous deep q-learning with model-based acceleration." International Conference on Machine Learning. 2016.
[5] Uhlenbeck, George E., and Leonard S. Ornstein. "On the theory of the Brownian motion." Physical review 36.5 (1930): 823.
[6] Silver, David, et al. "Deterministic policy gradient algorithms." Proceedings of the 31st International Conference on Machine Learning (ICML-14). 2014.
[7] Peters, Jan, and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." Neural networks 21.4 (2008): 682-697.
[8] Sutton, Richard S., and Andrew G. Barto. "Reinforcement learning: An introduction." Vol. 1. No. 1. Cambridge: MIT press, 1998.
[9] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[10] Ghavamzadeh, Mohammad, and Sridhar Mahadevan. "Hierarchical policy gradient algorithms." Proceedings of the 20th International Conference on Machine Learning (ICML-03). 2003.
[11] Barto, Andrew G., and Sridhar Mahadevan. "Recent advances in hierarchical reinforcement learning." Discrete event dynamic systems 13.4 (2003): 341-379.
[12] Kulkarni, Tejas D., et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation." Advances in Neural Information Processing Systems. 2016.
[13] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.
[14] Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters. "A survey on policy search for robotics." Foundations and Trends in Robotics 2.12 (2013): 1-142.
[15] Abadi, Martn, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).
[16] Zhang, Lixing, et al. "Neural-network-based reinforcement learning control for path following of underactuated ships." Control Conference (CCC), 2016 35th Chinese. IEEE, 2016.
[17] Shen, Zhi-peng, Chen Guo, and Shi-chun Yuan. "Reinforcement learning control for ship steering using recursive least-squares algorithm." IFAC Proceedings Volumes 38.1 (2005): 133-138.
[18] Ye, Guang, and Chen Guo. "SA-RL Algorithm Based Ship Steering Controller." Neural Networks and Brain, 2005. ICNN and B'05. International Conference on. Vol. 3. IEEE, 2005.
[19] Konda, Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." Advances in neural information processing systems. 2000.