Thesis for the Degree of Master of Science

Application of Machine and Deep Learning to Mooring, Dynamic Positioning, and Ship Berthing Systems

Daesoo Lee

August 2020

Department of Naval Architecture and Ocean Systems Engineering

Graduate School

Korea Maritime and Ocean University

Application of Machine and Deep Learning to Mooring, Dynamic Positioning, and Ship Berthing Systems

by

Daesoo Lee

Advised by

Prof. Seungjae Lee

A Thesis Submitted to the Graduate School of

Korea Maritime and Ocean University in the partial fulfillment

Of the requirements for degree of

Master of Science

Table of Contents

List of Tables iv
List of Figures vi
Abstract xii

1. Introduction

2. Background of Machine and Deep Learning
2.1 Machine Learning 4
2.2 Deep Learning 9
2.2.1 Types of Deep Learning Layers 9
2.2.2 Activation Function and Weight Initialization Methods 18
2.2.3 Optimizers 19
2.2.4 Training Dataset Scaling 26
2.2.5 Transfer Learning 28
2.3 Reinforcement Learning 28

3. Machine Learning-Based Mooring Line Tension Prediction System
3.1 Introduction 39
3.2 Brief Comparison Between Conventional and Proposed Mooring Line Tension Prediction Systems 40
3.3 Proposed K-Means-Based Sea State Selection Method 41
3.3.1 Padding 42
3.3.2 K-Means 44
3.3.3 K-Means-Based Monte Carlo Method 45
3.3.4 Feature Vector Generation 47

3.3.5 Clustering of Relevant Sampled Sea States with K-Means 48
3.4 Proposed Hybrid Neural Network Architecture 50
3.4.1 Architecture 50
3.4.2 Training Procedure 54
3.5 Simulation and Result Discussion 55
3.5.1 Simulation Conditions 55
3.5.2 Overall Hs-focused NN model 56
3.5.3 Effectiveness of Batch Normalization 59
3.5.4 Low Hs-focused NN model 60
3.5.5 Proposed Hybrid Neural Network Architecture 61

4. Motion Predictive Control for DPS Using Predicted Drifted Ship Position Based on Deep Learning and Replay Buffer
4.1 Introduction 65
4.2 PID Feed-Back System and Wind Feed-Forward System 66
4.3 Proposed Motion Predictive Control 69
4.4 Numerical Modeling of Target Ship's Behavior 73
4.4.1 Target Ship and DPS 73
4.4.2 Equation of Motion of Target Ship 74
4.5 Effectiveness of Proposed Algorithms 76
4.5.1 Simulation Conditions 76
4.5.2 Types of Deep Learning Layers 77
4.5.3 Real-Time Normalization Method 78
4.5.4 Replay Buffer 80
4.6 Simulation and Result Discussion 81
4.6.1 Simulation Under One Environmental Condition 81
4.6.2 Simulation Under Two Different Sequential Environmental Conditions 84

5. Reinforcement Learning-Based Adaptive PID Controller for DPS
5.1 Introduction 88
5.2 Target Ship and DPS 90
5.2.1 PID Control in DPS 91
5.2.2 Hydrodynamics Associated with a Drifting Motion of a Ship 93

5.3 Proposed Adaptive Fine-Tuning System for PID Gains in DPS 95
5.4 Simulation Results 99
5.4.1 Effectiveness of the Proposed Adaptive Fine-Tuning System 99
5.4.2 Overall Performance Assessment 103
5.5 Discussion 107

6. Application of Recent Developments in Deep Learning To ANN-based Automatic Berthing System

6.1 Introduction 111
6.2 Mathematical Model of Ship Maneuvering 112
6.2.1 Mathematical Model for Ship-Maneuvering Problem 113
6.2.2 Modeling of Propeller and Rudder 114
6.3 Artificial Neural Network and Important Factors in Training the Network 115
6.3.1 Artificial Neural Network 115
6.3.2 Optimizer 117
6.3.3 Input Data Scaling 117
6.3.4 Number of Hidden Layers 118
6.3.5 Overfitting Prevention 118
6.4 Application of Recent Developments in Deep Learning to Automatic Berthing 119
6.5 Simulation and Result Discussion 125

7. Conclusion

7.1 Machine Learning-Based Mooring Line Tension Prediction System 131
7.2 Motion Predictive Control for DPS Using Predicted Drifted Ship Position Based on Deep Learning and Replay Buffer 132
7.3 Reinforcement Learning-Based Adaptive PID Controller for DPS 133
7.4 Application of Recent Developments in Deep Learning to ANN-Based Automatic Berthing System 134

List of Tables

Table 1 Procedure of the batch normalization during the training 18

Table 2 Brief procedure of the RL based on the Q function 31

Table 3 Overall learning process of the DQN 36

Table 4 Overall learning process of the DDPG 37

Table 5 Procedure of the K-means-based Monte Casrlo method 45

Table 6 Procedure of finding the representative sea state in each cluster 49

Table 7 Detailed procedure of training the proposed hybrid neural network architecture........... 54

Table 8 Principal dimension of the target ship 55

Table 9 Properties of the mooring lines 56

Table 10 Parameter settings for the K-means-based sea state selection method......................... 56

Table 11 Principal dimension of the FPSO 73

Table 12 Locations and thrust limits of the azimuth thrusters 74

Table 13 Settings of the parameters in the proposed motion predictive control 77

Table 14 Environmental condition for the simulations for the effectiveness test 77

Table 15 Details of the three DNNs.. 78

Table 16 Statistics of the ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition) 82

Table 17 Two different sequential environmental conditions.. 84

Table 18 Statistics of the ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under two environmental conditions)................................... 86

Table 19 Principal dimension of the FPSO ship 90

Table 20 Details of DP thrusters.. 91

Table 21 Definition of sea states... 99

Table 22 Environmental condition for the effectiveness test..100

Table 23 Environmental condition for the overall performance test..104

Table 24 Principal particulars of a target ship 112

Table 25 List of input data-scaling methods...117

Table 26 Details of training models with respect to activation and weight initialization methods 119

Table 27 Details of training models with respect to activation and weight initialization methods 120

Table 28 Procedure for obtaining berthing performance history 121

Table 29 Details of training models with respect to number of hidden layers...........................122

Table 30 Details of training models when BN is used 123

Table 31 Trained model selection algorithm for a neural network for automatic ship berthing .125

List of Figures

Fig. 1 Overall configuration of machine learning 4

Fig. 2 Example of the labeled training data 5

Fig. 3 Illustration of the data clustering.. 6

Fig. 4 Illustration of the dimensionality reduction 6

Fig. 5 Illustration of the RL learning... 7

Fig. 6 Definitions of the machine learning and deep learning.. 8

Fig. 7 Illustration of the ANN and DNN 8

Fig. 8 Calculation in one node of the ANN 10

Fig. 9 Activation function of the tanh.. 10

Fig. 10 Configuration of the DNN 11

Fig. 11 Configuration of the RNN layer 12

Fig. 12 Configuration of RNN cell... 13

Fig. 13 Example of the long-term dependency 13

Fig. 14 Configuration of the LSTM layer.. 14

Fig. 15 Configuration of the LSTM cell.. 14

Fig. 16 Configuration of the GRU layer 15

Fig. 17 Configuration of the GRU cell 15

Fig. 18 Illustration of the overfitting problem compared to underfitting and appropriate fitting. 16

Fig. 19 Illustration of the dropout 17

Fig. 20 Illustration of the gradient descent 21

Fig. 21 Illustration of the gradient descent with respect to two weights...................................... 21

Fig. 22 Effects of the learning rate.. 21

Fig. 23 Illustration of the backpropagation in one neuron in the last hidden layer compared to forward-propagation 22

Fig. 24 Illustration of the backpropagation in the DNN 23

Fig. 25 Illustration of the effect of different distribution and scale of the input feature vector during the training 27

Fig. 26 Comparison between the traditional learning and the transfer learning 28

Fig. 27 Diagram of the major RL algorithms 29

Fig. 28 Learning process of an agent with MDP 30

Fig. 29 Common applications of the value-based RL algorithm and the policy-based RL algorithm 33

Fig. 30 Illustration of the policy in the value-based RL algorithms 34

Fig. 31 Illustration of the policy in the policy-based RL algorithms 34

Fig. 32 Illustration of the replay buffer in the DQN 36

Fig. 33 Architecture of the actor in the DDPG 38

Fig. 34 Brief diagram of the conventional ANN-based mooring line tension prediction system . 40

Fig. 35 Plain ANN model used in the conventional ANN-based mooring line tension prediction systems 41

Fig. 36 Brief diagram of the proposed DL-based mooring line tension prediction system 41

Fig. 37 Detailed diagram of the proposed K-means-based sea state selection method 42

Fig. 38 Brief diagram of the CNN's convolution operation in one layer 43

Fig. 39 Illustration of application of the padding 43

Fig. 40 Application of the padding to the wave scatter diagram 44

Fig. 41 Examples of data clustering by the K-means 45

Fig. 42 Comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method 46

Fig. 43 Application of the K-means-based Monte Carlo method to the padded valid sea state area 46

Fig. 44 Comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method on the padded valid sea state area 47

Fig. 45 Example of the FFTCGx 48

Fig. 46 Illustration of the clustering of the relevant sampled sea states with the K-means........ 49

Fig. 47 Illustration of the selected sea states by the representative sea states 50

Fig. 48 Architecture of the proposed hybrid neural network 51

Fig. 49 Architecture of the overall Hs-focused NN model 52

Fig. 50 Architecture of the low Hs-focused NN model 52

Fig. 51 Architecture of the proposed soft-binary attention module........ 53

Fig. 52 Mooring configuration of the target ship 56

Fig. 53 Test sea states........ 56

Fig. 54 Selected sea states obtained by the manual sea state selection and the proposed K-means-based sea state selection method........ 57

Fig. 55 R-squared and MAE scores of the trained overall Hs-focused NN model on the test sea states........ 58

Fig. 56 Predicted tension time histories of the overall Hs-focused NN model trained on the sea states obtained by the proposed K-means-based sea state selection method........ 59

Fig. 57 R-squared and MAE scores of the trained overall Hs-focused NN models with and without the BN on the test sea states........ 60

Fig. 58 Sampled low Hs-sea states for Dl........ 60

Fig. 59 R-squared and MAE scores of the trained low Hs-focused NN model on the test sea states 61

Fig. 60 R-squared and MAE scores of the trained hybrid neural network architecture on the test sea states........ 62

Fig. 61 Tension time histories along with the attention value time histories of the trained hybrid neural network architecture 64

Fig. 62 Coordinate system of a ship 67

Fig. 63 Structure of the PID feed-back system in the DPS 68

Fig. 64 Structure of the PID feed-back system with the wind feed-forward system 69

Fig. 65 Structure of the PID feed-back system with the proposed motion predictive control...... 69

Fig. 66 Structure of the DNN 71

Fig. 67 Overall training process of the proposed motion predictive control 72

Fig. 68 Arrangement of the azimuth thrusters 74

Fig. 69 Environmental loads contributing to the drifting motion 75

Fig. 70 Ship motion time histories of the three different DNNs based on the ANN, LSTM, and GRU 78

Fig. 71 Ship motion time histories with respect to the use of the real-time normalization method 80

Fig. 72 Loss time histories with respect to the use of the real-time normalization method 80

Fig. 73 Ship motion time histories with respect to the use of the replay buffer 80

Fig. 74 Loss time histories with respect to the use of the replay buffer 81

Fig. 75 Ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition) 82

Fig. 76 Comparison between the actual and predicted ship motion time histories (under one environmental condition) 83

Fig. 77 Thrust time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition) 84

Fig. 78 Ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under two environmental conditions) 85

Fig. 79 Comparison between the actual and predicted ship motion time histories (under two environmental conditions) 85

Fig. 80 Global and local coordinate systems 91

Fig. 81 Arrangement of DP thrusters... 91

Fig. 82 Structure of the PID control system in the DPS 92

Fig. 83 Environmental loads contributing to the drifting motion 93

Fig. 84 Difference between the 1st order and 2nd order wave loads 94

Fig. 85 Overall structure of the proposed adaptive fine-tuning system 95

Fig. 86 Illustration of the drifting side (direction of an environmental load: right to left)........... 97

Fig. 87 Example of the wave drift load time history (sea state: very rough, environmental direction to a ship: 40°) 97

Fig. 88 Illustration of the large rebound motion caused by the drastic increase of the integral of error.. 98

Fig. 89 Architectures of the actor and critic.. 98

Fig. 90 Effects of the adaptive PD gains 101

Fig. 91 Effects of the adaptive I gain and the update gate of the integral of error.....................103

Fig. 92 The overall performance assessment of the adaptive fine-tuning system......................106

Fig. 93 Wave drift loads with respect to the environmental directions (sea state: ‘high’) 107

Fig. 94 Directionality of thrust in the control system 109

Fig. 95 Coordinate system 112

Fig. 96 Basic architecture of neural network, and concept of imaginary line 115

Fig. 97 Teaching dataset..116

Fig. 98 Flowchart for ANN-based ship berthing system 117

Fig. 99 Loss function histories with respect to activation and weight initialization methods 119

Fig. 100 Loss function histories with respect to input data-scaling methods 121

Fig. 101 Loss function and berthing performance histories with respect to number of hidden layers 123

Fig. 102 Set of new initial positions for berthing performance history to verify universal berthing performance 123

Fig. 103 Loss function and berthing performance histories when BN is used 124

Fig. 104 Berthing trajectories with initial positions from training dataset (red: with BN, blue: without BN) 127

Fig. 105 Berthing trajectories with interpolated initial positions (red: with BN, blue: without BN) 128

Fig. 106 Berthing trajectories with extrapolated initial positions, Type 1 (red: with BN, blue: without BN) 129

Fig. 107 Berthing trajectories with extrapolated initial positions, Type 2 (red: with BN, blue: without BN) 129

Fig. 108 Berthing trajectories with extrapolated initial positions, Type 3 (red: with BN, blue: without BN) 130

Application of Machine and Deep Learning to Mooring, Dynamic Positioning, and Ship Berthing Systems

Daesoo Lee

Department of Naval Architecture and Ocean Systems Engineering
Graduate School of Korea Maritime and Ocean University

Abstract

In recent years, there have been a surge of advances in machine and deep learning due to accessibility to a large amount of digital data, developments in computer hardware, and state-of-the-art machine and deep learning algorithms proposed. The robust performance of the recent machine and deep learning algorithms have been proven in many applications such as natural language processing, computer vision, market research, self-driving car, autonomous shipping, and so on. The application of machine and deep learning is very powerful in a sense that one does not need to build such a complex and hard-coded system to implement sophisticated functionality. Instead, a machine and deep learning-based system can be trained on a collected training dataset and the trained system can robustly perform as desired. There are two main advantages of the use of machine and deep learning-based systems over the traditional hard-coded systems. First, as mentioned, the machine and deep learning-based systems do not require such complex and hard-coded algorithms, therefore, such learning systems are less prone to errors and faster to implement without much debugging. Second, the machine and deep learning-based systems can adapt to varying circumstances through re-training based on collected data. An example of the varying circumstance can be a varying purchase trend impacted by the media. Therefore, even if the input distribution from the circumstance changes over time, the machine and deep learning-based systems can easily adapt. In this paper, the machine and deep learning algorithms are applied to various applications such as a mooring system, dynamic positioning system (DPS), and ship berthing system. Specifically, the machine and deep learning algorithms are utilized to build a mooring line tension prediction system, a feed-forward system for

DPS, an adaptive proportional-integral-derivative (PID) controller for DPS, and an automatic ship berthing system.

KEY WORDS: Machine learning, Deep learning, Mooring system, Dynamic positioning system (DPS), Automatic berthing system

Chapter 1 Introduction

Traditionally, the machine and deep learning algorithms were not as widely used as now due to their limited performance for practical applications. However, in the past decades, there have been major changes that contributed to wide applications of the machine and deep learning. First, a huge amount of digital data such as texts, images, and numerical data, became available, which is called big data in the deep learning literature. Second, an efficient computational approach was developed for matrix calculation of neural networks by utilizing a Graphics Processing Unit (GPU) instead of a Central Processing Unit (CPU) in which the neural networks are the main component in the deep learning. This utilization of the GPU could extraordinarily speed up the neural network training. Third, the state-of-the-art deep learning algorithms have been proposed for a learning algorithm (optimizer), a neural network architecture, and a neural network layer. The robust performance of the recent machine and deep learning algorithms have been proven in many applications such as natural language processing (NLP), computer vision, market research, self-driving, and so on. For example, in a domain of language translation in the NLP, it was very difficult to process texts in one language to another because there are so many things to consider such as different grammar rules, nuance, non-matching words, words with multiple meanings, gender-dependency, and so on. But, with the help of the deep learning and the big data, now there are deep-learning-based translators that perform nearly as good as a human translator. In the computer vision, an image processing is done using something called a filter which is a $n \times n$ matrix and it filters an input image to process to obtain a desired output. But the filter used to be manually designed by experts in the domain, which made the computer vision system hard to build. Nonetheless, with the deep learning algorithm called a convolutional neural network (CNN), the manual design of the filter is no longer needed but the CNN can learn to design the filter to efficiently process an input image for a given task. There are two main advantages of the use of machine and deep learning-based systems over the traditional hard-coded systems. First, the machine and deep learning-based systems do require the hard coding to implement some functionality for a specific task. Instead, they are just trained to perform the given task. This way, the time to build a system for the given task can be shortened because building the system is completed once the training is completed, the developed learning system is less prone to errors, which provides robustness to the system, and the performance of the system is likely to be better than the hard-coded systems because the deep learning can learn how to do the task very efficiently beyond human understanding and intuition on the given task. Second, the machine and deep learning-based systems have adaptability to a varying circumstance which provides a varying input to the system. For hard-coded systems, it can perform well for a target input data distribution which was set when designed. However, if the input data distribution changes, the hard-coded system is likely to poorly perform because it was not designed for the different input data distribution and now requires additional codes to cope with the change, which is not efficient in terms of maintenance. In case of the machine and deep learning-based systems, although the input data distribution changes a bit, they can still yield good performance because the machine and deep learning algorithms have a good extrapolation ability based on the understanding learned from the training dataset. Even if the input data distribution changes significantly, the system can be re-trained to adjust to the changed input data distribution, which makes the machine and deep learning-based systems easier to maintain in the long term.

In this paper, the machine and deep learning algorithms are applied to various applications such as a mooring system, dynamic positioning system (DPS), and ship berthing system. Specifically, the machine and deep learning algorithms are utilized to build a mooring line tension prediction system, a motion predictive control for DPS, a reinforcement learning (RL)-based adaptive proportional-integral-derivative (PID) controller for DPS, and an automatic ship berthing system.

The proposed mooring line tension prediction system is a system that predicts mooring line top tension at every timestep given a previous ship motion history based on the deep learning algorithm. The mooring line tension prediction system has been developed in several previous studies to speed up dynamic simulation and fatigue analysis on mooring lines by replacing the dynamic simulation process for the mooring tension calculation with the proposed tension prediction system. This speed-up can be achieved because the mooring line tension prediction system can calculate the tension very quickly, while the dynamic simulation requires a huge amount of numerical calculation to represent the actual system's dynamic behavior. The quick calculation of the mooring line tension prediction system is possible because it only requires matrix calculation of the neural network.

The proposed motion predictive control for DPS is a control algorithm for the DPS to reduce the ship drifting motion. The reduction of the drifting motion is achieved by utilizing a predicted drifted ship position. The prediction of the drifted ship position is performed by the deep learning algorithm with its input of ship positions, velocities, acceleration, thrust force, and estimated wind force. A proportional term in the conventional PID is replaced with the predicted drifted ship position and this replacement allows the DPS to counteract the future drifting motion in advance, which results in the reduction of the drifting motion.

The proposed RL-based adaptive PID controller for DPS is a PID controller with its gains that adaptively changes at every timestep. Therefore, the main difference between the RL-based adaptive PID controller and the conventional PID is the changeability of the gains. The gain change is performed by the RL algorithm given its input of ship positions, velocities, and thrust force. The RL algorithm is adopted to perform the adaptive gain change because the RL algorithm does not require any prior knowledge in ship dynamics or the DPS, but it can still learn an optimal policy for the adaptive gain control through an interaction with a given environment.

The proposed automatic ship berthing system is an automatic ship berthing system based on the deep learning system. The deep learning system takes an input of ship positions, velocities, and perpendicular distances between a target berthing point and a current ship position and outputs a target rudder angle and a target revolution per second (RPS). Traditionally, the berthing was performed by an experienced captain because it was a difficult task to automate the berthing due to non-linear characteristics in a ship motion occurring while berthing, where the non-linear characteristics in the ship motion during the berthing occurs because of sudden changes in the rudder angle, RPS, and the slow speed of the ship. However, by the proposed automatic ship berthing system utilizing the deep learning algorithms, it can effectively automate the berthing process.

In the following chapters, backgrounds of the machine learning, deep learning are first introduced. Then, details of each proposed system are presented.

Chapter 2 Background of Machine and Deep Learning

2.1. Machine Learning

An overall configuration of the machine learning is shown in Fig. 1. The machine learning consists of three main components: a) supervised learning, b) unsupervised learning, c) reinforcement learning.

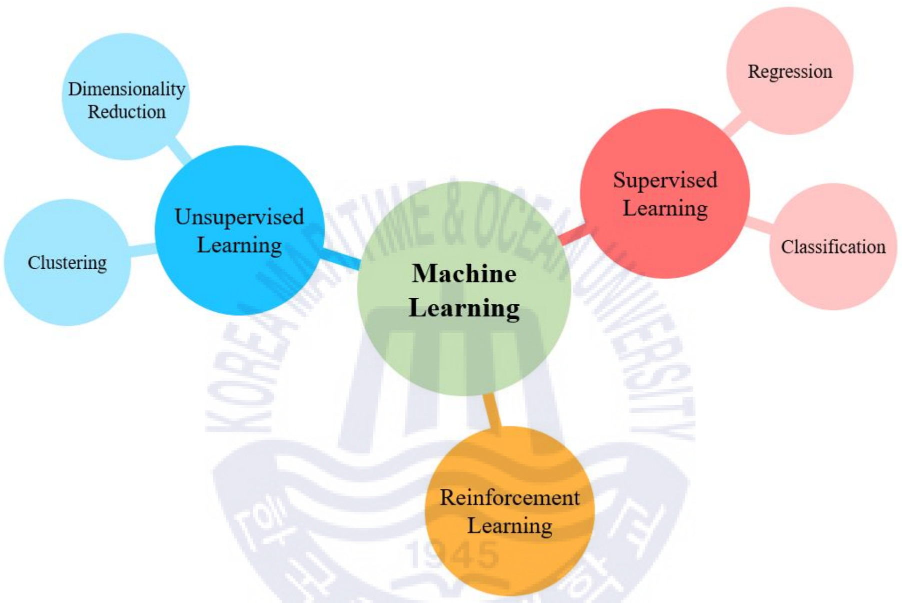
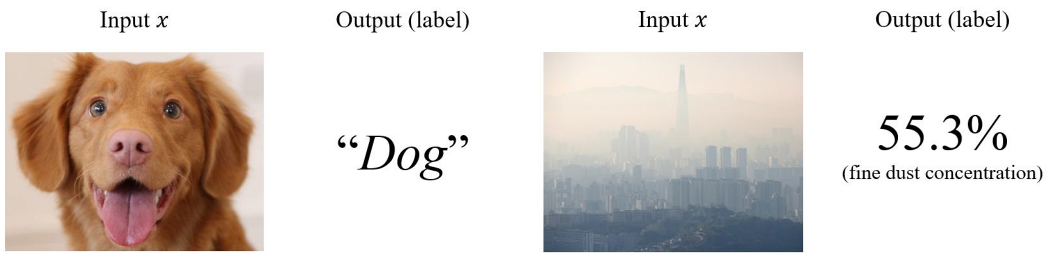


Fig. 1 Overall configuration of machine learning

The supervised learning is a type of machine learning where the machine learning system is trained to yield an appropriate output given an input. Depending on the type of output, the supervised learning is performed to do either a classification task or a regression task. In the classification task, the type of output is an integer that refers to a classification index. In the regression task, the type of output is usually a real number. For example, if the machine learning system is trying to classify whether an image is a dog or a cat, it is solving the classification problem, and if the machine learning system is trying to predict a today's fine dust concentration given an image of the sky, it is solving the regression problem. When the machine learning is trained by the supervised learning, it requires a labeled training dataset. The labeled training dataset is the training dataset consisting of pairs of an input and output (label). An example of the labeled training data is shown in Fig. 2 where the input images are converted to numeric matrices and the output of the word for the classification is converted to the classification index number for the training. Then, the labeled training dataset is simply a large number of the input-output pairs stacked.



(a) classification (b) regression

Fig. 2 Example of the labeled training data

The unsupervised learning is a type of machine learning where the machine learning system is trained with an unlabeled training dataset. Unlike the labeled training dataset, the unlabeled training dataset does not have the output label but input only. This unlabeled training dataset can be used to conduct data clustering or dimension reduction. The data clustering is basically grouping scattered data into multiple groups. Some examples of the data clustering are grouping customers based on their purchase items and grouping online users based on their search histories. The data clustering helps to find patterns behind the data. The dimensionality reduction is a technique to effectively reduce a dimension of input feature vector in which the input feature vector is the input vector that is fed into the machine learning system. By doing so, the size of the machine learning system can be reduced as the input feature vector is downsized, which results in faster implementation to yield the output given the input. Illustrations of the data clustering and dimensionality reduction are shown in Fig. 3 and Fig. 4, respectively. In Fig. 3, the scattered data is clustered into two groups given the features of income and spending. It can be seen that a spending pattern can be found through the data clustering. In Fig. 4, the dimension of a circle in a three-dimension is projected to a two-dimension, which reduces the dimension of the circle. In the same concept as Fig. 4, the machine learning-based dimensionality reduction is performed to reduce the dimension of the input feature vector.

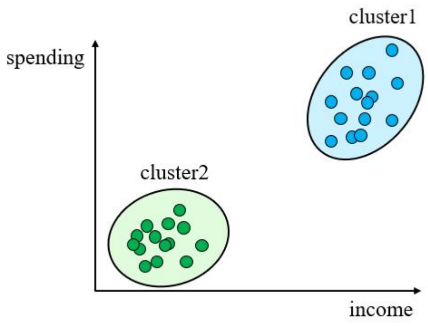


Fig. 3 Illustration of the data clustering

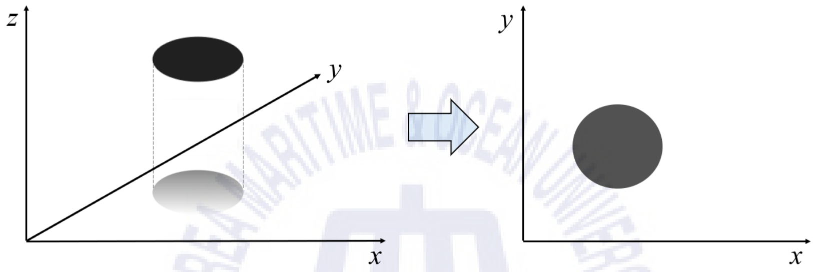
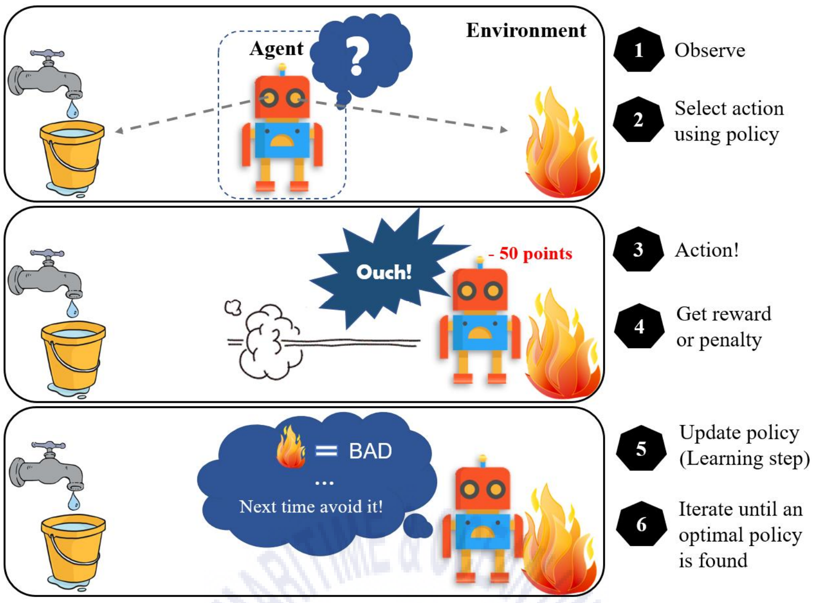


Fig. 4 Illustration of the dimensionality reduction

The reinforcement learning is a type of machine learning where the machine learning system does not require any pre-obtained training dataset. But the learning process is performed through an interaction with a given environment by experiencing trial-and-errors. A unit that interacts with the given environment is called an agent and the interaction with the environment is done by the agent taking actions. After the agent takes action in one state in the environment, the agent is given the next state and a reward in which the reward is a measurement for how good your action was in the previous state. The agent gets to learn a better and better policy to take appropriate actions to maximize the rewards. The RL learning is completed when the rewards that the agent can receive are maximized throughout an episode. The policy that can maximize the rewards is called an optimal policy. An illustration of the RL learning is shown in **Fig. 5** where a penalty is equal to a negative reward.



Agent
Environment
1 Observe
2 Select action using policy
Ouch!
- 50 points
3 Action!
4 Get reward or penalty
= BAD
...
Next time avoid it!
5 Update policy (Learning step)
6 Iterate until an optimal policy is found

Fig. 5 Illustration of the RL learning

There are two main advantages of the RL. First, the RL algorithm does not need any pre-obtained training dataset which is often costly to obtain. Without any pre-obtained training dataset, the RL algorithm can find the optimal policy. Second, the RL algorithm does not require any understanding of dynamics lying in the environment but the RL algorithm can learn the dynamics through the interaction with the environment.

So far, a concept of the machine learning has been addressed but not the deep learning. The concept or definition of the deep learning is often confused with the machine learning. The definitions of the machine learning and deep learning are shown in **Fig. 6** to clarify the definition of each. The machine learning algorithms refer to algorithms whose performance increases as more and more data is fed to train. The simplest machine learning algorithm is a linear regression model. Among the machine learning algorithms, there is one algorithm called an artificial neural network (ANN). Unlike the rest of the machine learning algorithms whose performance does not increase over a certain amount of training data, the multi-layered ANN showed an outstanding performance when a vast amount data is used to train, where the multi-layered ANN is called a multi-layer perceptron (MLP) or a deep neural network (DNN). Due to its outstanding performance, the DNN became very popular and the term "deep learning" is given to the DNN-based learning. Illustration of the ANN and DNN are shown in **Fig. 7** where the caret in the output layer refers to a predicted value. The ANN was developed inspired by a neuron in a human brain. Multiple ANNs form an ANN hidden layer and stacked ANN hidden layers make the DNN. As for the RL, the learning performance of the RL significantly improved when the RL was conducted based on the DNN.

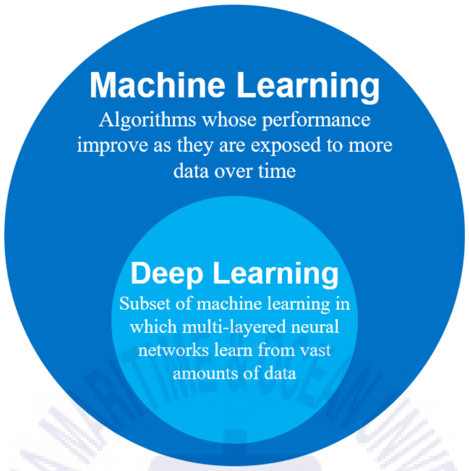

Machine Learning
Algorithms whose performance improve as they are exposed to more data over time
Deep Learning
Subset of machine learning in which multi-layered neural networks learn from vast amounts of data

Fig. 6 Definitions of the machine learning and deep learning

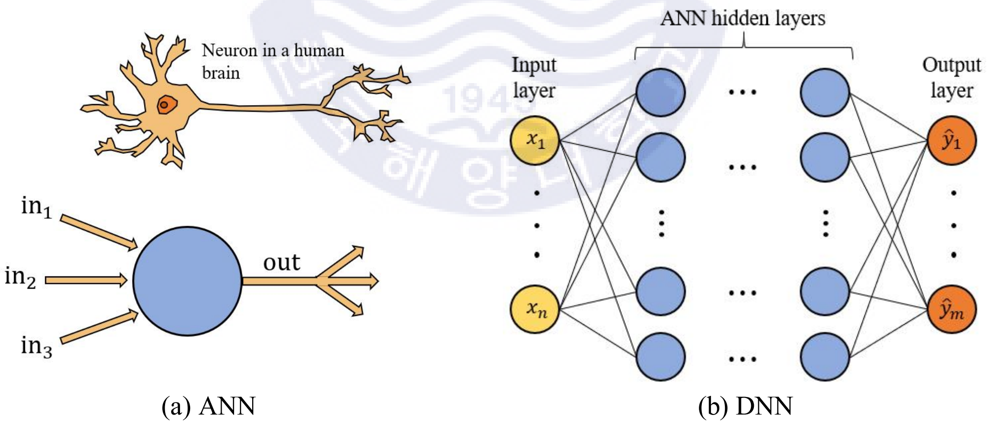


Fig. 7 Illustration of the ANN and DNN

As mentioned earlier, there are four studies performed in this paper: a) the mooring line tension prediction system, b) the feed-forward system for DPS, c) the RL-based adaptive PID controller for DPS, d) the automatic ship berthing system. In the mooring line tension prediction system, one data clustering machine learning algorithm and deep learning algorithms are used. In the feed-forward system for DPS, the deep learning algorithms are used. In the RL-based adaptive PID controller, the deep learning algorithms and an RL algorithm are used. In the automatic berthing system, the deep learning algorithms are used. In the next subchapter, details of the deep learning algorithms used in these studies are presented.

2.2. Deep Learning

The deep learning algorithms mainly consist of deep learning layers and the optimizers. One deep learning layer consists of multiple nodes and each node has an activation function that adds non-linearity in the system. Between the layers, the nodes are connected by edges which consist of learnable parameters that are updated by the optimizer during the training. The nodes and edges can be seen in **Fig. 7** (b) where calculation of the connecting edges between the two layers is performed by simple matrix calculation. To make a deep learning system, first, the deep learning system is built using the deep learning layers with appropriate activation function and weight initialization, then, it is trained with a training dataset using the optimizer. In this subchapter, the types of deep learning layers, activation function and weight initialization methods, and the optimizers are introduced. Then, important deep learning techniques such as a training dataset scaling and transfer learning are introduced.

2.2.1. Types of Deep Learning Layers

The most common deep learning layer is the ANN layer. Aside from it, other common types of layers are long short-term memory (LSTM) proposed by Hochreiter and Schmidhuber (1997), gated recurrent unit (GRU) proposed by Cho et al. (2014), convolutional neural network (CNN) proposed by Fukushima (1980). The LSTM and GRU layers are often used for sequential data processing and the CNN layer is often used for image processing. However, since only the ANN, LSTM, and GRU are used in the studies in this paper, only these three layers are explained in detail except for the CNN.

To explain the details of the ANN layer, first, calculation in one node of the ANN is shown in **Fig. 8**, Eq. (1) and Eq. (2), where x, w and b denote an input, the weight, and bias, respectively, g denotes the activation function, and $\circ$ denotes a dot product. One of the common activation functions is a tangent hyperbolic (tanh) which is shown in **Fig. 9**. The final output of one ANN node is a. The role of weight is to adjust steepness of the activation function and the role of bias is to shift the activation function in its x-axis.

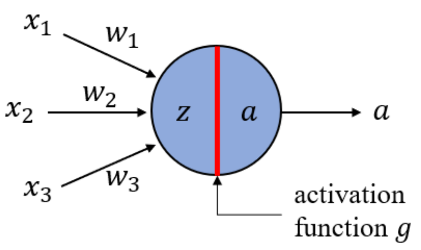


Fig. 8 Calculation in one node of the ANN

$$z = [x_1 \quad x_2 \quad x_3] \circ [w_1 \quad w_2 \quad w_3]^{\mathrm{T}} + [b_1] \tag{1}$$

$$a = g(z) \tag{2}$$

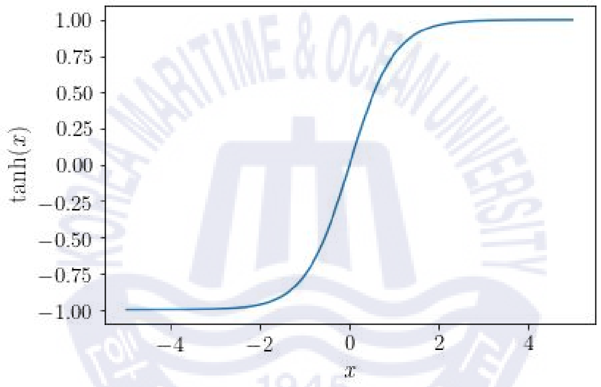


Fig. 9 Activation function of the tanh

Calculation of the multiple ANN hidden layers which form the DNN is presented in **Fig. 10** and Eqs. (3)-(6) where X, W, and B are shown in Eqs (7)-(13). The dimensions of X, W, and B are determined by (batch size, a number of input features), (size of a current layer, size of a next layer), and (batch size, size of the next layer), respectively. In **Fig. 10**, the batch size is one to consider one data only for a better understanding of the ANN. Depending on applications, the size of the hidden layer and the number of the hidden layers can be much larger, which would have a large number of weights and biases. The DNN is known to be capable of learning a good approximate function that relates an input and output by being trained with a large number of training data even when a relation between the input and output is highly non-linear. The size and depth of the DNN model determine how much complexity the model has to approximate the input-output relation, where the size refers to the size of hidden layers and the depth refers to the number of hidden layers. The higher complexity means the higher capability to learn at the cost of higher computational load for the training.

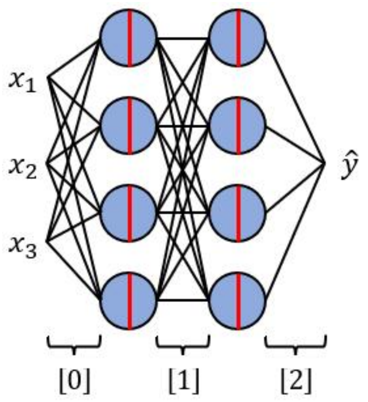


Fig. 10 Configuration of the DNN

$$a_0 = g\left(X \circ W^{[0]} + B^{[0]}\right) \tag{3}$$

$$a_1 = g\left(a_0 \circ W^{[1]} + B^{[1]}\right) \tag{4}$$

$$a_2 = g\left(a_1 \circ W^{[2]} + B^{[2]}\right) \tag{5}$$

$$\hat{y} = a_2 \tag{6}$$

$$X = [x_1 \quad x_2 \quad x_3] \tag{7}$$

$$W^{[0]} = \begin{bmatrix} w_{11}^{[0]} & w_{12}^{[0]} & w_{13}^{[0]} & w_{14}^{[0]} \\ w_{21}^{[0]} & w_{22}^{[0]} & w_{23}^{[0]} & w_{24}^{[0]} \\ w_{31}^{[0]} & w_{32}^{[0]} & w_{33}^{[0]} & w_{34}^{[0]} \end{bmatrix} \tag{8}$$

$$B^{[0]} = \left[b_1^{[0]} \quad b_2^{[0]} \quad b_3^{[0]} \quad b_4^{[0]}\right] \tag{9}$$

$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} & w_{14}^{[1]} \\ w_{21}^{[0]} & w_{22}^{[1]} & w_{23}^{[1]} & w_{24}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} & w_{34}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} & w_{43}^{[1]} & w_{44}^{[1]} \end{bmatrix} \tag{10}$$

$$B^{[1]} = \left[b_1^{[1]} \quad b_2^{[1]} \quad b_3^{[1]} \quad b_4^{[1]}\right] \tag{11}$$

$$W^{[2]} = \begin{bmatrix} w_{11}^{[1]} \\ w_{21}^{[2]} \\ w_{31}^{[3]} \\ w_{41}^{[4]} \end{bmatrix} \tag{12}$$

$$B^{[2]} = \left[b_1^{[2]}\right] \tag{13}$$

The ANN is a very powerful learning algorithm. However, when it comes to processing the sequential data, it is somewhat limited. For example, if an input is the sequential data with its length of 100, the size of the ANN hidden layer must be large enough to process the sequential input data. If the length of the sequential input data is even longer such as 500, the size of the ANN hidden layer must be even larger. Therefore, the ANN is inefficient in terms of the model size when processing the sequential data. Another main disadvantage is that once the ANN model is built and trained, the model's architecture cannot be changed, meaning that the length of the sequence that the ANN model can process is fixed. It can be quite inefficient because a flexible length may be required in some applications such as text generation where a length of sentence varies. To overcome these limitations, the LSTM and GRU are often used.

Before introducing the details of the LSTM and GRU layers, their base layer called a recurrent neural network (RNN) is first introduced in detail. The LSTM and GRU were developed to improve the RNN's performance. Therefore, the basic concept of the LSTM, and GRU can easily be explained in the RNN. Configurations of the RNN layer and RNN cell are shown in **Fig. 11** and **Fig. 12**, respectively. The subscript $< t >$ denotes a timestep, T denotes the end of the timestep, a is called a hidden state which carries the temporal information throughout the RNN cells. Equations for the RNN cell are shown in Eqs. (14)-(15). In Eq. (14), $[a^{<t-1>}, x^{<t>}]$ is a concatenated vector of $a^{<t-1>}$ and $x^{<t>}$. An activation function can be added to $a^{<t>}$ to yield a desired output range or the ANN layer can be added to $a^{<t>}$ to give higher learning capability to the RNN model when outputting $\hat{y}^{<t>}$.

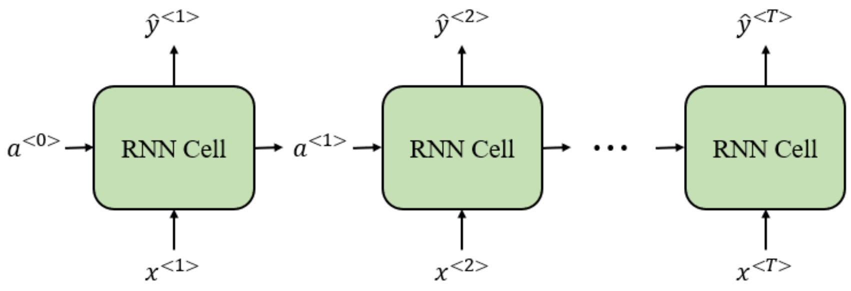


Fig. 11 Configuration of the RNN layer

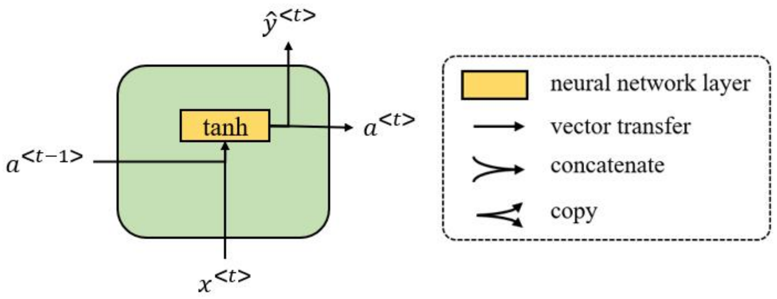
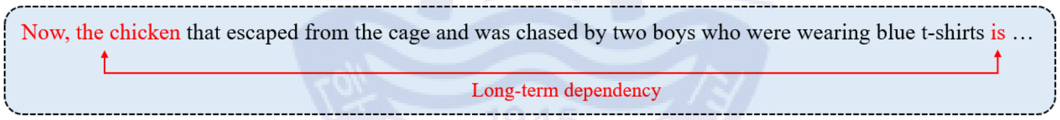


Fig. 12 Configuration of RNN cell

$$a^{<t>} = \tanh(W_a \circ [a^{<t-1>}, x^{<t>}] + b_a) \tag{14}$$

$$\hat{y}^{<t>} = a^{<t>} \tag{15}$$

One main disadvantage of the RNN is that it cannot process long sequential data, meaning that it cannot learn a long-term dependency in the sequential data. An example of the long-term dependency is shown in **Fig. 13**. To overcome this limitation, the LSTM was developed and it is capable of learning the long-term dependency.


Now, the chicken that escaped from the cage and was chased by two boys who were wearing blue t-shirts is …

Long-term dependency

Fig. 13 Example of the long-term dependency

The overall structure of LSTM layer is quite similar to the RNN. The LSTM layer is a collection of LSTM cells that are connected in sequential order. The sequentially connected LSTM cells process a sequential input to yield an output or a sequential output while maintaining its hidden state and a cell state. The cell state was introduced in addition to the hidden state to provide an ability to process long sequential data better. Configurations of the LSTM layer and the LSTM cell are shown in **Fig. 14** and **Fig. 15**, respectively, where a and c refer to the hidden state and cell state of the LSTM, respectively. In **Fig. 15**, it is well shown that the cell state goes straight from the left side to the right side. By doing so, the cell state plays an important role in carrying the temporal information over a long sequence. Equations for the LSTM cell are shown in Eqs. (16)-(22) where $*$ denotes element-wise multiplication. $\tilde{c}$ is a cell state candidate, and, Γ_u, Γ_f, and Γ_o are an update gate, forget gate, and output gate, respectively. The important components in the LSTM cell are Γ_u, Γ_f, and Γ_o. Γ_u

determines how much current information to keep because the new temporal information may be important for the future reference. Γ_f determines how much past information to forget because some temporal information from the past may no longer be important. Γ_o determines how much cell state to output. Finally, similar to the RNN, an activation function can be added to $a^{<t>}$ to yield a desired output range or the ANN layer can be added to $a^{<t>}$ to give higher learning capability to the LSTM model when outputting $\hat{y}^{<t>}$.

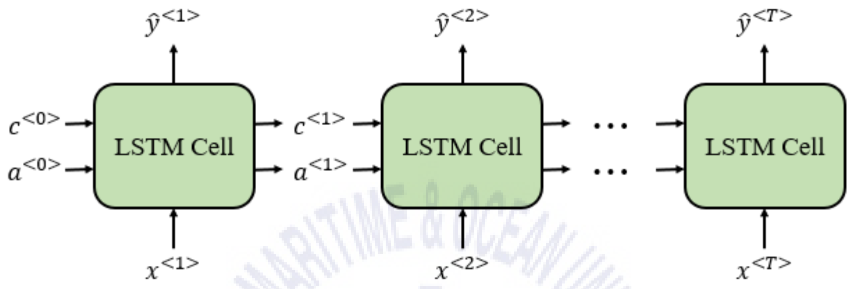


Fig. 14 Configuration of the LSTM layer

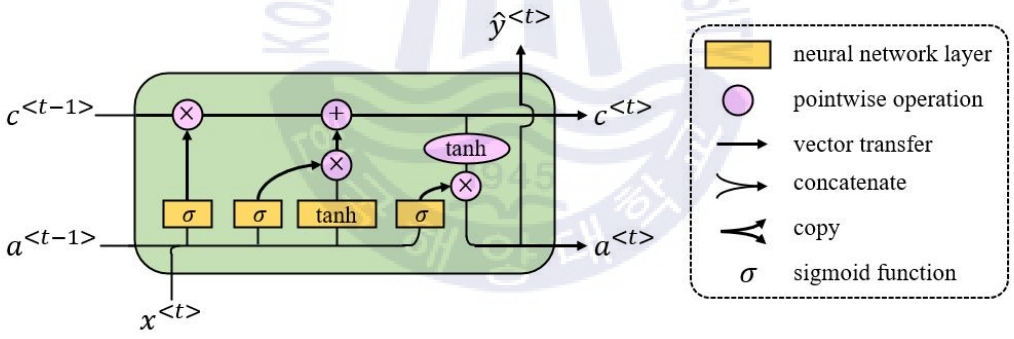


Fig. 15 Configuration of the LSTM cell

$$\tilde{c}^{<t>} = \tanh(W_c \circ [a^{<t-1>}, x^{<t>}] + b_c) \tag{16}$$

$$\Gamma_u = \sigma(W_u \circ [a^{<t-1>}, x^{<t>}] + b_u) \tag{17}$$

$$\Gamma_f = \sigma\left(W_f \circ [a^{<t-1>}, x^{<t>}] + b_f\right) \tag{18}$$

$$\Gamma_o = \sigma(W_o \circ [a^{<t-1>}, x^{<t>}] + b_o) \tag{19}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \tag{20}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>}) \tag{21}$$

$$\hat{y}^{<t>} = a^{<t>} \tag{22}$$

In many applications, the LSTM showed great performance for processing the long-term sequential data. But its architecture was quite complex and the complex architecture caused a high computational cost for the training and took a large computational memory. To simplify the LSTM's architecture, the GRU was developed. The main advantage of the GRU over the LSTM is a simpler architecture which reduces computational cost and memory. In theory, performance of the GRU is lower than that of the LSTM due to its simpler architecture. But, in practice, the performance of the GRU is mostly the same as that of the LSTM in most of the applications although the LSTM outperforms the GRU in some applications. Configuration of the GRU layer and the GRU cell are shown in **Fig. 16** and **Fig. 17**, respectively. Equations for the GRU cell are shown in Eqs. (23)-(27). Γ_u is the update gate and decides what information to throw away and what new information to add. Γ_r is a reset gate which decides how much past information to forget. Unlike the LSTM having the three gates (update gate, forget gate, output gate), the GRU has only two gates (update gate, reset gate) and it shows that the GRU has a simpler architecture than the LSTM. Finally, similar to the RNN and LSTM, an activation function can be added to $c^{<t>}$ to yield a desired output range or the ANN layer can be added to $c^{<t>}$ to give higher learning capability to the GRU model when outputting $\hat{y}^{<t>}$.

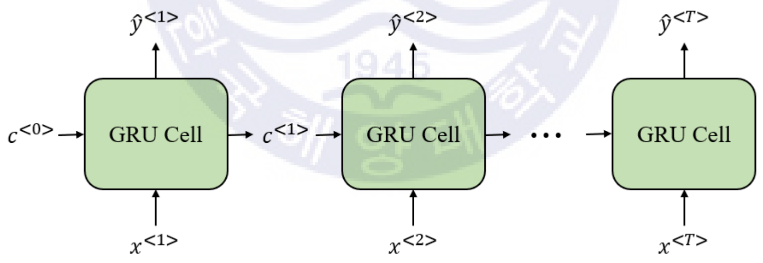


Fig. 16 Configuration of the GRU layer

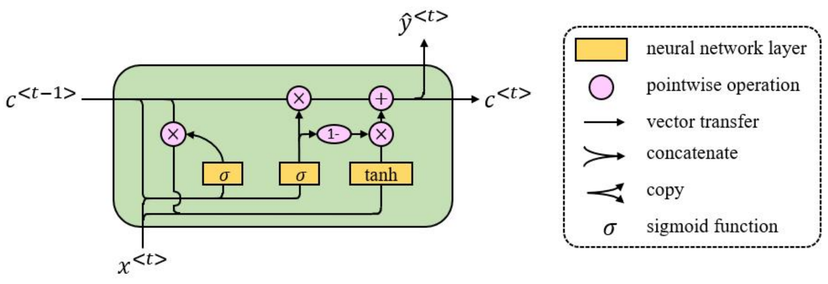


Fig. 17 Configuration of the GRU cell

$$\tilde{c}^{<t>} = \tanh(W_c \circ [\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \quad (23)$$

$$\Gamma_u = \sigma(W_u \circ [c^{<t-1>}, x^{<t>}] + b_u) \quad (24)$$

$$\Gamma_r = \sigma(W_r \circ [c^{<t-1>}, x^{<t>}] + b_r) \quad (25)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \quad (26)$$

$$\hat{y}^{<t>} = c^{<t>} \quad (27)$$

Aside from the ANN, LSTM, and GRU layers, there is a special type of layer that can prevent an overfitting problem. That layer is called a dropout layer proposed by Srivastava et al. (2014). The overfitting problem is a problem that the deep learning system becomes too fit to its training dataset during the training and results in poor performance on a test dataset. An illustration of the overfitting problem compared to underfitting and appropriate fitting is shown in **Fig. 18**. It is shown that when the overfitting problem occurs, the learning system loses its generalization over the data. The dropout can prevent this overfitting problem. The main functionality of the dropout is to randomly ignore some neurons in the neural network layers during the training. By doing so, the dropout forces the neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons and it leads the neural network to general understanding in a given dataset. Due to its simplicity for implementation and robustness, it has been widely adopted in many applications of the neural network. An illustration of the dropout is shown in **Fig. 19**.

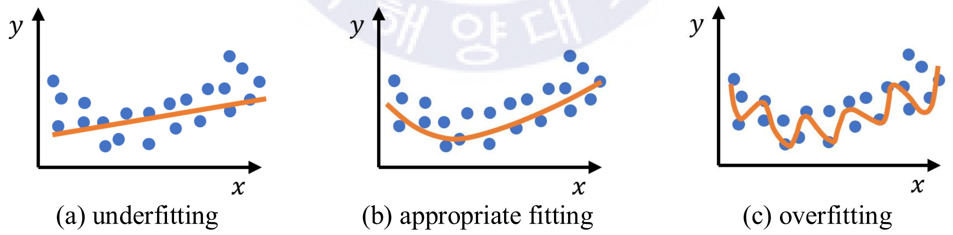
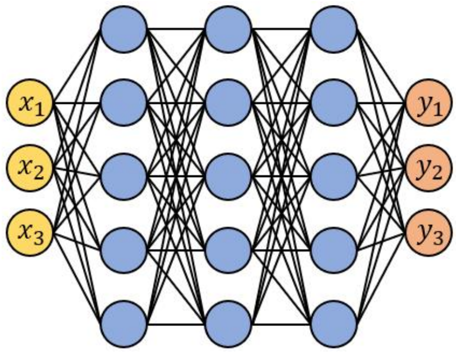
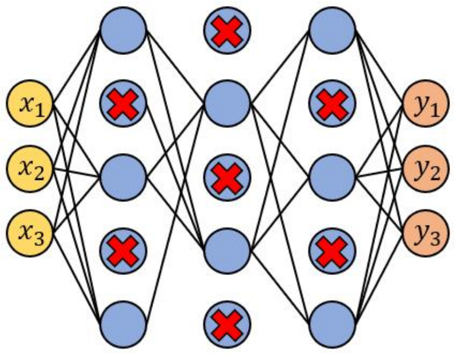


Fig. 18 Illustration of the overfitting problem compared to underfitting and appropriate fitting



(a) standard DNN



(b) standard DNN with the dropout

Fig. 19 Illustration of the dropout

There is another type of layer that is designed to prevent an internal covariance shift problem. That layer is called a batch normalization (BN) layer proposed by Ioffe and Szegedy (2015). The internal covariate shift problem is a problem that the input distribution in each hidden layer varies much, which may cause the unstable training such as a gradient vanishing or gradient exploding. Before the BN was proposed, methods such as careful weight initialization and small learning rate were used to prevent the unstable training. The main functionality of the BN is to normalize, re-scale, and re-center an output from a previous hidden layer. A procedure of the BN during the training is shown in **Table 1** where ϵ is a small number such as 1e-3. During the test and inference, the BN works a bit differently. Instead of using the mini-batch mean and mini-batch variance to normalize, a moving mean and moving variance are used, where the moving mean and variance can be obtained by applying a moving average filter on the mini-batch means and mini-batch variances over epochs. There are two main advantages of the BN. First, the training with the BN tends to converge faster and more stably, and the faster training with a higher learning rate is possible without falling into the unstable training because the input distribution in the hidden layers is stabilized by the normalization. Second, the BN prevents the overfitting problem to some extent since a constant change of the mini-batch mean and variance depending on the selected mini-batch at every training step provides a noise in the learning system during the training, which forces the learning system to learn robust features that can generalize the given training data.

Table 1 Procedure of the batch normalization during the training

```
Input:
- Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1, x_2, \cdots, x_m\}$
- Output from a previous hidden layer: $z$
- Parameters to be learned: $\gamma, \beta$
Output: $\{y = \mathrm{BN}_{\gamma,\beta}(\mathrm{z})\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$     // mini-batch variance
$\hat{z} \leftarrow \frac{z - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$                   // normalize
$y \leftarrow \gamma\hat{z} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(\mathrm{z})$    // scale and shift
```

2.2.2. Activation Function and Weight Initialization Methods

Choosing both an appropriate activation function and weight initialization method is quite important in building the deep learning system. The training performance and training speed can be greatly affected by these two parameters. For the activation function, many previous studies used a sigmoid function. However, it is known that the sigmoid function intrinsically has some disadvantages in training neural networks. The drawbacks of the sigmoid function are sharp damp gradients during backpropagation, gradient saturation, slow convergence, and nonzero-centered output. Thus, many studies have been conducted to develop better activation functions to overcome the problems of the sigmoid function. The activation functions that are popularly used these days are the tangent hyperbolic and rectified linear unit (ReLU) functions proposed by Hahnioser et al. (2000). The tangent hyperbolic function solves the problem of the limitation in the gradient update direction that exists in the sigmoid function, and results in faster training speed. The ReLU function avoids the gradient vanishing problem and lowers the complexity of the calculations as its derivative function is relatively simple. The equations for the sigmoid, tangent hyperbolic, and ReLU functions are shown in Eqs. (28)-(30):

$$\mathrm{sigmoid}(x) = 1/(1 + e^{-x}) \tag{28}$$

$$\tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \tag{29}$$

$$\mathrm{relu}(x) = \begin{cases} x; \text{ if } x \geq 0 \\ 0; \text{ otherwise} \end{cases} \tag{30}$$

For the weight initialization method, a normal distribution with a mean of zero and a standard deviation of 0.01 was mostly used. Later, however, many studies showed that the weight initialization method could greatly affect the training performance, and several other weight initialization methods have been proposed. Currently, the two most popular methods are a Xavier (Glorot) weight initialization method (Glorot and Bengio, 2010) and a He weight initialization method (He et al., 2015). The Xavier initialization is known to result in good training performance with the sigmoid and tanh functions and the He initialization is known to be very compatible with the ReLU function. The equations for the Xavier and He initialization methods are based on a normal distribution where a mean value is set to zero and variance is set to Eq. (31) for the Xavier initialization and Eq. (32) for the He initialization. n_{in} and n_{out} represent the numbers of nodes in the previous and next layers.

$$\sigma^2_{\mathrm{Xavier}} = \sqrt{2/(n_{\mathrm{in}} + n_{\mathrm{out}})} \tag{31}$$

$$\sigma^2_{\mathrm{He}} = \sqrt{2/n_{\mathrm{in}}} \tag{32}$$

2.2.3. Optimizers

The optimizers are used to train the deep learning system by updating the weights and biases. The update is performed based on a loss value that comes from a loss function, where the loss value acts as a guide for the optimizer to decide how to update. One of the most common loss functions is a mean squared error (MSE) which is used for the regression task mostly. The equation of MSE is shown in Eq. (33) where n denotes a number of data, y_i and $\hat{y}_i$ denote an actual output value and a predicted output value, respectively. It can be seen that the larger the gap between y_i and $\hat{y}_i$ is, the greater the MSE value becomes. The optimizer updates the weights and biases by minimizing the MSE loss function in case of the regression task. This updating process is the training process of the deep learning system.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{33}$$

The most basic optimizer is a gradient descent algorithm. The gradient descent minimizes the loss value by iteratively updating the parameters (weights and biases) in a direction of a negative gradient of the loss function. The parameter update by the gradient descent is shown in Eq. (34) where θ denotes the parameter, t denotes time, η denotes a learning rate, and $\nabla_\theta J(\theta; x, y)$ denotes the gradient of the loss function with respect to the parameter θ given the training data (x, y). To help understanding in the gradient descent, an illustration of the gradient descent is shown in **Fig. 20** where there is one weight w to update in an x axis and the loss function with respect to w, $J_w(w)$, is represented in an y axis. In **Fig. 20**, starting from the initial point, the first step is taken in the direction of negative gradient of the loss function with respect to w, $-\nabla_\mathrm{w} J(w)$. Then, the next steps are continuously taken in the same fashion to reach a global loss minimum, where the loss value is minimized. The minimized loss value means the minimized gap between the actual data and predicted data, therefore, the training of the learning system is completed when the loss is minimized. The length of each step which is represented as an orange arrow is determined by multiplication of the gradient of the loss function and the learning rate. Thus, the learning rate determines how much update is conducted in one step, and the amount of update decreases as the loss value gets close to the global loss minimum because the loss gradients nearby the global loss minimum are small. Another illustration of the gradient descent is shown in **Fig. 21** where there are two weights w_1 and w_2 and the global loss minimum is located in the very center. In both **Fig. 20** and **Fig. 21**, it takes multiple steps to reach the global loss minimum and one might want to increase the learning rate so that the convergence occurs faster. However, the learning rate must be determined appropriately because some learning problems may happen otherwise as shown in **Fig. 22**. When the learning rate is too low, the learning requires many update steps before reaching the global loss minimum. When the learning rate is too large, it causes the dramatic updates which leads to divergent behaviors. That is the reason that the learning rate must be set appropriately so that the learning can be conducted as **Fig. 22** (b).

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t; x, y) \tag{34}$$

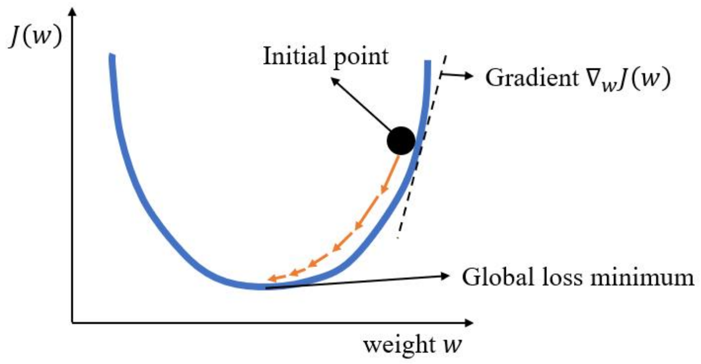


Fig. 20 Illustration of the gradient descent

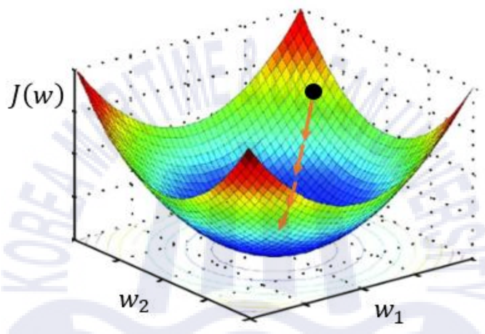


Fig. 21 Illustration of the gradient descent with respect to two weights

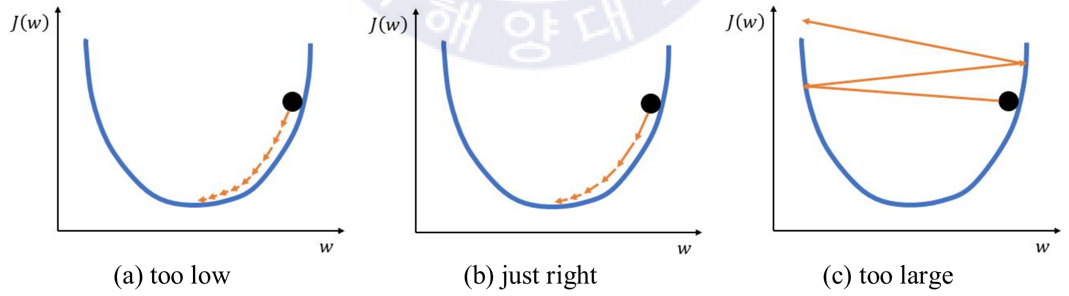
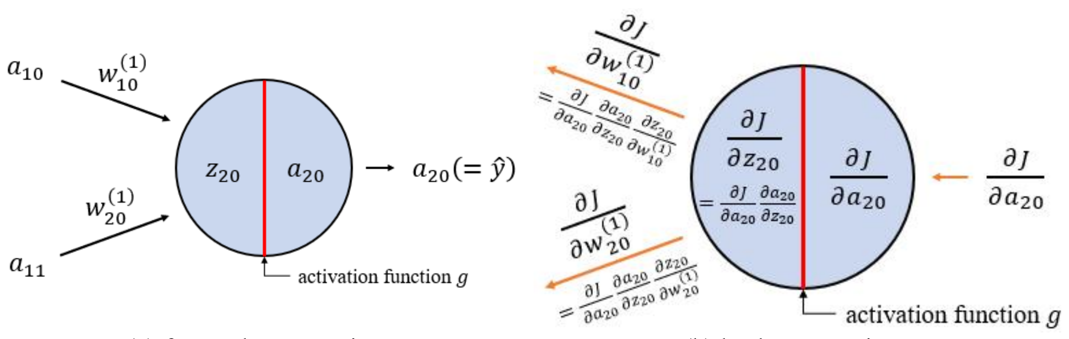


Fig. 22 Effects of the learning rate

The parameters (weights, biases) are updated by the gradient descent in the direction of the negative gradient of the loss function with respect to each parameter, and there are a large number of parameters in the deep learning system. It means the gradient of the loss function with respect to every parameter must be known for the update. An algorithm to calculate $\nabla_\theta J(\theta)$ for all parameters is called backpropagation. The backpropagation does it by backpropagating $\nabla_\theta J(\theta)$ in the last hidden layer based on a chain rule. An illustration of the backpropagation in one neuron in the last hidden layer compared to forward-propagation is shown in **Fig. 23** where the forward-propagation is the simple matrix calculation to yield the predicted output $\hat{y}$, and $\partial J/\partial w$ is the same as $\nabla_\mathrm{w} J(w)$. $\partial J/\partial w$ is called a first-order partial derivative of the loss function with respect to the weight. Through the forward-propagation, a loss function value J can be calculated. Using the calculated loss value, $\frac{\partial J}{\partial a_{20}}$ can be calculated. Then, $\frac{\partial J}{\partial z_{20}}$ can be calculated by redefining it to $\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}$ using the chain rule, where $\frac{\partial J}{\partial a_{20}}$ is already calculated in the previous step and $\frac{\partial a_{20}}{\partial z_{20}}$ can be obtained by calculating $\frac{\partial g(z_{20})}{\partial z_{20}}$ since a_{20} is equal to $g(z_{20})$. And then, $\frac{\partial J}{\partial w_{10}^{(1)}}$ and $\frac{\partial J}{\partial w_{20}^{(1)}}$ can be calculated by redefining them to $\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial w_{10}^{(1)}}$ and $\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial w_{20}^{(1)}}$, respectively, where $\frac{\partial J}{\partial a_{20}}$ and $\frac{\partial a_{20}}{\partial z_{20}}$ are already calculated in the previous steps, and $\frac{\partial z_{20}}{\partial w_{10}^{(1)}}$ and $\frac{\partial z_{20}}{\partial w_{20}^{(1)}}$ can be obtained by calculating $\frac{\partial\left(w_{10}^{(1)}a_{10}+w_{20}^{(1)}a_{20}\right)}{\partial w_{10}^{(1)}}$ and $\frac{\partial\left(w_{10}^{(1)}a_{10}+w_{20}^{(1)}a_{20}\right)}{\partial w_{20}^{(1)}}$, respectively. Finally, the weight updates for $w_{10}^{(1)}$ and $w_{20}^{(1)}$ can be performed as Eq. (35) and Eq. (36), respectively, where the first-order partial derivatives in the learning rate terms can be represented as Eq. (37) and Eq. (38), respectively. It is how the learning is performed in the neural network in general. As shown in **Fig. 23** (b), the calculation of the first-order partial derivative is conducted backwards, from the right side to the left side in order. That is the reason why the backpropagation has "back" in its term. To help understanding in the backpropagation in the DNN, an illustration of the backpropagation in the DNN is shown in **Fig. 24**.



a_{10}
$w_{10}^{(1)}$
$w_{20}^{(1)}$
a_{11}
z_{20}
a_{20}
$a_{20}(=\hat{y})$
activation function g
$\frac{\partial J}{\partial w_{10}^{(1)}}$
$=\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial w_{10}^{(1)}}$
$\frac{\partial J}{\partial w_{20}^{(1)}}$
$=\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial w_{20}^{(1)}}$
$\frac{\partial J}{\partial z_{20}}$
$=\frac{\partial J}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}$
$\frac{\partial J}{\partial a_{20}}$
$\frac{\partial J}{\partial a_{20}}$
activation function g

(a) forward-propagation (b) backpropagation

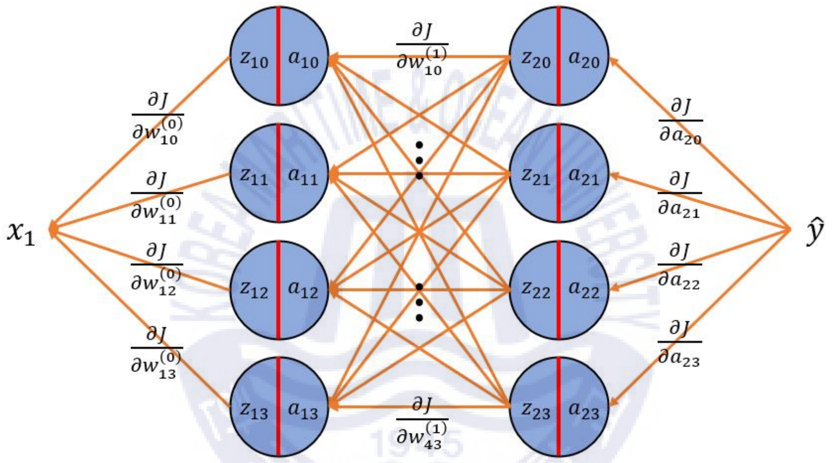
Fig. 23 Illustration of the backpropagation in one neuron in the last hidden layer compared to forward-propagation

$$w_{10}^{(1)} = w_{10}^{(1)} - \eta \frac{\partial J}{\partial w_{10}^{(1)}} \tag{35}$$

$$w_{20}^{(1)} = w_{20}^{(1)} - \eta \frac{\partial J}{\partial w_{20}^{(1)}} \tag{36}$$

$$\frac{\partial J}{\partial w_{10}^{(1)}} = \frac{\partial J}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial w_{10}^{(1)}} \tag{37}$$

$$\frac{\partial J}{\partial w_{20}^{(1)}} = \frac{\partial J}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial w_{20}^{(1)}} \tag{38}$$



x_1
$\frac{\partial J}{\partial w_{10}^{(0)}}$
$\frac{\partial J}{\partial w_{11}^{(0)}}$
$\frac{\partial J}{\partial w_{12}^{(0)}}$
$\frac{\partial J}{\partial w_{13}^{(0)}}$
z_{10} a_{10}
z_{11} a_{11}
z_{12} a_{12}
z_{13} a_{13}
$\frac{\partial J}{\partial w_{10}^{(1)}}$
$\frac{\partial J}{\partial w_{43}^{(1)}}$
z_{20} a_{20}
z_{21} a_{21}
z_{22} a_{22}
z_{23} a_{23}
$\frac{\partial J}{\partial a_{20}}$
$\frac{\partial J}{\partial a_{21}}$
$\frac{\partial J}{\partial a_{22}}$
$\frac{\partial J}{\partial a_{23}}$
$\hat{y}$

Fig. 24 Illustration of the backpropagation in the DNN

Using the backpropagation, the training of the deep learning system can be described as follows: 1) calculate the loss function through the forward-propagation, 2) update the parameters by the gradient descent using $\partial J/\partial\theta$ obtained by the backpropagation. For the big DNN, a lot of calculation for $\partial J/\partial\theta$ is required because there are a large number of parameters. This can be very difficult to do manually. Therefore, automatic differentiation is typically used. One of the popular open-source libraries that provides the powerful automatic differentiation is Tensorflow developed by a Google Brain team.

As mentioned earlier, one deep learning system has thousands or sometimes millions of parameters, which means that the optimizer needs to find the global minimum in a very high n dimensional space where there are n parameters. In that case, the learning can be difficult for the basic gradient descent. To improve the learning performance of the gradient descent, many other optimizers were developed such as the gradient descent with momentum (Rumelhart, Hinton and Williams, 1986), Adagrad (Duchi, Hazan and Singer, 2011), RMSProp (Hinton, Srivastava and Swersky, 2012), and Adam (Kingma and Ba, 2015). Among the developed optimizers, the Adam is the most popular one and only this optimizer is used in the studies in this paper. However, the Adam was developed based on several previous optimizers such as the gradient descent with momentum, Adagrad, and RMSProp. Therefore, the momentum, Adagrad, and RMSProp must be understood to understand the Adam.

The gradient descent with the momentum is basically the basic gradient descent with the momentum applied. A basic notion of the momentum is to give a kind of inertia force to the parameter update by the gradient descent so that the parameter update goes a bit further in the direction it was headed to. The parameter update by the gradient descent with the momentum is represented as Eqs. (39)-(40) where γ is a momentum value and it is usually set to 0.9. The detailed effect of v_t can be seen in Eq. (41). It can be seen that the use of v_t allows the parameter update to be conducted considering the weighted previous gradients. The advantages of the momentum are a reduction of oscillations and high variance of the parameters during the update and the faster training.

$$\theta_{t+1} = \theta_t - v_t \tag{39}$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t) \tag{40}$$

$$v_t = \eta \nabla_\theta J(\theta_t)_t + \gamma \eta \nabla_\theta J(\theta_{t-1})_{t-1} + \gamma^2 \eta \nabla_\theta J(\theta_{t-2})_{t-2} + \cdots \tag{41}$$

Unlike the basic gradient descent where the learning rate is the same for all the parameters, the Adagrad adaptively changes its learning rate. The basic idea in the Adagrad is that the low learning rate is applied to the parameters that have been updated much, and the high learning rate is applied to the parameters that have not been updated much because the parameters that have been updated much are likely to be nearby their optimum points, whereas, the parameters that have not been updated much are likely to need to be updated a lot to reach their optimum points. The parameter update by the Adagrad is represented as Eqs. (42)-(43) where G_t is a sum of squares of the previous gradients and ϵ is a small number such as 1e-8. It can be seen that the learning rate is adaptively changed by $1/\sqrt{G_t + \epsilon}$. During the training by the Adagrad, each parameter is updated to some extent, then, G_t increases accordingly. Given the magnitude of G_t of each parameter, the learning rate $\eta/\sqrt{G_t + \epsilon}$ for each parameter is determined. If the magnitude of G_t of one parameter is high, the learning rate is low, whereas, if the magnitude of G_t of one parameter is low, the learning rate is high. There are three main advantages of the Adagrad. First, the learning rate adaptively changes for each parameter.

Second, the learning rate does not have to be tuned manually. Third, the training can be effectively performed even on sparse data.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t) \tag{42}$$

$$G_t = G_{t-1} + \left(\nabla_\theta J(\theta_t)\right)^2 \tag{43}$$

Despite the advantages of the Adagrad, it has a limitation. For the Adagrad, as the training continues, G_t continues to increase because the squares of the previous gradients do not decay over time but stay in G_t. This makes the learning almost stop by the time G_t is too big. The stop of learning occurs inevitably in the Adagrad. To overcome this limitation, the RMSProp was proposed. In the RMSProp, the way of calculating G_t is different. It is changed in a way that the squares of the previous gradients decay over time so that the learning rate $\eta/\sqrt{G_t + \epsilon}$ does not converge to almost zero over time. The parameter update by the RMSProp is represented as Eqs. (44)-(45) where γ is an exponential decay rate which is usually set to 0.9, and the way of calculation for G_t is different from the Adagrad. The calculation form of Eq. (45) is from an exponential moving average. By calculating G_t as Eq. (45), G_t does not become excessively big, and different significances can be put to the squares of the previous gradients where higher significances are applied to the recent gradients and lower significances are applied to the past gradients.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t) \tag{44}$$

$$G_t = \gamma G_{t-1} + (1-\gamma)\left(\nabla_\theta J(\theta_t)\right)^2 \tag{45}$$

The Adam can be viewed as a combination of the RMSProp and the momentum. In the Adam, there are two important components that are shown in Eq. (46) and Eq. (47), respectively. m_t and v_t are called a biased first moment estimate and a biased second moment estimate, respectively, and β_1 and β_2 are the exponential decay rates for m_t and v_t, respectively. For settings of the parameters, m_t and v_t are initialized to zero, and β_1 and β_2 are usually set to 0.9 and 0.999, respectively. The calculation of m_t is similar to that of the momentum from Eq. (40), and the calculation of v_t is similar to that of the RMSProp from Eq. (45). But, since the two biased moment estimates are initialized to zero, they are too small at the beginning of the training. A bias-corrected first moment estimate and a bias-corrected first moment estimate compensate for that issue. The two bias-corrected moment estimates are shown in Eq. (48) and Eq. (49), respectively. It should be noted that β_i^t denotes β_i to the power of time t. Using the two bias-corrected moment estimates, the parameter update by the Adam can be represented as Eq. (50). The learning performance of the Adam has been proven to be very robust in a variety of applications. Therefore, the Adam is adopted as the optimizer for the studies in this paper.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta_t) \tag{46}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\big(\nabla_\theta J(\theta_t)\big)^2 \tag{47}$$

$$\hat{m}_t = m_t/(1-\beta_1^t) \tag{48}$$

$$\hat{v}_t = v_t/(1-\beta_2^t) \tag{49}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t \tag{50}$$

2.2.4. Training Dataset Scaling

The training dataset consists of multiple sets of the input and output data in which the input and output data are the feature vectors as shown in Eq. (51) and Eq. (52), where X_i and Y_i denote the *i*-th input feature vector and output feature vector, respectively, and x_n and y_n are the features in the feature vectors. Thus, the training dataset can be represented by Eq. (53) and Eq. (54) where X' and Y' denote the training dataset for the input and output, respectively. Thus, the training dataset scaling refers to application of some scaling method to the training dataset (X', Y').

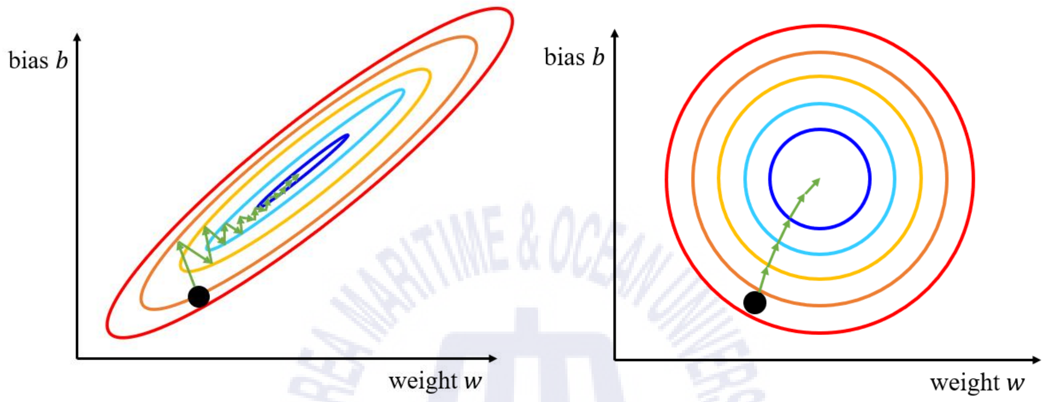
$$X_i = \{x_0, x_1, \cdots, x_n\} \tag{51}$$

$$Y_i = \{y_0, y_1, \cdots, y_n\} \tag{52}$$

$$X' = \{X_0, X_1, \cdots, X_n\} \tag{53}$$

$$Y' = \{Y_0, Y_1, \cdots, Y_n\} \tag{54}$$

There are very strong reasons that the training dataset should be scaled. The first reason is to match the distributions and scales of the features in the input and output vectors. The different distributions and scale in the feature vector deteriorate the learning process because the parameters in the neural network get to react more sensitively toward one feature than another. An example of the input feature vector with the different distribution and scale can be $\{\text{age}, \#\text{hairs}\}$ where the $\#\text{hairs}$ denotes a number of hairs. An illustration of the effect of different distribution and scale of the input feature vector during the training is shown in **Fig. 25** where the circles and elongated circles represent a contour of the loss function value, the black circle denotes the initial point, and the green arrow denotes the parameter update direction. It is shown that the training is more difficult when the input feature vector is not scaled, whereas the training is easier when the input feature vector is properly scaled.



(a) on an unscaled input feature vector (b) on a scaled input feature vector

Fig. 25 Illustration of the effect of different distribution and scale of the input feature vector during the training

The second reason for the scaling is to make the scale of input and output vectors small so that the weights and biases are updated to small values. If the scale of the input or output vectors is large, it causes the weights and biases to be updated to large values, which causes the unstable learning and poor performance because the neural network is now too sensitive to changes of the input. In terms of the backpropagation for the parameter update, if the scale of the output vector is large, it can result in the large loss value, then this causes a dramatic update of the parameters because the large loss value is backpropagated to the parameters to update them. In turn, this dramatic update of the parameters is likely to make the learning process unstable. For these reasons, the scaling of the training dataset is almost always conducted in practice.

There are mainly two types of scaling method. The first one is a standard scaling method, and the second one is a min-max scaling method. An equation of each scaling method is shown in Eq. (55) and Eq. (56), respectively, where μ and σ denote a mean and standard deviation, respectively. It should be noted that $\mu(.)$, $\sigma(.)$, $\min(.)$, $\max(.)$ are calculated for each feature across the entire dataset, therefore, the size of output by these operations is (1 $\times$ a number of features in the feature vector). These two scaling methods can be applied to the training dataset for the output in the same way.

$$X'_{\text{scaled}} = \frac{X' - \mu(X')}{\sigma(X')} \tag{55}$$

$$X'_{\text{scaled}} = \frac{X' - \min(X')}{\max(X') - \min(X')} \tag{56}$$

2.2.5. Transfer Learning

The transfer learning is the deep learning technique that utilizes learned weights from a trained neural network to train a new neural network faster with a much smaller training dataset. A comparison between the traditional learning and the transfer learning is shown in **Fig. 26**. With the traditional learning, a large amount of training dataset is required to train each learning system, therefore, if there are two learning systems, two large amounts of training datasets are required. But, with the transfer learning, a new learning system can be trained by utilizing the learned knowledge (learned weights and biases) from the trained learning system, and it results in faster training of the new learning system and requires a much smaller training dataset.

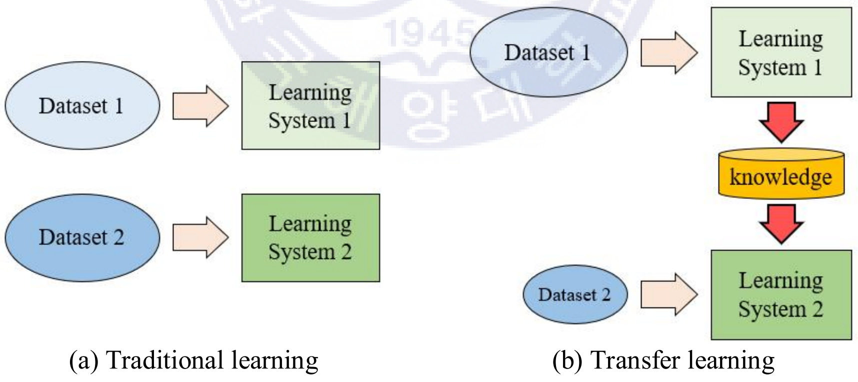


Fig. 26 Comparison between the traditional learning and the transfer learning

2.3. Reinforcement Learning

There are a large number of RL algorithms. **Fig. 27** shows a diagram of the major RL algorithms, where the green oval denotes a category of the RL algorithms and the yellow rectangle denotes a type of the RL algorithm. Aside from the RL algorithms presented in the figure, there are more RL algorithms such as a distributional DQN, rainbow DQN, D4PG, ACER, ACTKR, and so on. Since the RL algorithms are still actively studied and developed, there will certainly be more advanced RL algorithms in the future. This subchapter is mostly focused on explaining the DDPG (Lillicrap et al., 2016) because it is the RL algorithm used in one of the studies in this paper. To explain the DDPG, the base RL algorithms of the DDPG are first introduced, then, the details of DDPG are addressed.

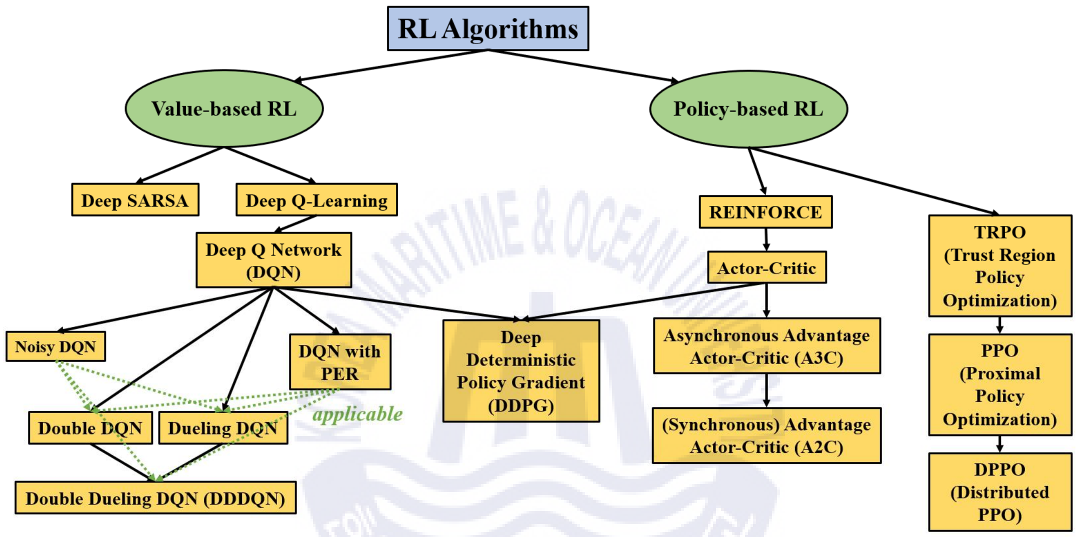

RL Algorithms
Value-based RL
Policy-based RL
Deep SARSA
Deep Q-Learning
Deep Q Network (DQN)
Noisy DQN
DQN with PER
Double DQN
Dueling DQN
applicable
Double Dueling DQN (DDDQN)
Deep Deterministic Policy Gradient (DDPG)
REINFORCE
Actor-Critic
Asynchronous Advantage Actor-Critic (A3C)
(Synchronous) Advantage Actor-Critic (A2C)
TRPO (Trust Region Policy Optimization)
PPO (Proximal Policy Optimization)
DPPO (Distributed PPO)

Fig. 27 Diagram of the major RL algorithms

In the standard reinforcement learning, there is an agent, A, and an environment, E, that that the agent interacts with in discrete time steps. At every timestep, the agent receives a state s_t, and takes an action a_t, and obtains a reward r_{t+1}, forming a sequential transition $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, \cdots)$ throughout an episode. An action a_t is determined by a policy π which maps a state s_t to probabilistic values of each action $\mathcal{P}(a_t)$ because an environment E may be stochastic. The learning process of the agent can be defined mathematically using a Markov Decision Process (MDP). A generic flowchart of the process is shown in **Fig. 28**. In the figure, as the agent continues to interact with an environment, the policy π eventually converges to an optimal policy π_*.

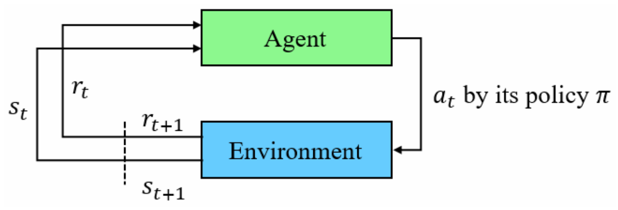


Fig. 28 Learning process of an agent with MDP

In many RL algorithms, an action-value function, so-called Q function, is used in the learning process. The Q function describes the expected return value G_t which is expressed as Eq. (57) where γ is a discounting factor to consider influence of uncertainties towards the future after taking an action a_t in a state s_t and thereafter following the policy π. Thus, the Q function simply tells how good a certain action is at a certain state. The Q function can be written in a recursive form as Eq. (58) where $\mathbb{E}[.]$ denotes an expectation value. This recursive form is called a Bellman equation, and Eq. (58) is called the Bellman equation of the Q function. If a deterministic policy μ that maps a state s_t to deterministic values of each action a_t is used, Eq. (58) can be rewritten as Eq. (59). If the deterministic policy is to take a_{t+1} that maximizes the Q function at s_{t+1}, Eq. (59) can be rewritten as Eq. (60). The recursive form in Eq. (60) is called a Bellman optimality equation.

$$G_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \tag{57}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q_\pi(s_{t+1}, a_{t+1})]\right] \tag{58}$$

$$Q_\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q_\mu\big(s_{t+1}, \mu(s_{t+1})\big)\right] \tag{59}$$

$$Q_\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \max\Big(Q_\mu(s_{t+1}, a_{t+1})\Big)\right] \tag{60}$$

The RL based on the Q function can be briefly explained using Eq. (60) as **Table 2** where the policy that takes action to go to the next state that provides the highest Q function is called a greedy policy. The greedy policy can be expressed as Eq. (61) where $Q(s, a)$ is a vector that consist of multiple Q functions with respect to different actions, which is shown in Eq. (62). As the loop continues, more and more Q functions at various states are continuously updated. Eventually, the Q functions at various states converge. Then, the RL is completed and the optimal policy is found. By taking the actions according to the greedy policy, the agent can maximize the rewards. This way of learning the Q function is called Q-learning proposed by Watkins and Dayan (1992). Although the greedy policy is used in **Table 2** for simplicity, the greedy policy is usually not ideal for the RL because it discourages the agent from exploring which is important to find the optimal policy that is optimal globally, not locally. Therefore, a stochastic behavior policy is often used as Eq. (63) which is called an epsilon-greedy policy where p is a random value ranging from zero to one and ε is criteria to determine whether to take the action according to the greedy policy or a random action. The stochastic random action encourages the agent to explore the environment which helps to find the global optimal policy. ε is usually initialized as a high value such as one and is decayed over time so that the agent is encouraged to explore at the beginning but take the greedy actions later.

Table 2 Brief procedure of the RL based on the Q function

```
Environment $E$

Starts from an initial state $s_0$
**while** optimal policy is found **do**
    Take action $a_t$ to go to the next state that provides the highest
    Q function
    Obtain reward $r_{t+1}$ and next state $s_{t+1}$ from $E$
    Calculate the Bellman optimality equation of the Q function
    Update the Q function with $\{s_t, a_t, r_{t+1}, s_{t+1}\}$
**end while**
```

$$\pi(a|s) = \operatorname{argmax}\big(Q(s,a)\big) \tag{61}$$

$$Q(s,a) = \{Q(s,a^{(0)}), Q(s,a^{(1)}), \cdots, Q(s,a^{(n)})\} \tag{62}$$

$$\pi(a|s) = \begin{cases} \operatorname{argmax}\big(Q(s,a)\big) & \text{if } p \geq \varepsilon \\ \text{random action} & \text{if } p < \varepsilon \end{cases} \tag{63}$$

However, the RL has difficulties in learning when the environment is complex because the complex environment has too many states and the states may have non-linearity. Therefore, it is common to use the neural network as a function approximator to calculate the Q function. This neural network-based approximator is parameterized by θ^Q, and θ^Q can be optimized by minimizing the following loss function Eq. (64) by the optimizer, where ρ^β denotes a state visitation distribution by the stochastic behavior policy which is represented as β and y_t is shown in Eq. (65). It should be noted that the loss function Eq. (64) is quite similar to the loss function for the regression task as Eq. (33). In Eq. (33), y_t and $\hat{y}_t$ denote the actual output and the predicted output, respectively. Matching Eq.

(33) and Eq. (64), it can be seen that $Q(s_t, a_t|\theta^Q)$ is the predicted output and $r(s_t, a_t) + \gamma \max\left(Q(s_{t+1}, a_{t+1}|\theta^Q)\right)$ is the actual output. Since $Q(s_t, a_t|\theta^Q)$ is predicted by the neural network, it is a predicted output, but $r(s_t, a_t) + \gamma \max\left(Q(s_{t+1}, a_{t+1}|\theta^Q)\right)$ is semi-actual output because $r(s_t, a_t)$ is an actual value but $Q(s_{t+1}, a_{t+1}|\theta^Q)$ is a predicted value by the neural network. It might seem a bit odd that the true Q function can be learned by minimizing the loss function Eq. (64), but it is possible by conducting the Q-learning from **Table 2** long enough with the epsilon-greedy policy. The Q-learning using the neural network to approximate the Q function is called Deep Q-learning. One important term in the Q-learning is bootstrapping. The bootstrapping is the way of updating the Q function where the semi-actual output consisting of the reward and predicted Q function is used as the actual output in the loss function. The bootstrapping is used to be able to update the Q function at s_t at every timestep. The bootstrapping is utilized in most of the RL algorithms.

$$J(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[\left(y_t - Q(s_t, a_t|\theta^Q)\right)^2\right] \tag{64}$$

$$y_t = r(s_t, a_t) + \gamma \max\left(Q(s_{t+1}, a_{t+1}|\theta^Q)\right) \tag{65}$$

There are two RL algorithm categories as shown in **Fig. 28**. The first one is the value-based RL algorithm, and the second one is the policy-based RL algorithm. The value-based RL algorithms are the RL algorithms whose policy is implicitly determined by the Q function as Eq. (61). In other words, there is no explicit policy but the policy is determined based on the Q function approximated by the neural network. Unlike the value-based RL algorithms, in the policy-based RL algorithms, the policy is explicitly determined in which the policy is directly approximated by the neural network. The advantages and disadvantages of the policy-based RL algorithms compared to the value-based RL algorithms are as follows: The policy-based RL algorithms can output multiple continuous actions at one timestep, while the value-based RL algorithms can only output one discrete action. Therefore, the policy-based RL algorithms are more suitable for applications such as robot control. However, the policy-based RL algorithm is prone to falling into the local optimal policy and its evaluation process for the policy is typically inefficient with high variance. Common applications of the value-based RL algorithm and the policy-based RL algorithm are shown in **오류! 참조 원본을 찾을 수 없습니다.** where **오류! 참조 원본을 찾을 수 없습니다.** (a) and **오류! 참조 원본을 찾을 수 없습니다.** (b) shows one of Atari games called breakout and autonomous car driving, respectively. The action and the action space of each application in **오류! 참조 원본을 찾을 수 없습니다.** (a) and **오류! 참조 원본을 찾을 수 없습니다.** (b) is shown in Eq. (66) and Eq. (67), respectively. The action space in

Eq. (66) has three discrete actions and only one action is available at one time. The action space in Eq. (67) has two continuous actions where the angle of the steering wheel and the car speed can range from 0.0° to n° and from 0.0 km/h to m km/h, respectively. It can also be seen that multiple actions can be taken at one time by using the policy-based RL algorithm.

(a) value-based RL algorithm (b) policy-based RL algorithm

Fig. 29 Common applications of the value-based RL algorithm and the policy-based RL algorithm

$$a = \text{argmax}(\{\text{left}, \text{nothing}, \text{right}\}) \tag{66}$$

$$a = \{\text{angle of steering wheel}, \text{car speed}\} \tag{67}$$

Illustrations of the policies in the value-based RL algorithms and policy-based RL algorithms are shown in **Fig. 30** and Fig. 31, respectively, where HL stands for the hidden layer, softmax denotes a softmax activation layer, P denotes a probability, μ denotes mean, σ denotes standard deviation, and N denotes a normal distribution. The softmax activation layer allows to yield probabilistic outputs in which the sum of the outputs is one. Also, in **Fig. 30** and Fig. 31 (a), the greedy policy is used. In **Fig. 30**, it is shown that the HL outputs the approximated Q functions, and the action that maximizes the Q function is selected. In Fig. 31, two types of the policy-based RL algorithm are shown according to their action spaces. In the policy-based RL algorithm with the discrete action space, a probability of each action is output from the HL and one action with the highest probability is chosen. In the policy-based RL algorithm with the continuous action space, the mean and standard deviation of each action is output from the HL and the normal distribution of each action is formed. The action is taken by sampling from the normal distribution of each action, which results in a vector of three values for three actions.

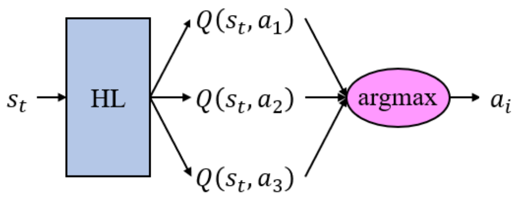
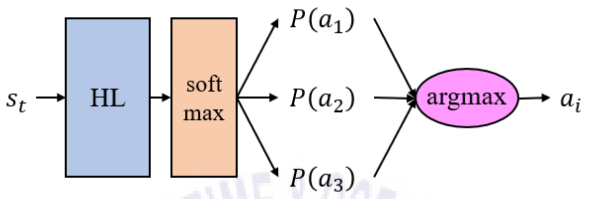
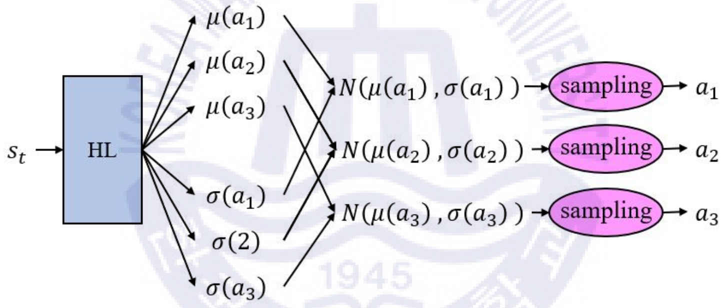


Fig. 30 Illustration of the policy in the value-based RL algorithms



(a) for discrete action space



(b) for continuous action space

Fig. 31 Illustration of the policy in the policy-based RL algorithms

The learning process of the neural network in the policy-based RL algorithm is conducted by "maximizing" an objective function, Eq. (68) where $V_\pi(s)$ is the Bellman equation of a state value function which is expressed as Eq. (69). The Bellman equation of the state value function is quite similar to that of the Q function from Eq. (58). The Q function tells how good a_t is at s_t, whereas the state value function tells how good s_t is. Therefore, it can be said that the Q function is a subset of the state value function. Seeing Eq. (68), the learning process can be intuitively explained: The parameters of the neural network in the policy-based RL algorithm updated in a direction that maximizes $V_\pi(s)$, which is equal to the maximization of the rewards. The detailed parameter update can be expressed as Eq. (70) where θ is the parameter of the neural network for the policy and the gradient of the loss function with respect to θ is expressed as Eq. (71). Eq. (71) is called policy gradient proposed by Sutton et al. (2000).

$$J(\theta) = V_\pi(s) \tag{68}$$

$$V_\pi(s_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}[V_\pi(s_{t+1})]\right] \tag{69}$$

$$\theta = \theta + \eta \nabla_\theta J(\theta) \tag{70}$$

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log(\pi_\theta(a\,|s))\; Q_\pi(s, a)] \tag{71}$$

However, there is one problem in Eq. (71). The Q function $Q_\pi(s, a)$ in that equation cannot be obtained because there is only one neural network for the policy not for the Q function in the basic notion of the policy-based RL algorithm. To resolve this problem, the Q function in Eq. (71) is often obtained by using another neural network. In this case, the policy-based RL algorithm has two neural networks. One is for the policy, and another is for the Q function. The first policy-based RL algorithm that took this approach is called actor-critic. In the actor-critic, the actor represents the neural network for the policy and the critic represents the neural network for the Q function. The update of critic is conducted the same way as the update of the Q-function in the deep Q-learning, and the update of actor is conducted using the policy gradient.

Before addressing the details of the DDPG, there is one more RL algorithm that the DDPG is based on, which is called the DQN proposed by Mnih et al. (2013). The DQN was proposed to improve the learning performance of the deep Q-learning. The deep Q-learning has two main limitations. First, the constant update of the Q function makes the learning unstable, which results in the unstable policy. Second, the Q function is updated with correlated data, which results in failure of the generalization. An example of the two correlated data are $\{s_t, a_t, r_{t+1}, s_{t+1}\}$ and $\{s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}\}$. In order for the learning system to be generalized well, a diversity of data should be ensured in the training data. The DQN overcomes these two limitations by introducing a replay memory (replay buffer), an experience replay, and a target neural network. The replay buffer is a kind of storage for $\{s_t, a_t, r_{t+1}, s_{t+1}\}$, the experience replay is a way of the training with a random mini-batch of n transitions $\{s_i, a_i, r_{i+1}, s_{i+1}\}$ from the replay buffer, and the target neural network is a duplicate of the original neural network from the deep Q-learning in which the original neural network is called the training neural network in the DQN. The target neural network is updated by its parameters being replaced with those of the training neural network every n steps. The replay buffer and experience replay ensure the diversity of the training data which helps the generalization, and the target neural network provides the stable training of the training neural network. A loss function to train the training neural network in the DQN is expressed as Eq. (72) where y_t is shown in (73). It should be noted that $Q(s_t, a_t|\theta^Q)$ and $Q'\left(s_{t+1}, a_{t+1}|\theta^{Q'}\right)$ are obtained by the training neural network and the target neural network, respectively. An illustration of the replay buffer in the DQN is shown in **Fig. 32** and an overall learning process of the DQN is shown in **Table 3**.

$$J(\theta^Q) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, r_t\sim E}\left[\left(y_t - Q(s_t, a_t|\theta^Q)\right)^2\right] \tag{72}$$

$$y_t = r(s_t, a_t) + \gamma \max\left(Q'\left(s_{t+1}, a_{t+1}|\theta^{Q'}\right)\right) \tag{73}$$

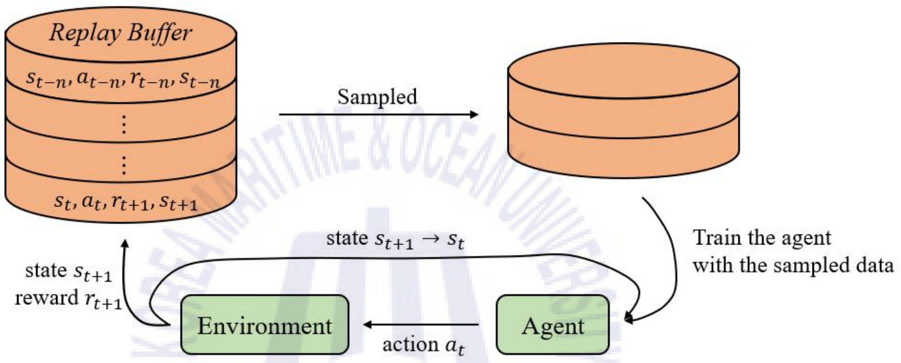

Replay Buffer
$s_{t-n}, a_{t-n}, r_{t-n}, s_{t-n}$
$s_t, a_t, r_{t+1}, s_{t+1}$
Sampled
Train the agent with the sampled data
state $s_{t+1} \rightarrow s_t$
state s_{t+1}
reward r_{t+1}
Environment
Agent
action a_t

Fig. 32 Illustration of the replay buffer in the DQN

Table 3 Overall learning process of the DQN

```
Environment $E$
Replay buffer $R$

Starts from an initial state $s_0$
while optimal policy is found do
    Take action $a_t$
    Obtain reward $r_{t+1}$ and next state $s_{t+1}$ from $E$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random mini-batch of $n$ transitions
    $\{s_i, a_i, r_{i+1}, s_{i+1}\}$ from $R$
    Train the training neural network by minimizing the loss
    function Eq. (72)

    Update the target neural network by replacing its parameters
    with those of the training neural network every $n$ steps
end while
```

The DDPG is the RL algorithm based on the DQN and actor-critic combined. The characteristics of the DDPG are as follows. First, the DDPG has four neural networks in which the first two neural networks are the actor and critic from the actor-critic, and the other two neural networks are a target actor and a target critic. The target actor and target critic are duplicates of the actor and critic. This notion is from the target neural network in the DQN. Second, the DDPG utilizes the replay buffer from the DQN. Third, the target actor and target critic are updated by being replaced with a small percentage of parameters of the actor and critic every step instead of being completely replaced every n steps. This way of target neural network update is called soft update. Fourth, the actor in the DDPG is learned by using a deterministic policy gradient (Silver et al., 2014) instead of the policy gradient. The deterministic policy is a deterministic version of the policy gradient from Eq. (71) and is known to outperform the policy gradient in high-dimensional action space. An equation of the deterministic policy is shown in Eq. (74) where θ^{μ} denotes the parameter of the actor for its deterministic policy μ, and θ^{Q} denotes the parameter of the critic.

$$\nabla_{\theta^{\mu}} J(\theta) = \mathbb{E}\left[\nabla_a Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}\right] \tag{74}$$

In the DDPG, the use of the actor-critic allows to output the multiple continuous actions, the use of target actor and target critic provides the learning stability which is additionally enhanced by the soft update, the use of replay buffer eliminates the correlated training data problem, and the deterministic policy gradient improves the learning performance. An overall learning process of the DDPG is shown in **Table 4**. An architecture of the actor in the DDPG is shown in **Fig. 33**. It should be noted that the actions are deterministically taken.

Table 4 Overall learning process of the DDPG

```
Environment $E$
Replay buffer $R$

Starts from an initial state $s_0$
**while** optimal policy is found **do**
   Take action $a_t$
   Obtain reward $r_{t+1}$ and next state $s_{t+1}$ from $E$
   Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
   Sample a random mini-batch of $n$ transitions
   $\{s_i, a_i, r_{i+1}, s_{i+1}\}$ from $R$
   Update the critic by minimizing the loss function Eq. (72)
```

```
    Update the actor by using the deterministic policy gradient
    Eq. (74)

    Update the target critic and target actor by being replaced
    with a small percentage of parameters of the actor and critic
end while
```

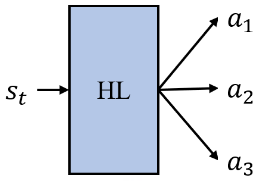


Fig. 33 Architecture of the actor in the DDPG

Chapter 3 Machine Learning-Based Mooring Line Tension Prediction System

3.1. Introduction

Typically, dynamic simulation for a moored floating structure and fatigue analysis on mooring lines takes a long computational time because it requires a huge amount of numerical calculation to represent the actual system's dynamic behavior. To speed up the dynamic simulation and fatigue analysis, several studies have been conducted. Simões, Tiquilloca and Morishita (2002) proposed a neural network architecture to predict the top tension on mooring lines and a hawser in a system where a turret-FPSO is connected with a shuttle tanker by the hawser. Guarize et al. (2007) proposed a hybrid artificial neural network (ANN)-finite element method (FEM) approach where the ANN is trained with short motion and top tension time histories generated by a FEM-based time-domain simulation and the trained ANN calculates the rest of top tension history with an input of a prescribed motion time history. Christiansen et al. (2013) proposed an ANN-based mooring line fatigue analysis approach for the fast fatigue analysis and a procedure for the fatigue analysis based on the proposed approach. Sidarta et al. (2017) proposed an ANN-based mooring line top tension prediction system that receives previous motion time histories only as an input unlike Christiansen et al. (2013) where the ANN receives previous motion time histories as well as previous predicted top tension histories as the input. Despite all the previous developments, there are two main limitations in the previous studies. First, there has not been any standardized method for selecting sea states to obtain a training dataset on which the ANN is trained. Instead, the previous studies manually selected the sea states to obtain the training dataset. However, this manual sea state selection can be problematic because the appropriate sea state selection requires expertise in the ANN domain, and even the careful selection might not be optimal because it is still based on human intuition. Second, the ANN-based mooring line tension prediction system showed relatively low tension prediction performance when a variety of sea states from a wave scatter diagram are considered (Christiansen et al., 2013). In this study, to overcome these two limitations, a K-means-based sea state selection method and a hybrid neural network architecture are proposed. The K-means (Macqueen, 1967) is one of the machine learning algorithms for data clustering and the K-means-based sea state selection method can efficiently select the sea states by clustering relevant sea states with the K-means and finding the nearest sea state to a cluster center in each cluster. The hybrid neural network architecture can provide the robust tension prediction performance across the various sea states in the wave scatter diagram. The hybrid neural network architecture improves the tension prediction performance by using the recent deep learning (DL) algorithms and techniques such as the deep neural network, batch normalization and transfer learning. Finally, it is shown that the proposed K-means-based sea state selection removes the manual selection by standardizing the sea state selection procedure and it results in high sampling and selection efficiency, and the proposed hybrid neural network architecture results in the robust tension prediction system across the entire wave scatter diagram. In the following subchapters, the conventional mooring line tension prediction system is introduced compared to the proposed mooring line tension prediction system, and the proposed K-means-based sea state selection method and hybrid neural network architecture are presented in detail. Finally, simulation conditions and simulation results are presented and discussed.

3.2. Brief Comparison Between Conventional and Proposed Mooring Line Tension Prediction Systems

A brief diagram of the conventional ANN-based mooring line tension prediction system is shown in **Fig. 34**. It should be noted that the sea state selection is conducted manually and a plain ANN is used. The plain ANN is composed of several hidden layers and its architecture is shown in **Fig. 35** where HL stands for hidden layer. For the input, the previous ship motion time histories are used instead of the ship motion at a previous timestep to consider the memory effect in the motion dynamics. The details of the input and output are shown in Eq. (75) and Eq. (76), respectively. In Eq. (75), x, y, and z denote surge, sway, and heave motions, respectively, Δt and N denote a timestep and length of the previous ship motion time histories, respectively. In Eq. (76), $T(t)$ denotes the mooring line top tension at time t and M denotes a number of mooring lines.

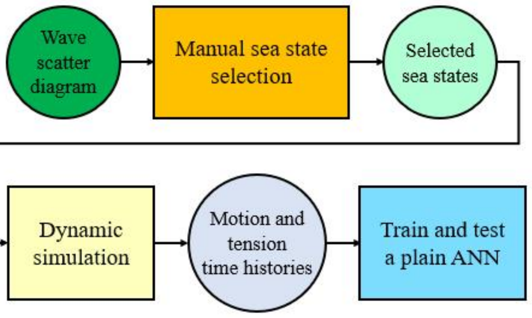

Wave scatter diagram
Manual sea state selection
Selected sea states
Dynamic simulation
Motion and tension time histories
Train and test a plain ANN

Fig. 34 Brief diagram of the conventional ANN-based mooring line tension prediction system

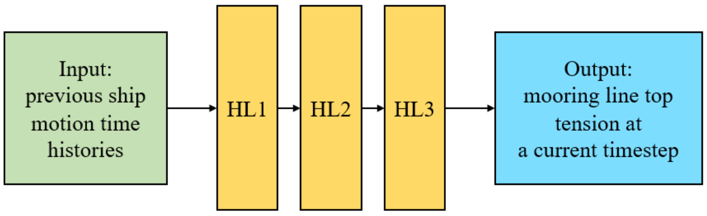


Fig. 35 Plain ANN model used in the conventional ANN-based mooring line tension prediction systems

$$X(t) = \begin{Bmatrix} x(t-N), x(t-(N-\Delta t)), x(t-(N-2\Delta t)), \cdots, x(t), \\ y(t-N), y(t-(N-\Delta t)), y(t-(N-2\Delta t)), \cdots, y(t), \\ z(t-N), z(t-(N-\Delta t)), z(t-(N-2\Delta t)), \cdots, z(t), \end{Bmatrix} \tag{75}$$

$$Y(t) = \{T1(t), T2(t), \cdots, TM(t)\} \tag{76}$$

A brief diagram of the proposed DL-based mooring line tension prediction system is shown in **Fig. 36**. The two main differences between **Fig. 34** and **Fig. 36** are the use of the proposed K-means-based sea state selection method instead of the manual sea state selection and the use of the proposed hybrid neural network architecture instead of the plain ANN. The details of the proposed K-means-based sea state selection method and hybrid neural network architecture are addressed in later subchapters.

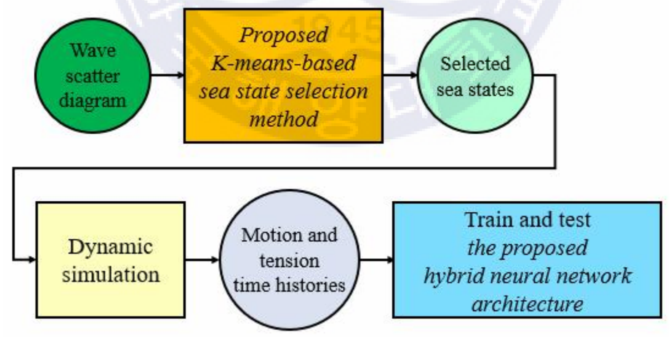


Fig. 36 Brief diagram of the proposed DL-based mooring line tension prediction system

3.3. Proposed K-Means-Based Sea State Selection Method

A detailed diagram of the proposed K-means-based sea state selection method is shown in **Fig. 37**. The details of the components in the K-means-based sea state selection method are presented in this subchapter.

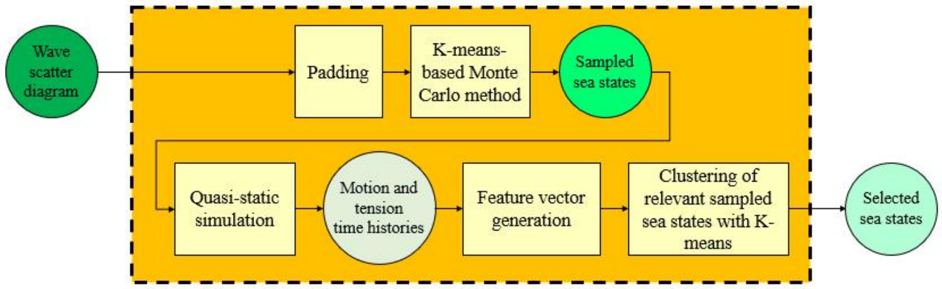
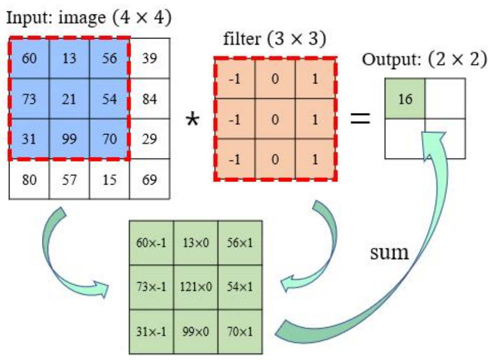

Wave scatter diagram
Padding
K-means-based Monte Carlo method
Sampled sea states
Quasi-static simulation
Motion and tension time histories
Feature vector generation
Clustering of relevant sampled sea states with K-means
Selected sea states

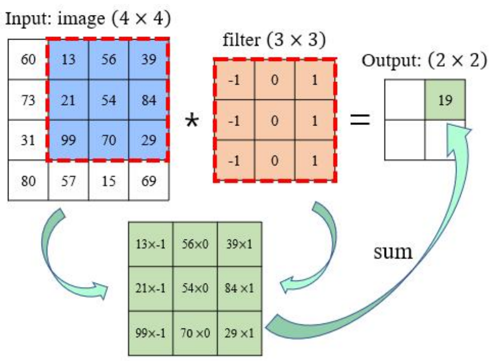
Fig. 37 Detailed diagram of the proposed K-means-based sea state selection method

3.3.1. Padding

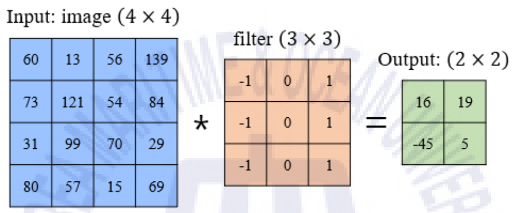
There are many types of neural networks aside from the ANN. One of them is the convolutional neural network (CNN) which is often used in DL-based computer vision systems. A brief diagram of the CNN's convolution operation in one layer is shown in **Fig. 38** where the input refers to a small image, the filter consists of the learnable parameters, and the asterisk denotes the convolution operation. As shown in **Fig. 38**, dimensions of the input and output are different due to characteristics of the convolution operation, and this dimension reduction causes data loss which may be undesirable. To compensate for the data loss during the convolution operation, padding is often used. An illustration of application of the padding is shown in **Fig. 39**. The padding is basically adding zero cells around the input so that even after the convolution operation, the dimension of the output can remain the same as that of the input. In the proposed K-means-based sea state selection method, the padding is applied to the wave scatter diagram. The application of the padding to the wave scatter diagram is shown in **Fig. *40***. The wave scatter diagram is from DNV-GL (2018), Hs and Tz refer to significant wave height and zero-crossing period, respectively, and the blue area and grey area denote a valid sea state area and a padded area, respectively. An output of application of the padding to the wave scatter diagram is the padded valid sea state area in which the valid sea state area is added with an extra surrounding area by the padding. This padded valid sea state area is fed into the K-means-based Monte Carlo method. By using the padded valid sea state area, the data loss can be prevented during the use of the K-means-based Monte Carlo method.



(a) Convolution operation (step1)



(b) Convolution operation (step2)



(c) Convolution operation (finished)

Fig. 38 Brief diagram of the CNN's convolution operation in one layer

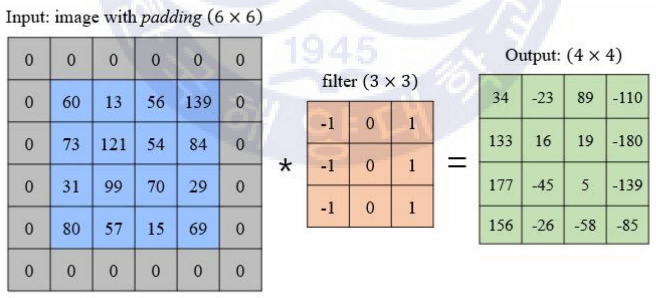
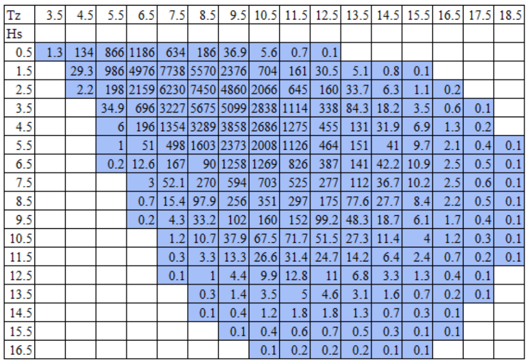
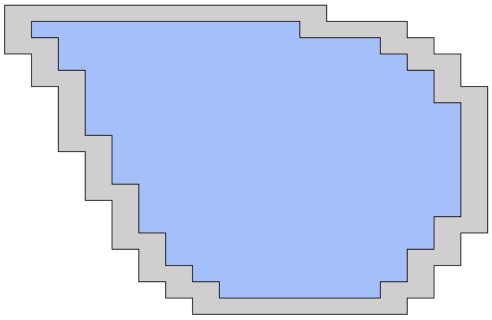


Fig. 39 Illustration of application of the padding



Tz	3.5	4.5	5.5	6.5	7.5	8.5	9.5	10.5	11.5	12.5	13.5	14.5	15.5	16.5	17.5	18.5
Hs																
0.5	1.3	134	866	1186	634	186	36.9	5.6	0.7	0.1						
1.5		29.3	986	4976	7738	5570	2376	704	161	30.5	5.1	0.8	0.1			
2.5		2.2	198	2159	6230	7450	4860	2066	645	160	33.7	6.3	1.1	0.2		
3.5			34.9	696	3227	5675	5099	2838	1114	338	84.3	18.2	3.5	0.6	0.1	
4.5			6	196	1354	3289	3858	2686	1275	455	131	31.9	6.9	1.3	0.2	
5.5			1	51	498	1603	2373	2008	1126	464	151	41	9.7	2.1	0.4	0.1
6.5			0.2	12.6	167	90	1258	1269	826	387	141	42.2	10.9	2.5	0.5	0.1
7.5				3	52.1	270	594	703	525	277	112	36.7	10.2	2.5	0.6	0.1
8.5				0.7	15.4	97.9	256	351	297	175	77.6	27.7	8.4	2.2	0.5	0.1
9.5				0.2	4.3	33.2	102	160	152	99.2	48.3	18.7	6.1	1.7	0.4	0.1
10.5					1.2	10.7	37.9	67.5	71.7	51.5	27.3	11.4	4	1.2	0.3	0.1
11.5					0.3	3.3	13.3	26.6	31.4	24.7	14.2	6.4	2.4	0.7	0.2	0.1
12.5					0.1	1	4.4	9.9	12.8	11	6.8	3.3	1.3	0.4	0.1	
13.5						0.3	1.4	3.5	5	4.6	3.1	1.6	0.7	0.2	0.1	
14.5						0.1	0.4	1.2	1.8	1.8	1.3	0.7	0.3	0.1		
15.5							0.1	0.4	0.6	0.7	0.5	0.3	0.1	0.1		
16.5								0.1	0.2	0.2	0.2	0.1	0.1			

(a) Wave scatter diagram used in this study

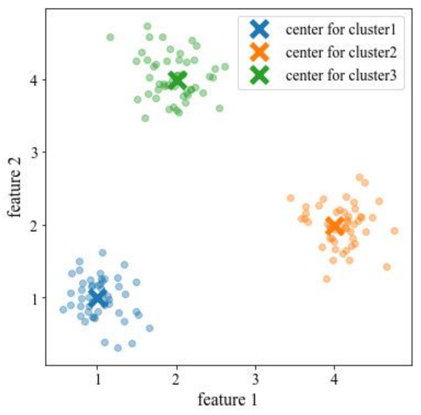


(b) Padded valid sea state area

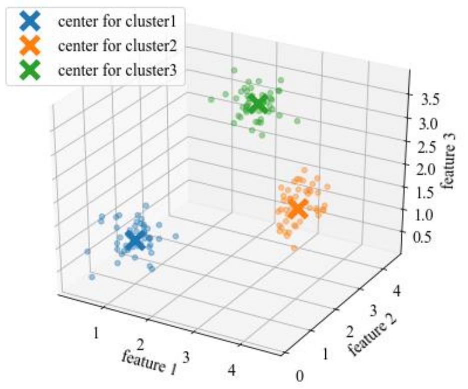
Fig. 40 Application of the padding to the wave scatter diagram

3.3.2. K-Means

The K-means is one of the most popular data clustering algorithms in machine learning and used in various domains such as pattern recognition, marketing research, fraud detection, and so on. Examples of the data clustering by the K-means are shown in **오류! 참조 원본을 찾을 수 없습니다.** where scattered data is clustered to three groups and a cluster center in each cluster is presented as a cross mark. The K-means basically clusters scattered data to k clusters given their features in which k is a parameter and is determined depending on how many clusters are desired. The clustering of the K-means is achieved by minimizing an objective function $\mathcal{L}$ defined in Eq. (77) where k is a number of clusters, i is a cluster number, S_i is a set of data in i-th cluster, and μ_i is a cluster center of i-th cluster. Eq. (78) is used for the calculation of Eq. (77) and the output from Eq. (78) is called a Euclidean distance that is a distance between two n-dimensional vectors. Seeing Eq. (77), it should be noted that the data clustering with the K-means is possible even when a dimension of vectors is very high. When using the K-means, it is also important to note that the input x should be normalized over its features if a distribution of each feature varies much so that each $(x_n - \mu_i)$ in Eq. (78) can have equal significance to the output $\|x - \mu_i\|_2$. This K-means is efficiently utilized for the K-means-based Monte Carlo method and the clustering of relevant sampled sea states with the K-means.



(a) 2D



(b) 3D

Fig. 41 Examples of data clustering by the K-means

$$\mathcal{L} = \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|_2 \tag{77}$$

$$\|x - \mu_i\|_2 = \sqrt{(x_1 - \mu_i)^2 + (x_2 - \mu_i)^2 + \cdots + (x_n - \mu_i)^2} \tag{78}$$

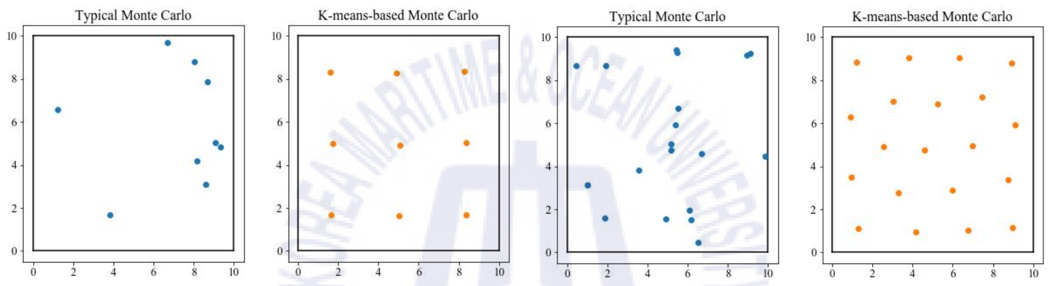
3.3.3. K-Means-Based Monte Carlo Method

The K-means-based Monte Carlo method is proposed in this study to provide efficient sampling than a typical Monte Carlo method. A procedure of the K-means-based Monte Carlo method is presented in **Table 5** where the large number should large enough for generated data to be distributed uniformly within the specified area.

Table 5 Procedure of the K-means-based Monte Carlo method

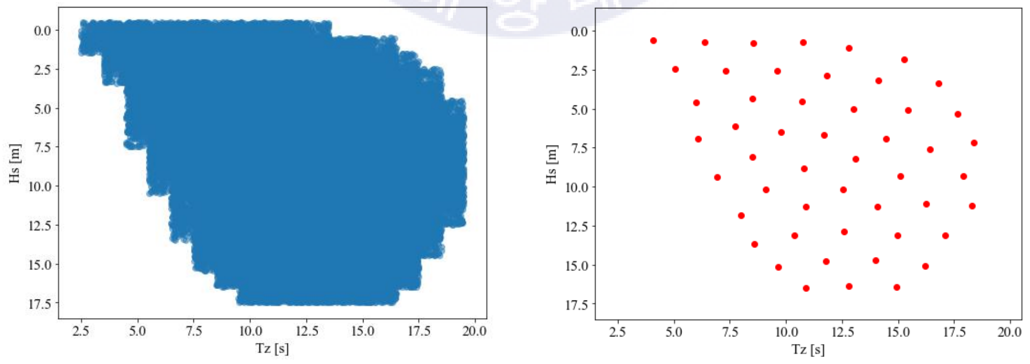
1. Randomly generate a large number of data within a specified area
2. Obtain k clusters with the K-means
3. Obtain k cluster centers → sampled data

To examine the effectiveness of the proposed K-means-based Monte Carlo method, a comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method is presented in **Fig. 42** where data is generated within the black square. It can be observed that the K-means-based Monte Carlo method generates random data much more uniformly. This uniform random data generation is important when it is necessary to sample data evenly to ensure a diversity of the sampled data with respect to its features. The application of the K-means-based Monte Carlo method to the padded valid sea state area is shown in **Fig. 43** and a comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method on the padded valid sea state area is shown in **Fig. 44**, where k is 50. In **Fig. 43**, it is shown that the sea states are sampled very evenly by the K-means-based Monte Carlo method which ensures the diversity of the sea states and the padded valid sea state area allows the K-means-based Monte Carlo method to sample the sea states even at the edge of the valid sea state area. In **Fig. 44**, the effectiveness of the K-means-based Monte Carlo method is clearly shown compared to the typical Monte Carlo method on the padded valid sea state area.



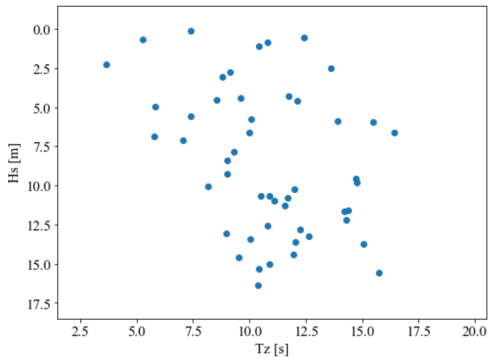
(a) number of samples: 9 (b) number of samples: 20

Fig. 42 Comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method

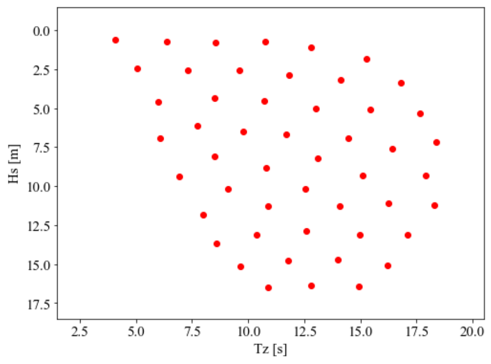


(a) 50,000 randomly generated sea states (b) 50 sampled sea states by the K-means

Fig. 43 Application of the K-means-based Monte Carlo method to the padded valid sea state area



(a) Typical Monte Carlo method



(b) K-means-based Monte Carlo method

Fig. 44 Comparison between the typical Monte Carlo method and the K-means-based Monte Carlo method on the padded valid sea state area

After the sea states are sampled by the K-means-based Monte Carlo method, the quasi-static simulations are conducted on the sampled sea states. Then, ship motion and mooring line tension time histories are obtained for each sampled sea state. The motion and tension time histories are processed into feature vectors through the feature vector generation.

3.3.4. Feature Vector Generation

The feature vector generation processes the ship motion and mooring line tension time histories into the feature vector. The feature vector is used to cluster the relevant sampled sea states with the K-means which is the last step in the proposed K-means-based sea state selection method. A configuration of the feature vector for one sea state is defined as Eq. (79) where the feature vector is represented as $\mathcal{F}$. X', Y', Z', RX', RY', and RZ' refer to normalized six degree-of-freedom ship motion time histories (surge, sway, heave, roll, pitch, yaw), T' and M refer to a normalized mooring line tension time history and a number of mooring lines, respectively. μ and σ refer to a mean and standard deviation, respectively. $\mathrm{FFT}_{\mathrm{CG}x}$ refers to a center of gravity of an FFT graph in an x-axis where FFT stands for Fast Fourier Transform. An example of $\mathrm{FFT}_{\mathrm{CG}x}$ is shown in **Fig. 45** where its argument is $\sin(40t) + 0.5\sin(90t)$, a blue line represents the FFT graph, and a red dotted line represents a value of the $\mathrm{FFT}_{\mathrm{CG}x}$. The size of the feature vector is determined by Eq. (80), therefore, if there are 16 mooring lines, the size of the feature vector is 66. The components in the feature vector were carefully chosen to distinguish different sea states since the different sea states can be distinguished considering overall amplitudes and periods of the ship motions and mooring line tensions. The overall amplitudes are represented by the μ and σ terms in the feature vector, and the overall periods are represented by $\mathrm{FFT}_{\mathrm{CG}x}$ terms in the feature vector.

$$\mathcal{F} = \left\{ \begin{array}{c} \mu(X'), \mu(Y'), \mu(Z'), \mu(RX'), \mu(RY'), \mu(RZ'), \\ \sigma(X'), \sigma(Y'), \sigma(Z'), \sigma(RX'), \sigma(RY'), \sigma(RZ'), \\ \mathrm{FFT}_{\mathrm{CG}x}(X'), \mathrm{FFT}_{\mathrm{CG}x}(Y'), \mathrm{FFT}_{\mathrm{CG}x}(Z'), \mathrm{FFT}_{\mathrm{CG}x}(RX'), \mathrm{FFT}_{\mathrm{CG}x}(RY'), \mathrm{FFT}_{\mathrm{CG}x}(RZ'), \\ \mu(T'1), \mu(T'2), \cdots, \mu(T'M), \\ \sigma(T'1), \sigma(T'2), \cdots, \sigma(T'M), \\ \mathrm{FFT}_{\mathrm{CG}x}(T'1), \mathrm{FFT}_{\mathrm{CG}x}(T'2), \cdots, \mathrm{FFT}_{\mathrm{CG}x}(T'M), \end{array} \right\} \tag{79}$$

$$\mathrm{Size}(\mathcal{F}) = (18 + 3M) \tag{80}$$

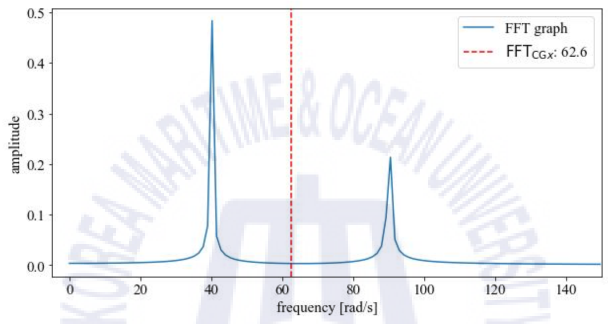


Fig. 45 Example of the $\mathrm{FFT}_{\mathrm{CG}x}$

3.3.5. Clustering of Relevant Sampled Sea States with K-Means

The clustering of relevant sampled sea states with K-means is conducted by clustering the sampled sea states with the K-means based on their feature vectors. Because the feature vector can represent overall characteristics of each sampled sea state, the K-means can cluster the relevant sea states together. An illustration of the clustering of the relevant sampled sea states with the K-means is shown in **Fig. 46** where the number of clusters k is nine and the sea states that are in the same cluster are colored in the same color, otherwise, in different colors.

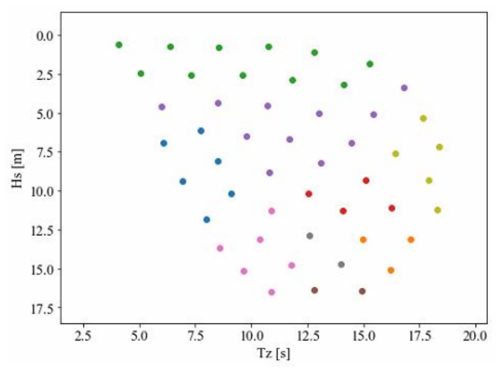
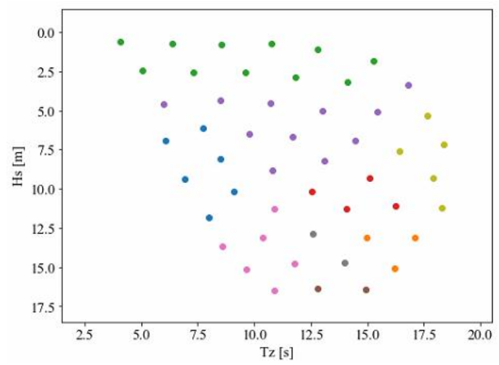


Fig. 46 Illustration of the clustering of the relevant sampled sea states with the K-means

The final goal of the proposed K-means-based sea state selection method is to output the selected sea states which are used to generate the training dataset with the dynamic simulation. The sea state selection is conducted by selecting a representative sea state in each cluster based on the obtained clusters and their cluster centers. A procedure of finding the representative sea state in each cluster is shown in **Table 6**. By selecting the representative sea state in each cluster, a number of the sampled sea states can be reduced to a number of the clusters. This way of sea state selection is very efficient because it significantly reduces a number of sea states to run the dynamic simulation on while it can maintain the diversity of the sea states' features in the wave scatter diagram to ensure the tension prediction performance of a trained neural network model. Ultimately, it results in a large reduction of simulation time. An illustration of the selected sea states by the representative sea states is shown in **Fig. 47**.

Table 6 Procedure of finding the representative sea state in each cluster

1. Obtain a cluster center in one cluster 2. Find the sampled sea state whose feature vector has the lowest Euclidean distance to the cluster center's feature vector in the cluster → a representative sea state 3. Repeat the above steps for all the other clusters


(a) Relevant sampled sea states clustered by K-means

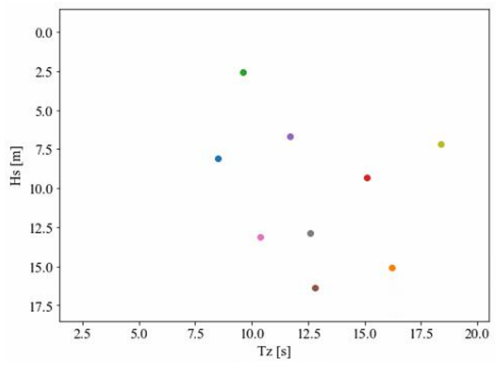

(b) Representative sea states from the clusters

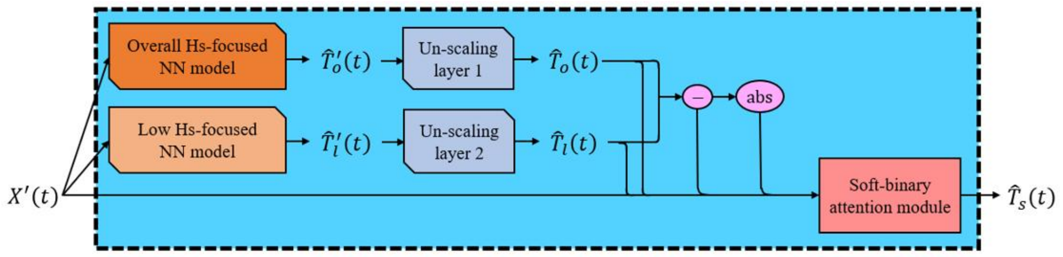
Fig. 47 Illustration of the selected sea states by the representative sea states

The final output from the proposed K-means-based sea state selection method is the selected sea states by selecting the representative sea states in the clusters. The selected sea states are used to generate the training dataset of the motion and tension time histories with the dynamic simulation, and the obtained motion and tension time histories are used to train the proposed hybrid neural network architecture.

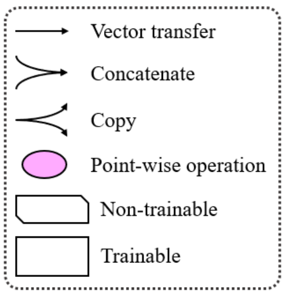
3.4. Proposed Hybrid Neural Network Architecture

The proposed hybrid neural network is proposed to improve the tension prediction performance for various sea states in the wave scatter diagram. It utilizes several modern DL algorithms and techniques such as the deep neural network (DNN), batch normalization (BN), and transfer learning.

3.4.1. Architecture



(a)



(b) for continuous action space

Fig. 48 Architecture of the proposed hybrid neural network

$$X(t) = \left\{\begin{matrix} x(t-10), x\big(t-(9.8)\big), x\big(t-(9.6)\big), \cdots, x(t), \\ y(t-10), y\big(t-(9.8)\big), y\big(t-(9.6)\big), \cdots, y(t), \\ z(t-10), z\big(t-(9.8)\big), z\big(t-(9.6)\big), \cdots, z(t), \\ rx(t-10), rx\big(t-(9.8)\big), rx\big(t-(9.6)\big), \cdots, rx(t), \\ ry(t-10), ry\big(t-(9.8)\big), ry\big(t-(9.6)\big), \cdots, ry(t), \\ rz(t-10), rz\big(t-(9.8)\big), rz\big(t-(9.6)\big), \cdots, rz(t), \end{matrix}\right\} \tag{81}$$

The architecture of the proposed hybrid neural network is presented in **Fig. 48** where NN stands for neural network, $\hat{T}'(t)$ and $\hat{T}(t)$ denote the scaled predicted tension and the predicted tension at time t, respectively. $\hat{T}'(t)$ is output from the NN models because the tension histories in the training dataset are scaled before used for the training to ensure the training stability. The subscripts o, l, and s denote an overall Hs-focused NN model, low Hs-focused NN model, and a soft-binary attention module, respectively. $X'(t)$ is a scaled $X(t)$ with a min-max scaling and $X(t)$ is the input of the previous ship motion time histories and its configuration is shown in Eq. (81) where rx, ry, and rz denote pitch, roll, and yaw motions. A length of $X(t)$ is 10s, and the timestep is 0.2s. The overall Hs-focused NN model is a NN model trained on the training dataset generated from overall sea states in which the overall sea states are the selected sea states by the proposed K-means-based sea state selection method. The low Hs-focused NN model is another NN model trained on the training dataset generated from low Hs-sea states in which the low Hs-sea states are defined as the sea states with the Hs lower than 4m. The un-scaling layer is a layer that un-scales $\hat{T}'(t)$ to $\hat{T}(t)$. To bind the two NN models, the soft-binary attention module is proposed in this study. The proposed soft-binary attention module determines how much attention to pay between the overall Hs-focused NN model and the low Hs-focused NN model given a concatenated vector of $X'(t)$, $\hat{T}_o(t)$, $\hat{T}_l(t)$, $\hat{T}_o(t) - \hat{T}_l(t)$, and abs$\left(\hat{T}_o(t) - \hat{T}_l(t)\right)$. A well-trained soft-binary attention module should output high attention to the low Hs-focused NN model and low attention to the overall Hs-focused NN model when the sea state has the low Hs. When the sea state has the high Hs, the high attention should be paid to the overall Hs-focused NN model and the low attention to the low Hs-focused NN model. When the Hs is somewhere between low and high, the appropriate amounts of attention should be paid to each of the NN models. This fusion of the two NN models by the soft-binary attention module improves the tension prediction performance because the overall Hs-focused NN model can make good tension predictions overall but yields relatively poor prediction performance on the low Hs-sea states and this limitation can be backed up by the low Hs-focused NN model. It is supposed that the overall Hs-focused NN model results in the relatively poor prediction performance because of characteristics of backpropagation. Typically, the weights in the NN model is updated through the backpropagation based on a loss value from a loss function. For the NN-based mooring line tension prediction system, the loss values on the low Hs-sea states are much smaller than on the medium or high Hs-sea states due to the scale of the tension amplitudes, therefore, a majority of the weight update is focused on the medium or high Hs-sea states.

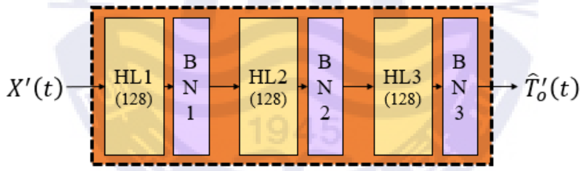


Fig. 49 Architecture of the overall Hs-focused NN model

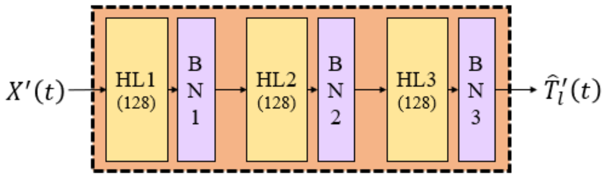


Fig. 50 Architecture of the low Hs-focused NN model

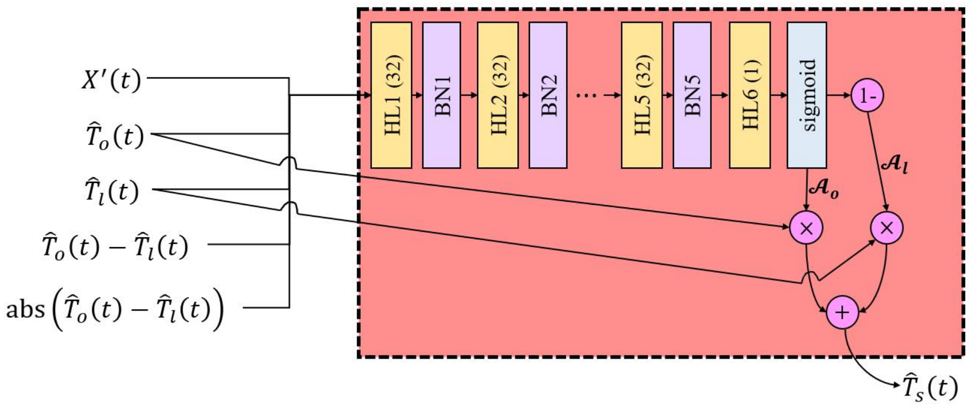


Fig. 51 Architecture of the proposed soft-binary attention module

$$\hat{T}_o(\mathrm{t}) = \hat{T}_o'(t)(\max(T_o) - \min(T_o)) + \min(T_o) \tag{82}$$

$$\hat{T}_l(\mathrm{t}) = \hat{T}_l'(t)(\max(T_l) - \min(T_l)) + \min(T_l) \tag{83}$$

Architectures of the overall Hs-focused NN model, low Hs-focused NN model, and the soft-binary attention module are shown in **Fig. 49**, **Fig. 50**, and **Fig. 51**, respectively. The numbers right next to the HL and BN denote layer index numbers and the numbers in parenthesis next to the HL denote the size of the HL. In **Fig. 51**, the sigmoid denotes a sigmoid activation layer, the pink circles represent a point-wise operation. Importantly, $\mathcal{A}_o$ and $\mathcal{A}_l$ represent amounts of the attention for $\hat{T}_o(t)$ and $\hat{T}_l(t)$, respectively. $\mathcal{A}$ is called an attention value in this study. The attention values range from zero to one due to the characteristics of the sigmoid. The attention values can be visualized to examine how much attention is paid between the overall Hs-focused NN model and the low Hs-focused NN model during inference of the proposed hybrid neural network. For the un-scaling layers 1 and 2, their calculations are conducted by Eq. (82) and Eq. (83), respectively, where T_o denotes the entire tension time histories the overall Hs-focused NN model is trained on and T_l denotes the entire tension time histories the low Hs-focused NN model is trained on. In the proposed hybrid neural network architecture, the overall Hs-focused NN model and low Hs-focused NN model are set to non-trainable, meaning their weights are not updated during the training of the proposed hybrid neural network architecture but used to conduct the inference only. Only the soft-binary attention module is trained during the training of the proposed hybrid neural network architecture. The overall Hs-focused NN model and low Hs-focused NN model are trained before the training of the proposed hybrid neural network architecture. This utilization of the trained NN models is a kind of the transfer learning. The detailed procedure of training the proposed hybrid neural network architecture is shown in the next sub-subchapter.

3.4.2. Training Procedure

The detailed procedure of training the proposed hybrid neural network architecture is presented in **Table 7**. In the steps 2 and 5, the obtained training datasets D_o and D_l are scaled with the min-max scaling. In the step 4, n should be large enough to ensure enough training data for D_l and the low Hs-sea state area refers to the padded valid sea state area with its Hs below 4m. In the step 6, the weights of the low Hs-focused NN model are initialized with the learned weights from the trained overall Hs-focused NN model. This weight initialization is a kind of transfer learning and it helps the low Hs-focused NN model speed up the training process by utilizing the pre-learned knowledge (learned weights and biases). In the steps 7 and 8, by setting the overall Hs-focused NN model and low Hs-focused NN model to non-trainable, these two NN models are not trained during the training of the proposed hybrid neural network architecture in which only the soft-binary attention module is trained.

Table 7 Detailed procedure of training the proposed hybrid neural network architecture

1. Obtain the selected sea states with the proposed K-means-based sea state selection method
2. Obtain the training dataset D_o with the motion and tension time histories from the dynamic simulations on the selected sea states and scale D_o
3. Train the overall Hs-focused NN model with the scaled D_o
4. Sample n low Hs-sea states within the low Hs-sea state area with the proposed K-means-based Monte Carlo method
5. Obtain the training dataset D_l with the motion and tension time histories from the dynamic simulations on the sampled low Hs-sea states and scale D_l
6. Train the low Hs-focused NN model with the scaled D_l
7. Set the overall Hs-focused NN model and low Hs-focused NN model to non-trainable
8. Train the proposed hybrid neural network architecture on D_o and D_l

3.5. Simulation and Result Discussion

3.5.1. Simulation Conditions

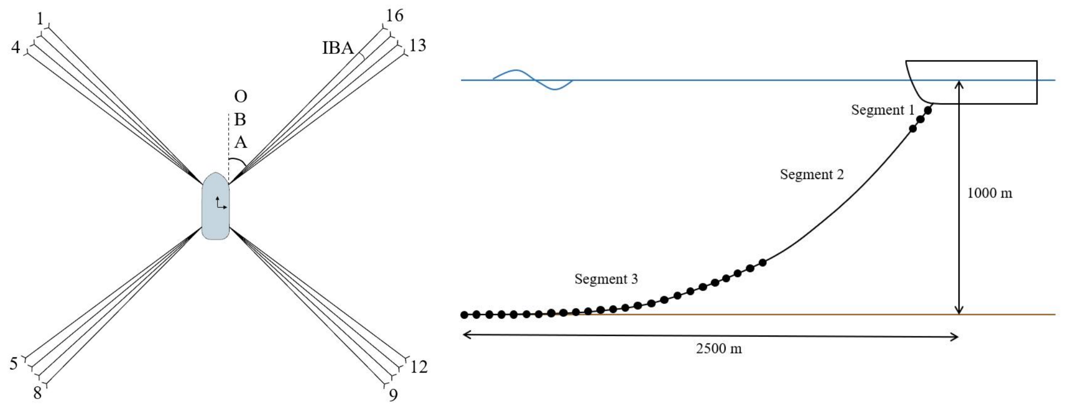
A target ship used in the simulations is a floating production unit (FPU) and its principal dimension is shown in **Table 8**. A mooring configuration of the target ship is shown in **Fig. 52** where the OBA (outer bundle angle) and IBA (inner bundle angle) are set to 50° and 4°, respectively. The small numbers at the ends of the mooring lines indicate the mooring line numbers which start from an upper left side and count counterclockwise. Properties of the mooring lines are shown in **Fig. 52**

Mooring configuration of the target ship

Table 9. An attack angle of environmental direction is 20° to the target ship. Parameter settings for the K-means-based sea state selection method are shown in Table 10. In the neural network training, the Adam is used for an optimizer with a learning rate of 1e-3. The learning rate begins to be decayed to 1e-7 by an exponential learning decay with a decay rate of 0.996 when the training's loss history reaches a plateau. The test sea states are shown in **Fig. 53**. The dynamic simulations on the test sea states are conducted with a different random seed from the training sea states.

Table 8 Principal dimension of the target ship

LBP [m]	Breath [m]	Draft [m]	Volume [m^3]	GM [m]	XCG [m]	YCG [m]	VCG [m]	k_{xx} [m]	k_{yy} [m]	k_{zz} [m]
244	50	18.6	169,614	1.99	117.7	0	19.5	17.4	59.3	60



(a) Top (b) Profile

Fig. 52 Mooring configuration of the target ship

Table 9 Properties of the mooring lines

	Type	Length [m]	Diameter [mm]	Weight in air [kg/m]	Weight in water [kg/m]	MBL [kN]
Segment 1	R4 studless chain	62	130	336	292	15,559
Segment 2	Spiral strand wire	900	120	57	50	9,120
Segment 3	R4 studless chain	1800	130	336	292	15,559

Table 10 Parameter settings for the K-means-based sea state selection method

k in the K-means-based Monte Carlo method	k in the clustering of relevant sampled sea state with K-means
50	10

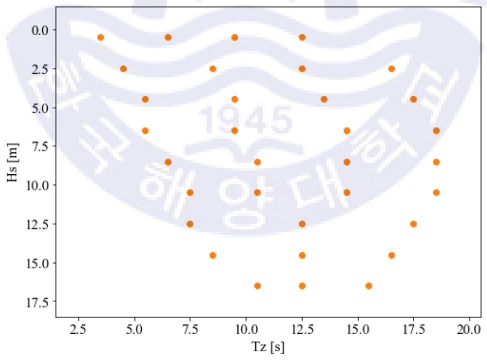
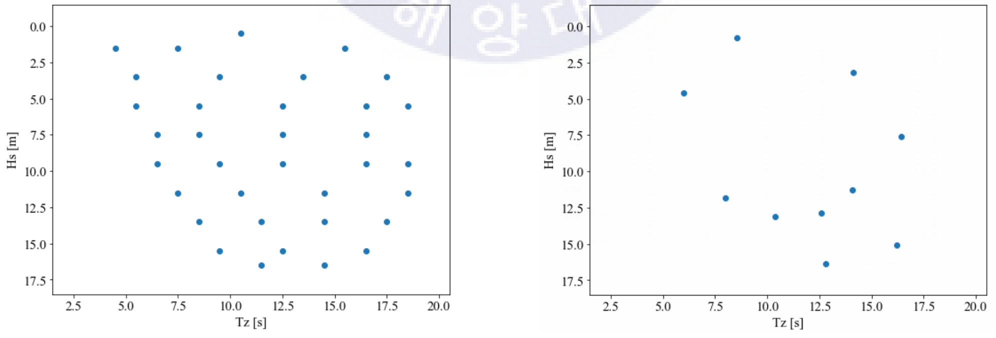


Fig. 53 Test sea states

3.5.2. Overall Hs-focused NN Model

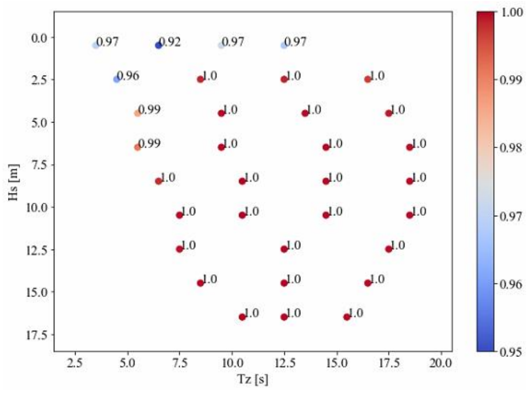
The sea states for the training were manually selected in the previous studies, but they are selected by the proposed K-means-based sea state selection method in this study. Then, the selected sea states are used to obtain the ship motion and tension time histories by the dynamic simulation and the training dataset D_o can be obtained from the motion and tension time histories. Finally, the overall Hs-focused NN model is trained with the scaled D_o. The tension prediction performance of the trained overall Hs-focused NN model and the effectiveness of the proposed sea state selection method compared to the manual sea state selection are investigated by the r-squared and mean average error (MAE) scores of the overall Hs-focused NN models trained on the sea states obtained by manual sea state selection and the proposed sea state selection method, respectively. The selected sea states obtained by the manual sea state selection and the proposed sea state selection method are shown in **Fig. 54**. The r-squared and MAE scores of the trained overall Hs-focused NN models on the test sea states are shown in **Fig. 55**. The r-squared score is a common measurement for regression tasks. Typically, prediction performance for the regression task is considered good if the r-squared score is around 0.9, the performance is considered very good if the score is around 0.95, and the performance is considered excellent if the score is higher than 0.98. Through **Fig. 54** and **Fig. 55**, it can be observed that a number of sea states required to train the NN model can be significantly reduced by using the proposed K-means-based sea state selection method while maintaining the tension prediction performance. The reduction of the number of sea states can result in a large reduction in computational time for the dynamic simulations. Predicted tension time histories of the overall Hs-focused NN model trained on the sea states obtained by the proposed K-means-based sea state selection method are shown in **Fig. *56*** where the tension time histories of the mooring line number 16 are presented and the last 1800s time histories are presented for visual clarity. It can be seen that the tension prediction performance is excellent when the r-squared score is 1.0 and the performance is still very good when the r-squared score is 0.95.

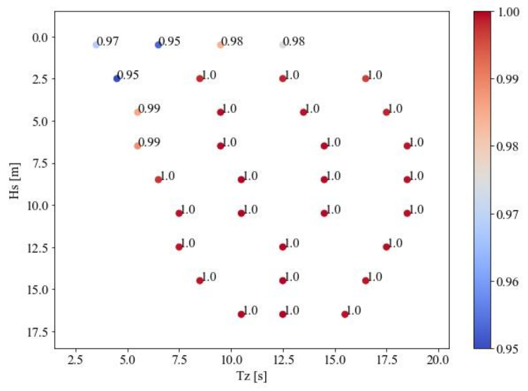


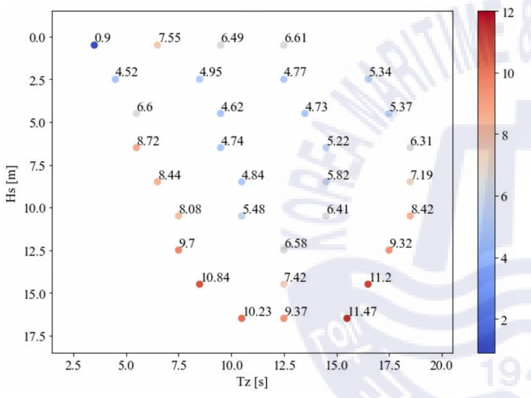
(a) Manual sea state selection
(35 sea states)

(b) Proposed K-means-based sea state selection method
(10 sea states)

Fig. 54 Selected sea states obtained by the manual sea state selection and the proposed K-means-based sea state selection method


(a) Manual sea state selection (r-squared)


(b) Proposed K-means-based sea state selection method (r-squared)


(c) Manual sea state selection (MAE)

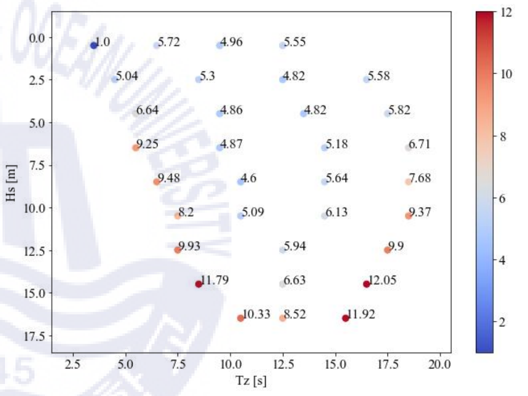
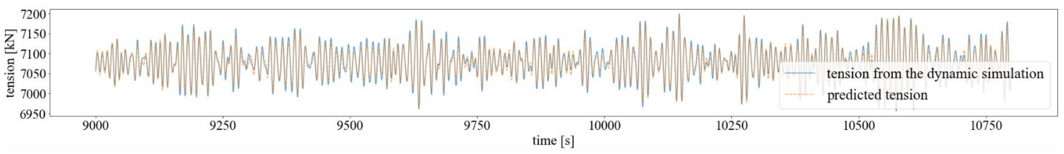
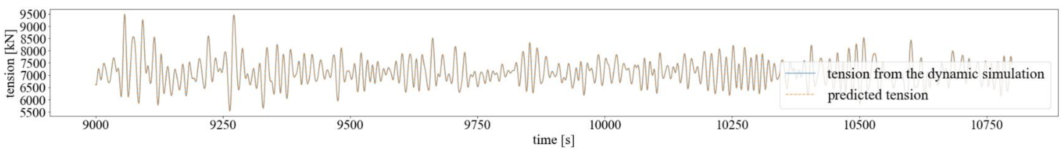

(d) Proposed K-means-based sea state selection (MAE)

Fig. 55 R-squared and MAE scores of the trained overall Hs-focused NN model on the test sea states



(a) Hs: 0.5m, Tz: 6.5s (r-squared score: 0.95)

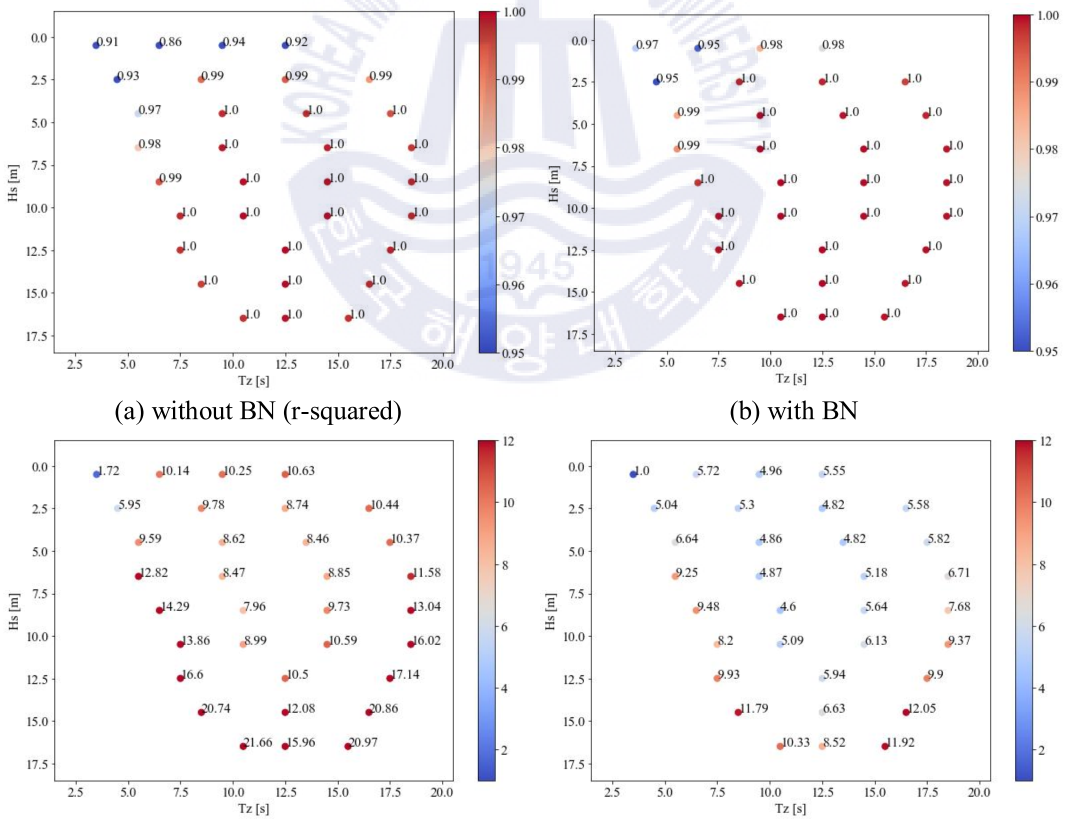


(b) Hs: 8.5m, Tz: 14.5s (r-squared score: 1.0)

Fig. 56 Predicted tension time histories of the overall Hs-focused NN model trained on the sea states obtained by the proposed K-means-based sea state selection method

3.5.3. Effectiveness of Batch Normalization

The BN is utilized in the NN models in this study. One of the advantages of the BN is to improve the generalization, which leads to better performance on test data. The effectiveness of the BN is investigated with respect to the r-squared scores of the trained overall Hs-focused NN models with and without the BN. The overall Hs-focused NN model with the BN consists of three pairs of HL-BN layers, but the overall Hs-focused NN model without the BN consists of three HL layers only. The r-squared and MAE scores of the trained overall Hs-focused NN models with and without the BN on the test sea states are shown in **Fig.** *57*. It is obvious that the use of BN improves the prediction performance, which makes the BN an essential component in the NN model.



(a) without BN (r-squared) (b) with BN

(c) without BN (MAE) (d) with BN (MAE)

Fig. 57 R-squared and MAE scores of the trained overall Hs-focused NN models with and without the BN on the test sea states

3.5.4. Low Hs-focused NN Model

The low Hs-focused NN model is used in the proposed hybrid neural network architecture to back up the overall Hs-focused NN model for the better prediction performance. The low Hs-focused NN model is trained on the scaled D_l in which D_l is obtained from the dynamic simulations on the sampled low Hs-sea states sampled by the proposed K-means-based Monte Carlo method. The sampled low Hs-sea states for D_l are shown in **Fig. 58**. The prediction performance of the trained low Hs-focused NN model is investigated by the r-squared and MAE scores on the test sea states. It can be observed that the low Hs-focused NN model provides the excellent prediction performance on the low Hs-sea states.

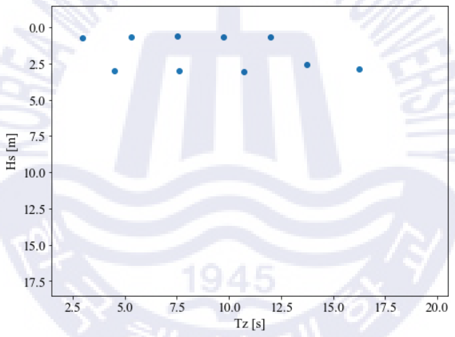
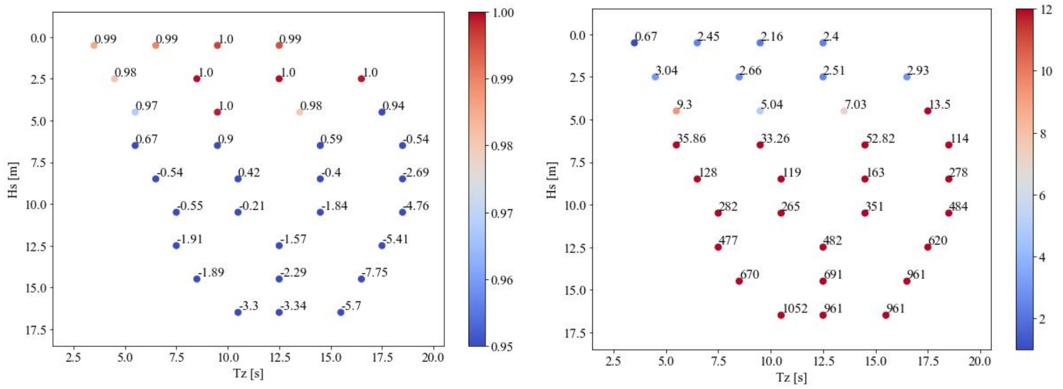


Fig. 58 Sampled low Hs-sea states for D_l

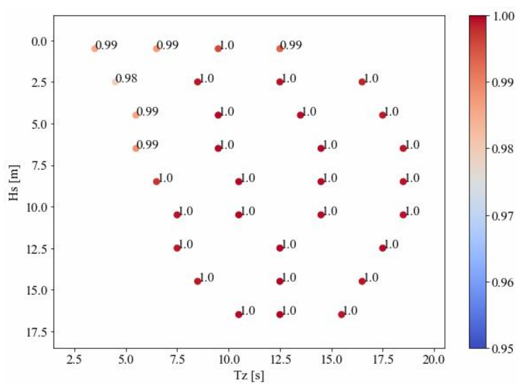


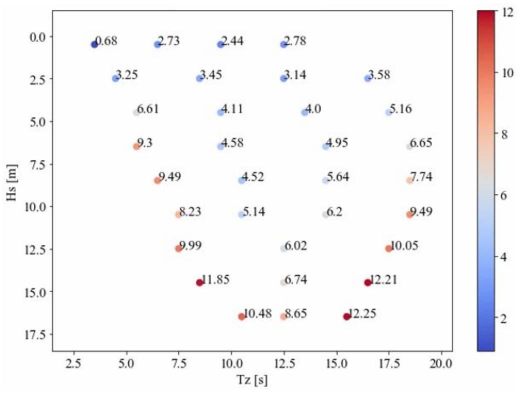
(a) r-squared (b) MAE

Fig. 59 R-squared and MAE scores of the trained low Hs-focused NN model on the test sea states

3.5.5. Proposed Hybrid Neural Network Architecture

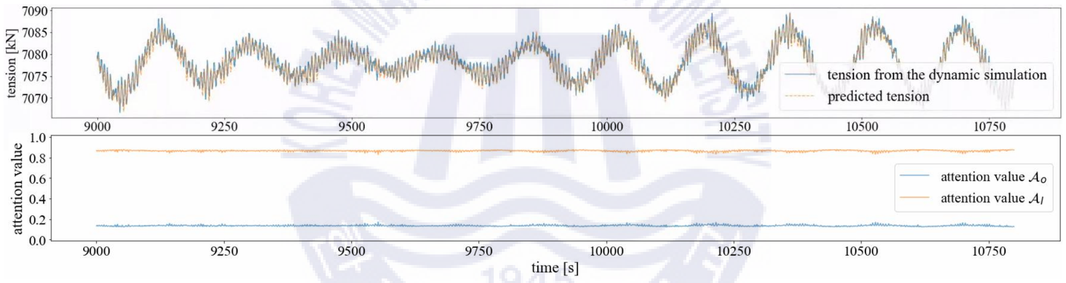
The proposed hybrid neural network architecture mainly consists of the overall Hs-focused NN model, the low Hs-focused NN model, and the soft-binary attention module. The tension prediction is made by appropriately fusing the output tensions from the overall Hs-focused NN model and low Hs-focused NN model with the attention values $\mathcal{A}_o$ and $\mathcal{A}_l$ which are obtained by the soft-binary attention module. The r-squared and MAE scores of the trained hybrid neural network architecture on the test sea states are shown in **Fig. 60** where the tension and attention value time histories of the mooring line number 16 are presented and the last 1800s time histories are presented for visual clarity. The tension time histories along with the attention value time histories of the proposed hybrid neural network architecture are shown in Fig. 61. In **Fig. 60**, it is shown that the hybrid neural network architecture results in the excellent prediction performance across the entire sea states. In Fig. 61, the attention value for the low Hs-focused NN model $\mathcal{A}_l$ is dominant when the sea state has the low Hs. As the sea state becomes more and more severe, the attention value for the overall Hs-focused NN model $\mathcal{A}_o$ becomes more and more dominant. Fig. 61 clearly shows that the soft-binary attention module can provide the appropriate amount of attention to efficiently fuse the overall Hs-focused NN model and the low Hs-focused NN model, which yields the excellent tension prediction performance across the entire wave scatter diagram.



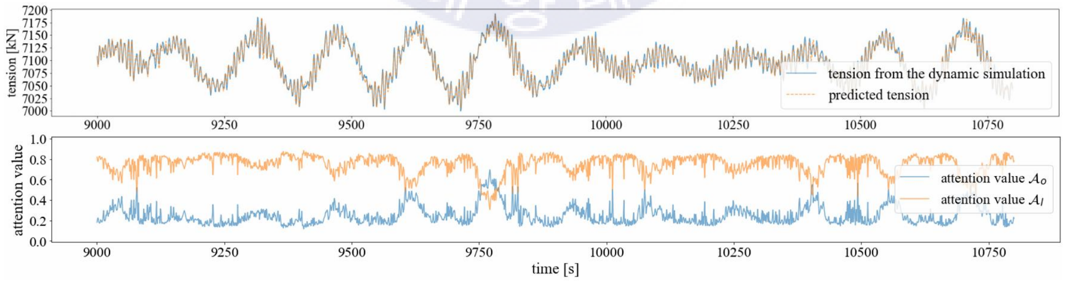


(a) r-squared (b) MAE

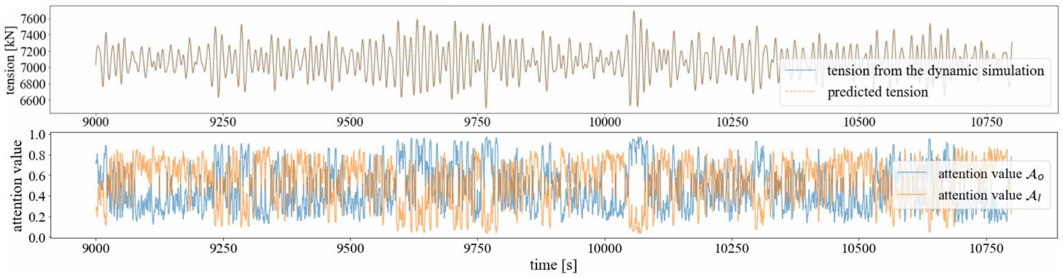
Fig. 60 R-squared and MAE scores of the trained hybrid neural network architecture on the test sea states



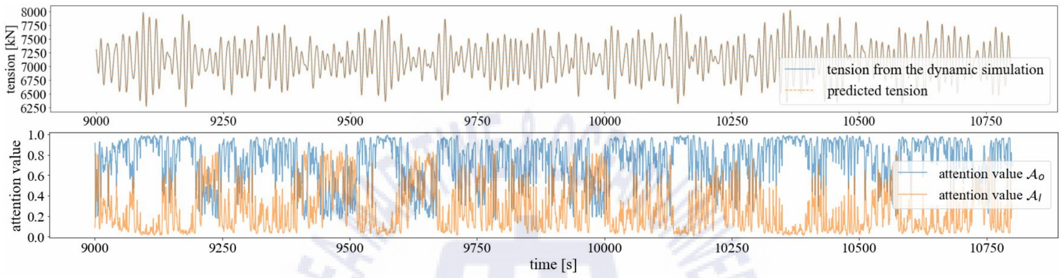
(a) Hs: 0.5m, Tz: 3.5s



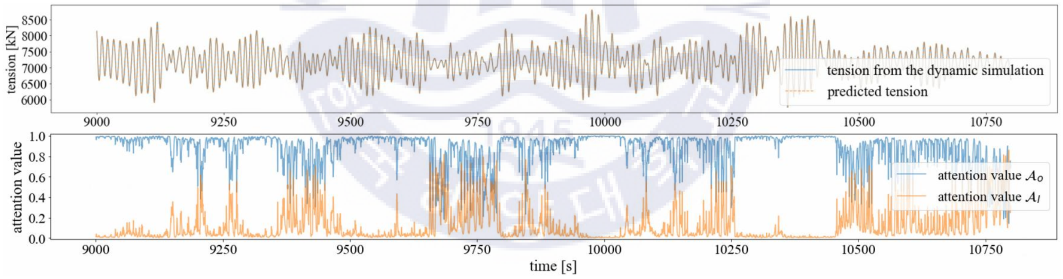
(b) Hs: 2.5m, Tz: 4.5s



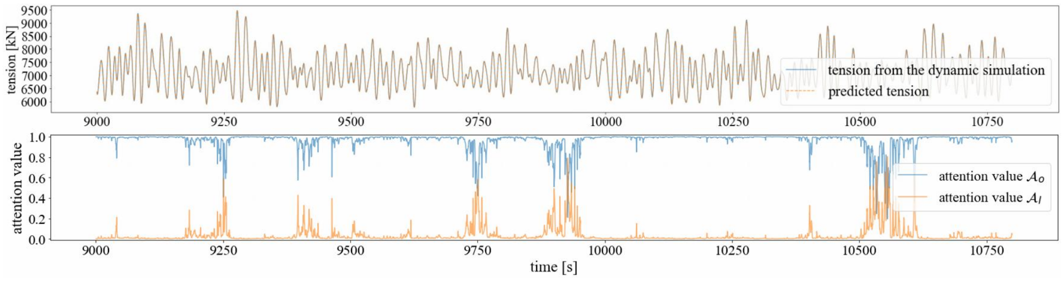
(c) Hs: 2.5m, Tz: 12.5s



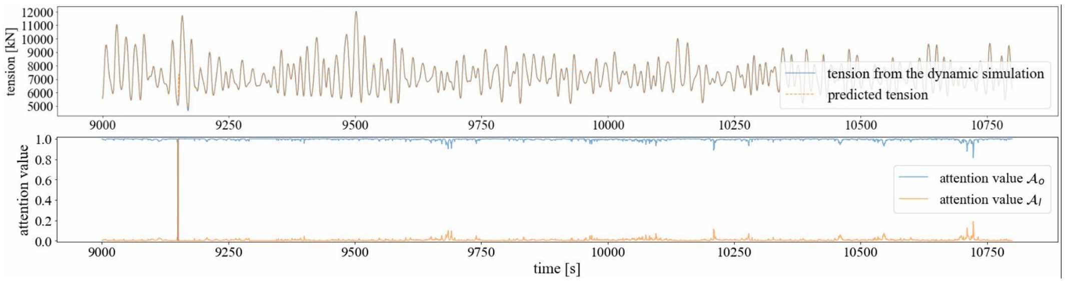
(d) Hs: 4.5m, Tz: 9.5s



(e) Hs: 6.5m, Tz: 9.5s



(f) Hs: 10.5m, Tz: 14.5s



(g) Hs: 16.5m, Tz: 15.5s

Fig. 61 Tension time histories along with the attention value time histories of the trained hybrid neural network architecture

Chapter 4 Motion Predictive Control for DPS Using Predicted Drifted Ship Position Based on Deep Learning and Replay Buffer

4.1. Introduction

Dynamic positioning (DP) of a ship refers to a process of automatically controlling a ship's thrusters to maintain the ship at a reference position and reference heading. Conventionally, a dynamic positioning system (DPS) uses a proportional-integral-derivate (PID) controller to calculate its required thrust to maintain the ship position. The PID controller calculates the required thrust for a current timestep T_t based on an error at a current timestep e_t, where the error is calculated by difference between a reference position and a ship position. Then, the DPS interacts with a given environment with T_t and it results in the ship position at the next timestep. Then again, the error at the next timestep e_{t+1} is calculated based on the reference position and the ship position, which results in T_{t+1} through the PID controller. This way, it forms a feed-back loop and is called a feed-back system. Normally, the feed-back system stabilizes the station-keeping process of the DPS. To improve the station-keeping process of the DPS, the PID feed-back system often takes a feed-forward system. The feed-forward system cancels disturbance such as wind before the disturbance affects a ship's response by providing additional thrust based on a magnitude of the disturbance to cope with the disturbance. Typically, a wind feed-forward system which is the feed-forward system that considers the wind disturbance has been widely adopted because it is straightforward to measure a direction and speed of wind, which makes the wind feed-forward easy to implement. Although the wind feed-forward system can improve the station-keeping performance of the PID feed-back system to some extent, there are still other factors contributing to a ship's drifting motion such as current and wave drift loads. Therefore, considering not only wind load but also current and wave drift loads would be more effective for the feed-forward system design. However, according to (Aalbers *et al.*, 2004), current and wave drift loads cannot be directly measured in the conventional DP control system. In an attempt to consider all the contributing factors to the drifting motion by wind, wave, and current for the feed-forward system, (Song *et al.*, 2016) proposed a feed-forward system that utilizes a predicted drifted ship position in the future because a drifting ship motion contains information about the drifting loads by wind, wave, and current. The feed-forward system proposed by (Song *et al.*, 2016) does the feed-forwarding by replacing the error in the PID controller with a predicted future error in which the predicted future error is calculated as difference between a reference ship position and a predicted drifted ship position in the future. Although (Song *et al.*, 2016) showed some improvement in the station-keeping performance, because the prediction was based on a kind of smoothing method, the prediction accuracy was not sufficient enough to yield the apparent improvement in the station-keeping performance. In this study, a motion predictive control for the PID controller of the DPS is proposed based on the notion of the feed-forward system from (Song *et al.*, 2016). The concept of the control system is the same as (Song *et al.*, 2016), but the major difference lies in the algorithm for the prediction for the drifted ship position. In our proposed motion predictive control, the prediction performance is improved by the following approaches: 1) deep learning (DL) algorithms such as Artificial Neural Network (ANN), Long Short-Term Memory (LSTM) proposed by Hochreiter and Schmidhuber (1997), Gated Recurrent Unit (GRU) proposed by Cho *et al.* (2014), and dropout proposed by Srivastava *et al.* (2014) are considered. 2) an online machine learning system is utilized to provide the proposed motion predictive control with adaptability to a varying sea environment and inexpensive computational load for neural network training. 3) a neural network training method using a replay buffer and a real-time normalization method is proposed to ensure consistently high performance for the prediction, where the replay buffer was proposed by Mnih *et al.* (2013).

In the following chapters, details of the deep learning algorithms and replay buffer, the PID feed-back system with the wind feed-forward system, the proposed motion predictive control, and the numerical modeling of a target ship's behavior are first presented. Then, the effectiveness of the algorithms in the proposed motion predictive control and the simulation results with result discussion are presented.

4.2. PID Feed-Back System and Wind Feed-Forward System

Before introducing the PID feed-back system and the wind feed-forward system, a coordinate system of a ship is first presented in **Fig. 62**. In the figure, $\{n\}$ and $\{b\}$ are the origins of global and local coordinate systems, respectively. ψ is a heading angle of the ship in degrees. The global and local positions of the ship are in meters and denoted as (x, y) and (x', y'), respectively.

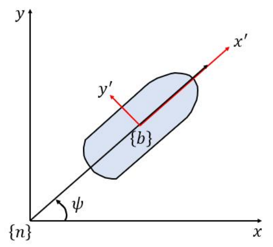
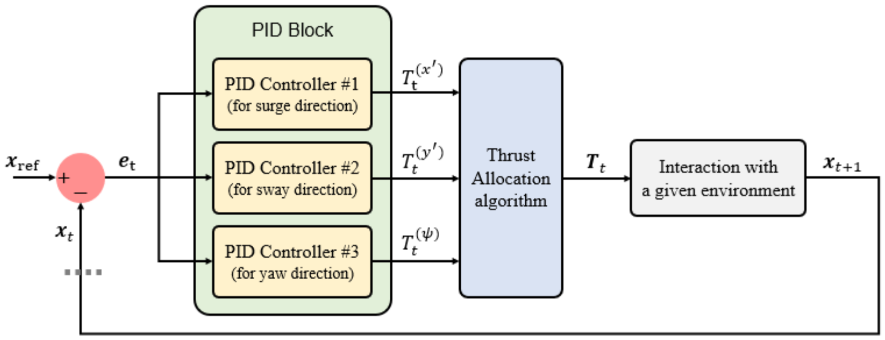


Fig. 62 Coordinate system of a ship

A control law of a single PID controller is shown in Eq. (84) where u_t is the controller's output at time t, e_t is an error which is defined as a reference position subtracted by a current position, k_p, k_i, and k_d are the proportional, integral, and derivative gains of the PID controller, respectively.

$$u_t = k_p e_t + k_i \int_0^t e_t \, dt + k_d \frac{de_t}{dt} \tag{84}$$

In case of the PID controller in the DPS, since there are three directional motions (surge, sway, and yaw) that need to be maintained by the DPS, three PID controllers are used in the DP's PID feed-back system. A structure of the PID feed-back system in the DPS is shown in Fig. 63. In this figure, $\boldsymbol{x}_{\text{ref}}$ is a vector of reference positions of a ship in the surge, sway, and yaw directions. $\boldsymbol{e}_t$ is a vector of the errors in the surge, sway, and yaw directions at time t, $T_t^{(.)}$ is a required thrust in each direction where the direction is denoted in the superscript parenthesis, and each T_t denotes the same as u_t. $\boldsymbol{T}_t$ is a vector of the allocated thrusts for each thruster, and $\boldsymbol{x}_{\mathbf{t+1}}$ is a vector of ship positions in the surge, sway, and yaw directions at time $t + 1$. The details of $\boldsymbol{x}_{\text{ref}}$, $\boldsymbol{e}_t$. $\boldsymbol{T}_t$, and $\boldsymbol{x}_{\mathbf{t+1}}$ are shown in Eqs. (85)-(88). In $\boldsymbol{T}_t$, the number next to T represents a thrust number. For $\boldsymbol{x}_{\mathbf{t+1}}$, it should be noted that the ship positions are the positions filtered by a Kalman filter which is used to remove the ship motion components induced by the 1st-order wave load (Fossen and Perez, 2009).



PID Block
PID Controller #1 (for surge direction)
PID Controller #2 (for sway direction)
PID Controller #3 (for yaw direction)
x_{ref}
e_t
x_t
$T_t^{(x')}$
$T_t^{(y')}$
$T_t^{(\psi)}$
Thrust Allocation algorithm
T_t
Interaction with a given environment
x_{t+1}

Fig. 63 Structure of the PID feed-back system in the DPS

$$\boldsymbol{x}_{\text{ref}} = \{x'_{\text{ref}}, y'_{\text{ref}}, \psi_{\text{ref}}\} \tag{85}$$

$$\boldsymbol{e}_t = \boldsymbol{x}_{\text{ref}} - \boldsymbol{x}_t = \{(x'_{\text{ref}} - x'_t), (y'_{\text{ref}} - y'_t), (\psi_{\text{ref}} - \psi_t)\} = \left\{e_t^{(x')}, e_t^{(y')}, e_t^{(\psi)}\right\} \tag{86}$$

$$\boldsymbol{T}_t = \left\{T1_t^{(x')}, T1_t^{(y')}, T2_t^{(x')}, T2_t^{(y')}, \cdots\right\} \tag{87}$$

$$\boldsymbol{x}_{t+1} = \{x'_{t+1}, y'_{t+1}, \psi_{t+1}\} \tag{88}$$

In Fig. 63, the flow of the control system is as follows: First, $\boldsymbol{e}_t$ is calculated based on $\boldsymbol{x}_{\text{ref}}$ and $\boldsymbol{x}_t$, and it is fed into each of the three PID controllers according to their directions. Then, each PID controller outputs $T_t^{(.)}$ and it is fed into the thrust allocation algorithm to output $\boldsymbol{T}_t$. For the thrust allocation algorithm for the DP thrust, a Lagrange Multiplier method is used (De Wit, 2009).

As mentioned earlier, the conventional PID feed-back system in the DPS often adopts the wind feed-forward system. The PID feed-back system with the wind feed-forward system is expressed as Eq. (89) where the direction is denoted in the parentheses, and $\hat{\tau}_{\text{wind}}(t)$ is an estimated wind load at time t (Fossen and Perez, 2009). It should be noted that the terms in the square brackets denote the PID feed-back system and $-\hat{\tau}_{\text{wind}}(t)$ denotes the wind feed-forward system. A structure of the PID feed-back system with the wind feed-forward system is presented in Fig. 64 where $\hat{\boldsymbol{\tau}}_{\text{wind}}$ denotes a vector of the estimated wind loads in the surge, sway, and yaw directions.

$$T_t^{(.)} = \left[k_p^{(.)} e_t^{(.)} + k_i^{(.)} \int_0^t e_t^{(.)}\, dt + k_d \frac{de_t^{(.)}}{dt}\right] - \hat{\tau}_{\text{wind}}^{(.)}(t) \tag{89}$$

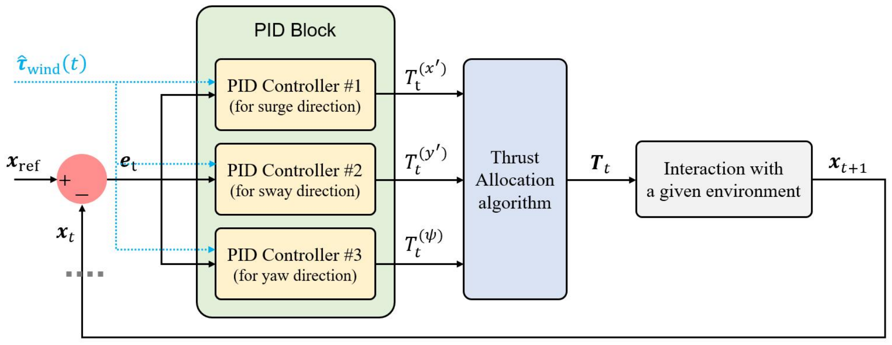

PID Block
$\hat{\tau}_{\text{wind}}(t)$
PID Controller #1 (for surge direction)
PID Controller #2 (for sway direction)
PID Controller #3 (for yaw direction)
x_{ref}
e_t
x_t
$T_t^{(x')}$
$T_t^{(y')}$
$T_t^{(\psi)}$
Thrust Allocation algorithm
T_t
Interaction with a given environment
x_{t+1}

Fig. 64 Structure of the PID feed-back system with the wind feed-forward system

4.3. Proposed Motion Predictive Control

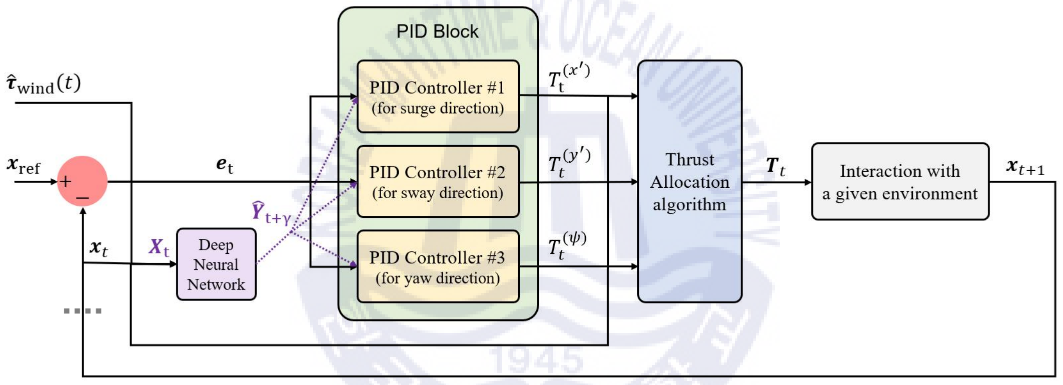

PID Block
$\hat{\tau}_{\text{wind}}(t)$
PID Controller #1 (for surge direction)
PID Controller #2 (for sway direction)
PID Controller #3 (for yaw direction)
x_{ref}
e_t
$\hat{Y}_{t+\gamma}$
x_t
X_t
Deep Neural Network
$T_t^{(x')}$
$T_t^{(y')}$
$T_t^{(\psi)}$
Thrust Allocation algorithm
T_t
Interaction with a given environment
x_{t+1}

Fig. 65 Structure of the PID feed-back system with the proposed motion predictive control

In Fig. 65, a structure of the PID feed-back system with the proposed motion predictive control is presented. The main characteristic is the use of the DNN in which the DNN stands for a deep neural network. The DNN takes the input $\boldsymbol{X}_t$ and outputs $\hat{\boldsymbol{Y}}_{t+\gamma}$, where $\boldsymbol{X}_t$ is based on $\boldsymbol{x}_t$, $T_t^{(.)}$, and $\hat{\boldsymbol{\tau}}_{\text{wind}}(t)$, and $\hat{\boldsymbol{Y}}_{t+\gamma}$ consists of the predicted drifted ship positions in the future. The details of $\boldsymbol{X}_t$ and $\hat{\boldsymbol{Y}}_{t+\gamma}$ are presented in Eqs. (90)-(92). In Eq. (90), $\boldsymbol{s}_t$ denotes a state vector at time t, and ζ and ξ represent length of past data to consider and a sampling rate for ζ, respectively, in which ξ must be a factor of ζ. In Eq. (91), the dot represents the time derivative. In Eq. (92), $\hat{x}'$, $\hat{y}'$, and $\hat{\psi}$ denote the predicted drifted ship positions in the surge, sway, and yaw directions, and γ represents the length of the prediction for the drifted ship motion in the future. It should be noted that the ship positions shown in Eqs. (91)-(92) are filtered by the Kalman filter as the same as in Eq. (88).

$$\boldsymbol{X}_t = \{\boldsymbol{s}_{t-\zeta}, \boldsymbol{s}_{t-\zeta+\xi}, \boldsymbol{s}_{t-\zeta+2\xi}, \cdots, \boldsymbol{s}_t\} \tag{90}$$

$$\boldsymbol{s}_t = \left\{ \begin{array}{c} x'_t, y'_t, \psi_t, \\ \dot{x}'_t, \dot{y}'_t, \dot{\psi}_t, \\ \ddot{x}'_t, \ddot{y}'_t, \ddot{\psi}_t, \\ T_{t-1}^{(x')}, T_{t-1}^{(y')}, T_{t-1}^{(y')}, \\ \hat{\tau}_{\text{wind}}^{(x')}(t), \hat{\tau}_{\text{wind}}^{(y')}(t), \hat{\tau}_{\text{wind}}^{(\psi)}(t) \end{array} \right\} \tag{91}$$

$$\widehat{\boldsymbol{Y}}_{t+\gamma} = \{\hat{x}'_{t+\gamma}, \hat{y}'_{t+\gamma}, \hat{\psi}_{t+\gamma}\} \tag{92}$$

With $\widehat{\boldsymbol{Y}}_{t+\gamma}$, the PID controller in Fig. 65 can be expressed as Eq. (93) where $\hat{e}_{t+\gamma}^{(.)}$ is the predicted future error in one direction and is shown in Eq. (94) in detail. The essence of the proposed motion predictive control lies in the use of $\hat{\boldsymbol{e}}_{t+\gamma}$ instead of e_t. As the PID controller with the proposed motion predictive control utilizes the predicted future error, it can prevent the drifting motion that would occur in the future in advance to some extent.

$$T_t^{(.)} = k_p^{(.)} \hat{e}_{t+\gamma}^{(.)} + k_i^{(.)} \int_0^t e_t^{(.)}\, dt + k_d \frac{de_t^{(.)}}{dt} \tag{93}$$

$$\begin{aligned} \hat{\boldsymbol{e}}_{t+\gamma} = \boldsymbol{x}_{\text{ref}} - \widehat{\boldsymbol{Y}}_{t+\gamma} &= \{(x'_{\text{ref}} - \hat{x}'_{t+\gamma}), (y'_{\text{ref}} - \hat{y}'_{t+\gamma}), (\psi_{\text{ref}} - \hat{\psi}_{t+\gamma})\} \\ &= \left\{\hat{e}_{t+\gamma}^{(x')}, \hat{e}_{t+\gamma}^{(y')}, \hat{e}_{t+\gamma}^{(\psi)}\right\} \end{aligned} \tag{94}$$

Since the station-keeping performance improvement is derived from the use of $\hat{\boldsymbol{e}}_{t+\gamma}$, the prediction accuracy of $\hat{\boldsymbol{e}}_{t+\gamma}$ is important. To achieve the high prediction accuracy of $\hat{\boldsymbol{e}}_{t+\gamma}$, the DNN is adopted as shown in Fig. 65. A structure of the DNN is presented in Fig. 66 where the number in the parenthesis next to the LSTM denotes the size of the cell states and hidden states in the LSTM, and the percentage next to the Dropout denotes a dropout rate. The decision for a type of the neural network layer was made based on simulation assessment which is presented in the next chapter. For the weight initialization in the LSTM, a Xavier initialization method (Glorot and Bengio, 2010) is used. For the weight training, an optimizer called Adam (Kingma and Ba, 2015) is used because it is known to be robust in a variety of areas and the most-used optimizer in the DL literature. For a loss function, mean error squared (MSE) is used.

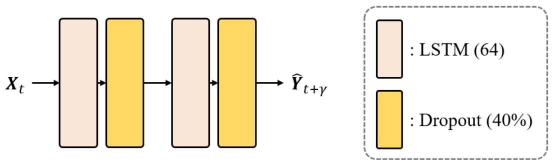
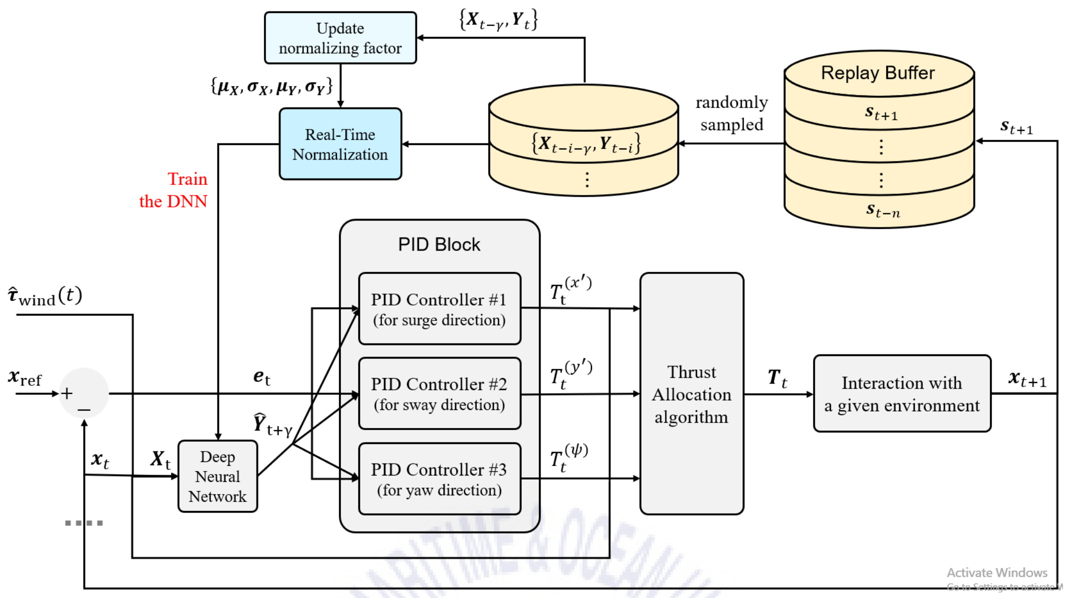


Fig. 66 Structure of the DNN

As briefly introduced in the introduction chapter, the training of the proposed motion predictive control is conducted using the online machine learning system, the replay buffer, and the real-time normalization method. The overall training process is shown in Fig. 67 in which the components related to the training process are in color only for visual clarity. In the figure, n denotes the size of the replay buffer, i denotes an arbitrary positive number that is not greater than n, $\boldsymbol{Y}_{t-i}$ is a vector of ship positions at time $t-i$ as shown in Eq. (95), the 'Real-Time Normalization' refers to the real-time normalization method, and the 'Update normalizing factor' refers to an updating process for the normalizing factors $\{\boldsymbol{\mu_X}, \boldsymbol{\sigma_X}, \boldsymbol{\mu_Y}, \boldsymbol{\sigma_Y}\}$ used in the 'Real-Time Normalization'. The flow of the training process is as follows: 1) obtained $\boldsymbol{s}_{t+1}$ is stored in the replay buffer, 2) a mini-batch of $\{\boldsymbol{X}_{t-i-\gamma}, \boldsymbol{Y}_{t-i}\}$ is obtained by randomly sampling from the replay buffer, 3) the mini-batch is normalized by the real-time normalization method with the updated normalizing factors, 4) the DNN is trained with the normalized mini-batch by one epoch. As to the normalizing factors, $\boldsymbol{\mu_X}$ and $\boldsymbol{\sigma_X}$ refer to feature-wise mean and standard deviation of $\{\boldsymbol{X}_{(t-\gamma)-n}, \boldsymbol{X}_{(t-\gamma)-n+1}, \cdots, \boldsymbol{X}_{(t-\gamma)}\}$, respectively, and $\boldsymbol{\mu_Y}$ and $\boldsymbol{\sigma_Y}$ refer to feature-wise mean and standard deviation of $\{\boldsymbol{Y}_{t-n}, \boldsymbol{Y}_{t-n+1}, \cdots, \boldsymbol{Y}_t\}$, respectively. The update equations of the normalizing factors for $\boldsymbol{\mu_X}$, $\boldsymbol{\sigma_X}$, $\boldsymbol{\mu_Y}$, and $\boldsymbol{\sigma_Y}$ are shown in Eqs. (96)-(99). By using the iterative equations for calculation of the normalizing factors, a significant amount of computational cost can be saved. As to the normalization of the mini-batch, the normalization is conducted as Eqs. (100)-(101) where Eq. (100) and Eq. (101) normalize $\boldsymbol{X}$ and $\boldsymbol{Y}$ in the mini-batch, respectively, and $\boldsymbol{X}'$ and $\boldsymbol{Y}'$ denote the normalized $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.



Update normalizing factor
$\{X_{t-\gamma}, Y_t\}$
$\{\mu_X, \sigma_X, \mu_Y, \sigma_Y\}$
Real-Time Normalization
$\{X_{t-i-\gamma}, Y_{t-i}\}$
randomly sampled
Replay Buffer
s_{t+1}
s_{t-n}
s_{t+1}
Train the DNN
PID Block
PID Controller #1 (for surge direction)
PID Controller #2 (for sway direction)
PID Controller #3 (for yaw direction)
$\hat{\tau}_{\text{wind}}(t)$
x_{ref}
e_t
$\hat{Y}_{t+\gamma}$
x_t
X_t
Deep Neural Network
$T_t^{(x')}$
$T_t^{(y')}$
$T_t^{(\psi)}$
Thrust Allocation algorithm
T_t
Interaction with a given environment
x_{t+1}

Fig. 67 Overall training process of the proposed motion predictive control

$$Y_{t-i} = \{x'_{t-i}, y'_{t-i}, \psi_{t-i}\} \tag{95}$$

$$\mu_X = (1/n)\left[X_{t-\gamma} + (n-1)\mu_X\right] \tag{96}$$

$$\sigma_X = \sqrt{(n-2)/(n-1)\sigma_X^2 + (1/n)\left(X_{t-\gamma} - \mu_X\right)^2} \tag{97}$$

$$\mu_Y = (1/n)[Y_t + (n-1)\mu_Y] \tag{98}$$

$$\sigma_Y = \sqrt{(n-2)/(n-1)\sigma_Y^2 + (1/n)(Y_t - \mu_Y)^2} \tag{99}$$

$$X' = (X - \mu_X)/\sigma_X \tag{100}$$

$$Y' = (Y - \mu_Y)/\sigma_Y \tag{101}$$

In the training process of the proposed motion predictive control, the online machine learning system, the replay buffer, and the real-time normalization method play important roles. First, the online machine learning system refers to a system that trains a neural network with both existing and new data periodically so that the neural network can adapt to the new data's distribution. It is generally adopted in areas where new data continuously comes in such as market analysis. For the DPS, it faces the sea environment where the environmental condition constantly changes over time with respect to a sea state and an environmental direction. Therefore, the DNN in the proposed motion predictive control needs to make the accurate predictions for the drifted ship position in the various sea environmental conditions. To do so, the DNN can be trained and used in the following two ways: 1) trained with data from all the possible environmental conditions and used without further training after deployed, 2) continuously trained with data from recent environmental conditions while being used, which refers to the online machine learning system that the proposed motion predictive control uses. The first way is very inefficient in this case because the DNN has to be trained with data from all the possible environmental conditions, which would require a lot of computational time and put a large load on the DNN for having to generalize all the given data from all the environmental conditions at once. That is why the proposed motion predictive control adopts the online machine learning system. Second, to make the online machine learning system robust, the online machine learning system is adopted with the replay buffer, in which the DNN is continuously trained with both existing data from the replay buffer and new data from the interaction with the sea environment. The training with the existing data from the replay buffer ensures the robust prediction performance, and the training with the new data allows the DNN to make the robust prediction when the new data is fed into the DNN from the new environmental condition as the sea state varies. Third, the real-time normalization method is used to stabilize and improve the neural network training process. It is well known that the training process can be very inefficient if distributions of features in training data are largely different, therefore, the normalization of training data is considered to be one of the critical steps in the neural network training (Lee, Lee and Seo, 2020a).

4.4. Numerical Modeling of Target Ship's Behavior

4.4.1. Target Ship and DPS

A target ship used in this paper is a FPSO and its principal dimension is shown in **Table 11** (Lee, 2008). For the DP system of the target ship, six azimuth thrusters are used. The arrangement and details of the azimuth thrusters are shown in **Fig. 68** and **Table 12**, respectively.

Table 11 Principal dimension of the FPSO

Item	Principal dimension
LBP	285 m

Longitudinal COG	142.26 m
B	63 m
T	13 m
Δ	225,518 ton

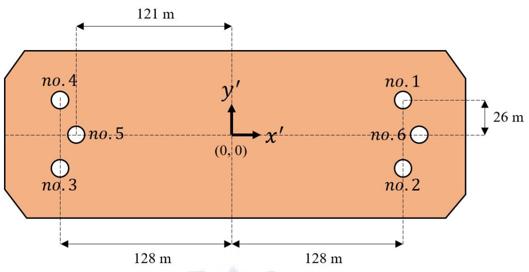


Fig. 68 Arrangement of the azimuth thrusters

Table 12 Locations and thrust limits of the azimuth thrusters

Azimuth thruster	x' [m]	y' [m]	Maximum thrust limit [kN]
no.1	128	26	2000
no.2	128	-26	2000
no.3	-128	-26	2000
no.4	-128	26	2000
no.5	-121	0	2000
no.6	135	0	2000

4.4.2. Equation of Motion of Target Ship

A ship's equation of motion can be expressed as Eq. (102) (Fossen and Perez, 2009).

$$M(p,a) + C(p,v) + K(p) = F(p,v,t) \tag{102}$$

where $M(p,a)$ is the system inertia load, $C(p,v)$ is the system damping load, $K(p)$ is the system stiffness load, $F(p,v,t)$ is the external load, p, v, and a are the position, velocity, and acceleration, respectively, and t is time. For the DPS, the external load $F(p,v,t)$ is equal to Eq. (103) (Fossen and Perez, 2009).

$$F(p,v,t) = \tau_{\text{wind}} + \tau_{\text{current}} + \tau_{\text{wave}} - \tau_{\text{thrusters}} \tag{103}$$

where τ represents external loads and subscripts represent sources of the external forces.

There are three types of environmental loads contributing to the drifting motion of a ship such as wind, current, and wave drift loads. The environmental loads contributing to the drifting motion are presented in **Fig. 69** (Song *et al.*, 2016).

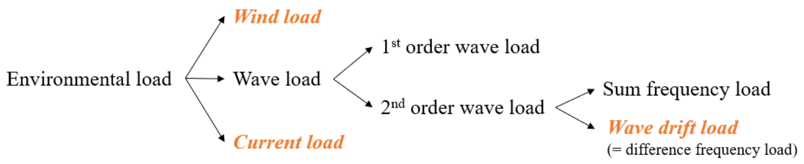


Fig. 69 Environmental loads contributing to the drifting motion

In order for the DPS to compensate for the drifting motion caused by the wind, current, and wave drift loads, azimuth thrusters need to counteract the environmental loads. This relation can be expressed as Eq. (104) in which the DPS maintains a ship's position by trying to satisfy this equation (Fossen and Perez, 2009).

$$(\tau_{\text{wind}} + \tau_{\text{current}} + \tau_{\text{wave drift}}) - \tau_{\text{thrusters}} = 0 \tag{104}$$

The wind and current loads can be expressed as Eqs. (105)-(108) (Fossen and Perez, 2009), and the wave drift load can be expressed as Eq. (109) (Orcina, 2019).

$$\tau_{\text{wind}} = \frac{1}{2}\rho_{\text{air}} V_r^2 \begin{bmatrix} C_{\text{surge}} A_{\text{surge}} \\ C_{\text{sway}} A_{\text{sway}} \\ C_{\text{yaw}} A_{\text{yaw}} \end{bmatrix} \tag{105}$$

$$A_{\text{yaw}} = L_{BP} A_{sway} \tag{106}$$

$$\tau_{\text{current}} = \frac{1}{2}\rho_{\text{water}} V_r^2 \begin{bmatrix} C_{\text{surge}} A_{\text{surge}} \\ C_{\text{sway}} A_{\text{sway}} \\ C_{\text{yaw}} A_{\text{yaw}} \end{bmatrix} \tag{107}$$

$$A_{\text{yaw}} = L_{BP}^2 D \tag{108}$$

where τ_{wind} and τ_{current} are wind and current loads, respectively, $C_{\text{surge}}, C_{\text{sway}}, C_{\text{yaw}}$ are the surge, sway and yaw coefficients with respect to the wind or current direction relative to a ship heading, ρ denotes density, V_r is a relative velocity of wind or current past a ship, $A_{\text{surge}}, A_{sway}$ are the exposed areas above the waterline for τ_{wind} and the submerged areas for τ_{current}. L_{BP} is length between perpendiculars, and D is a draft.

$$\tau_{\text{wave drift}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{Re}\{\text{QTF}_{\text{wave drift}}(\beta_i, \beta_j, T_i, T_j) a_i a_j \exp[i(\omega_i - \omega_j)t - (\phi_i - \phi_j)]\} \tag{109}$$

where n is the number of the regular wave components for describing irregular sea state, Re denotes the real part of a complex number, $\text{QTF}_{\text{wave drift}}$ is the wave drift quadratic transfer function which can be obtained from a diffraction analysis in the frequency domain on the target ship, β is the direction of the regular wave component relative to the ship's heading, T, a, ω, ϕ are the wave period, amplitude, frequency, and phase lag of the regular wave component. Eq. (109) shows that $\text{QTF}_{\text{wave drift}}$ is applied to each pair of the regular wave components to yield each pair's contribution to the wave drift load.

4.5. Effectiveness of Proposed Algorithms

The proposed motion predictive control utilizes the DNN to make the accurate prediction for the drifted ship motion. For the DNN, there are three types of the DL layers considered such as the ANN, LSTM, and GRU. Then, the DNN needs to be properly trained in the online machine leaning system-manner. To make the training process of the DNN robust and result in the accurate predictions, the following two algorithms are used: 1) the replay buffer, 2) the real-time normalization method. In this chapter, comparative simulation results with respect to a type of the DL layer, the use of the replay buffer, and the use of the real-time normalization method are presented. In the simulations, the equation of ship motion is solved by using a marine dynamic simulation software, OrcaFlex, developed by Orcina.

4.5.1. Simulation Conditions

In the simulations, the settings of the parameters in the proposed motion predictive control such ζ, ξ, γ, and n are presented as in Table 13 where Δt is a timestep for the time domain simulation. In the table, ζ and γ were determined considering periods of the low-frequency three-DOF ship motions (surge, sway, yaw) where the periods of the 3-DOF ship motions are determined by magnitude of the PID gains and environmental load. The length of ζ should be long enough to consider the memory effect (Yeung, 1983), but should not be too long since data from far in the past is irrelevant to a current timestep. The length of γ should not be zero since γ of zero eliminates the effects of using the proposed motion predictive control. The length of γ should be just long enough to counteract an upcoming drifting motion. ξ should be greater than 0.1s and lower than 1s since low ξ yields low efficiency in terms of computation memory and high ξ yields data loss between timesteps. The size of the replay buffer n should be big enough to capture statistical characteristics of a current environmental condition but not too big so that data stored in the replay buffer can efficiently adapt to a varying environment.

Table 13 Settings of the parameters in the proposed motion predictive control

$\boldsymbol{X}_t$		$\widehat{\boldsymbol{Y}}_{t+\gamma}$		The replay buffer	
Length of past data ζ	5 s	Prediction length γ	10 s	Size of the replay buffer n	3600 $\times (1/\Delta t)$
Sampling rate ξ	0.5 s				

The environmental condition for the simulations in this chapter is defined in **Table 14** according to the IMCA standard (IMCA, 2000) since it provides the wind-wave relationship table for the DP validation. Finally, the PID gains of the PID feed-back system is set based on a Ziegler-Nichols (ZN) method (Ziegler and Nathaniel, 1942).

Table 14 Environmental condition for the simulations for the effectiveness test

Duration	Wave			Wind	Current	Wind, wave, current directions	Attack angle of environments to a ship
	Type of spectrum	Hs	Tz	Vw	Vc		
0 ~ 10800 s	JONSWAP	6.12 m	9.07 m	17.5 m/s	1 m/s	co-linear	40°

4.5.2. Types of Deep Learning Layers

Three different DNNs based on the ANN, LSTM, and GRU, respectively, are examined in terms of station-keeping performance. Details of the three DNNs are shown in **Table 15**. In the table, the hyperparameters are determined heuristically. A fine hyperparameter search algorithm such as k-fold cross validation is not utilized because of two reasons. First, the prediction performance was consistent over different hyperparameter settings based on common neural network architectures. Second, because the DNN of the proposed motion predictive control is trained on data collected from the dynamic simulations, conducting the fine hyperparameter search is computationally too expensive.

Table 15 Details of the three DNNs

	Hidden layer size	Size of cell states	Size of hidden states	Number of hidden layers	Dropout rate
ANN	64			10	40 %
LSTM		64	64	2	40 %
GRU			64	2	40 %

Ship motion time histories of the three DNNs are shown in **Fig. 70**. It is shown that there is almost no difference in the station-keeping performance between the ANN, LSTM, and GRU. It is assumed that because the future drifted ship position is mostly affected by current and near past data, the LSTM and GRU do not have the advantage over the ANN in terms of the long-term sequential data processing, therefore, they result in the similar performance as the ANN. In this paper, the LSTM is used in the DNN of the proposed motion predictive control because it still shows the slightly better performance and much more commonly used than the GRU.

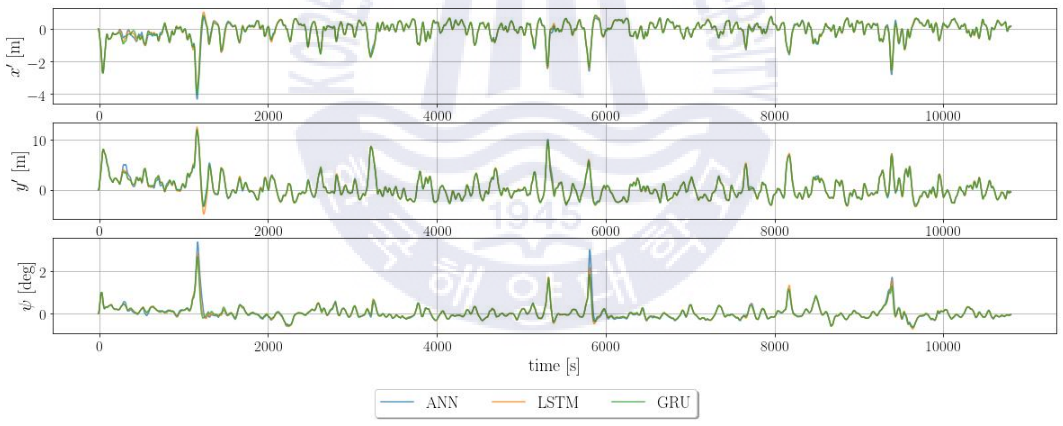
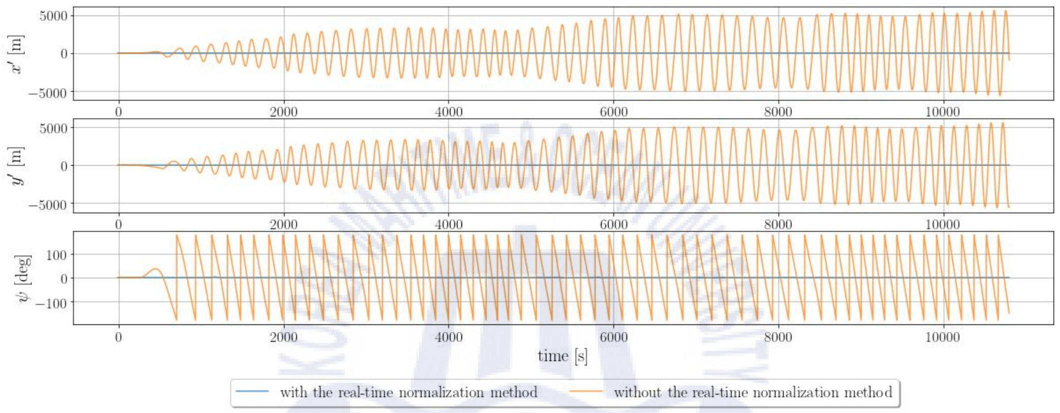


Fig. 70 Ship motion time histories of the three different DNNs based on the ANN, LSTM, and GRU

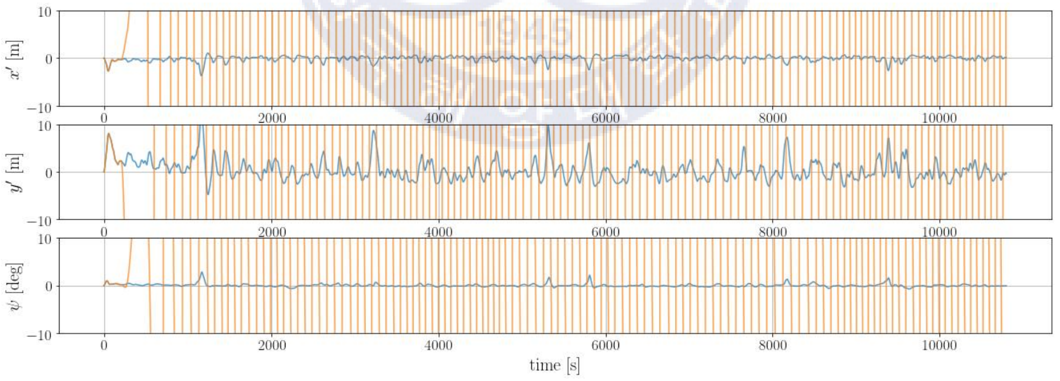
4.5.3. Real-Time Normalization Method

Effectiveness of the use of the real-time normalization method is examined. The following two cases are compared: 1) the PID feed-back system with the proposed motion predictive control as shown in Chapter 4.3, 2) the PID feed-back system with the proposed motion predictive control which does not use the real-time normalization method. The two cases are comparatively presented with the ship motion time histories and loss time histories in **Fig. 71** and **Fig. 72**, respectively. In **Fig. 71**, **Fig.**

71 (a) shows the overall graph and **Fig. 71** (b) shows the zoomed graph. **Fig. 71** shows that the station-keeping performance is much better when the real-time normalization method is used, and its performance difference comes from the training performance difference as shown in **Fig. 72**. As shown in **Fig. 72**, when the real-time normalization method is used, the training loss is much lower, which means that the predicted ship motion is very close to the actual ship motion during the training process. Thus, the low training loss means the high prediction accuracy of the DNN in the proposed motion predictive control, and the high prediction accuracy guarantees the station-keeping performance improvement as mentioned in Chapter 4.3.



(a) overall



(b) zoomed

Fig. 71 Ship motion time histories with respect to the use of the real-time normalization method

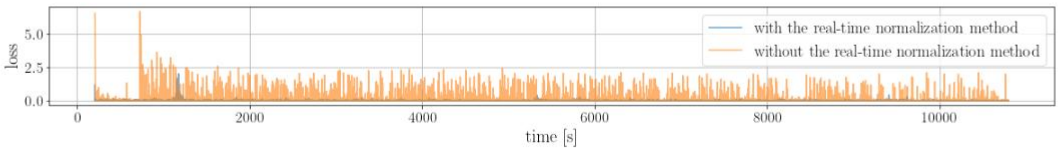


Fig. 72 Loss time histories with respect to the use of the real-time normalization method

4.5.4. Replay Buffer

Effectiveness of the use of the replay buffer is examined. The following two cases are compared: 1) the PID feed-back system with the proposed motion predictive control as shown in Chapter 4.3, 2) the PID feed-back system with the proposed motion predictive control which does not use the replay buffer. In the second case, the DNN is trained with the latest data only. The two cases are comparatively presented with the ship motion time histories and loss time histories in **Fig. 73** and **Fig. 74**, respectively. **Fig. 73** shows that the station-keeping performance is better when the replay buffer is used. Same as in the previous subchapter, this station-keeping performance difference comes from the training performance. **Fig. 74** shows that the training loss is much lower and stable when the replay buffer is used, which ensures the prediction accuracy of the DNN in the proposed motion predictive control.

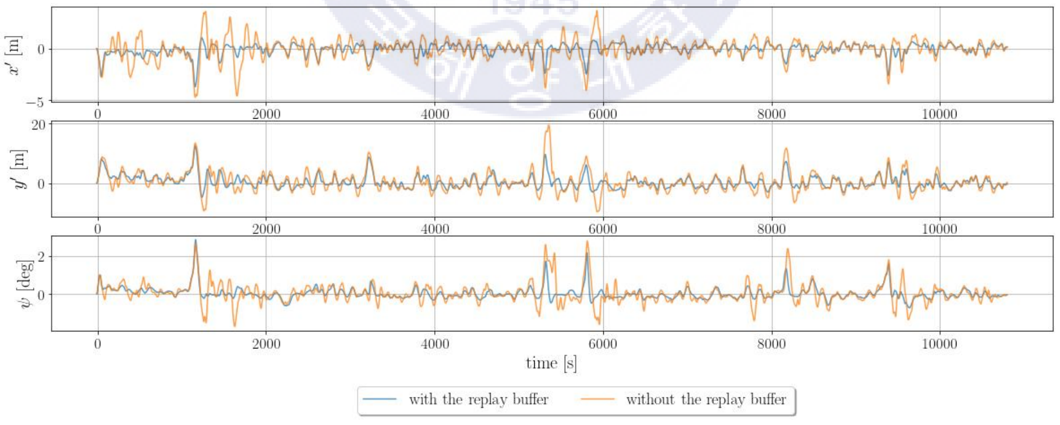


Fig. 73 Ship motion time histories with respect to the use of the replay buffer

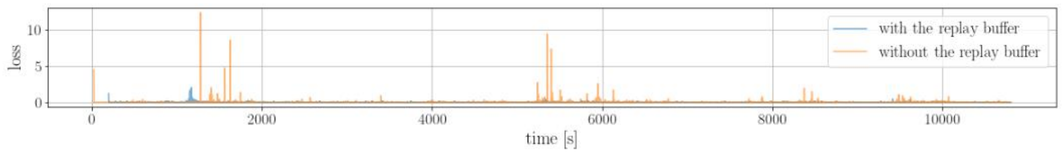


Fig. 74 Loss time histories with respect to the use of the replay buffer

4.6. Simulation and Results Discussion

To validate the proposed motion predictive control, simulation results from the following two systems are compared: 1) the PID feed-back system with the wind feed-forward system, 2) the PID feed-back system with the proposed motion predictive control. In Chapter 4.6.1, the two systems are simulated under one environmental condition to present performance difference and adaptability to a given environment from scratch. The adaptability is defined as an ability to adapt to a new sea environment by learning to make the good predictions for the drifted ship positions, which eventually leads to the better station-keeping. To compare the two systems in terms of the station-keeping performance, ship motion time histories are presented with their statistical result. And, predicted drifted ship position histories of the motion predictive control are presented to examine the prediction performance for the future drifted ship positions. Lastly, thrust time histories are presented. In Chapter 4.6.2, the two systems are simulated under two different sequential environmental conditions to present performance difference and the adaptability of the proposed motion predictive control from one environmental condition to another. Similar to Chapter 4.6.1, ship motion time histories are presented with their statistical result, and predicted drifted ship position histories of the proposed motion predictive control are presented. For simulations in this chapter, the parameter settings of the proposed motion predictive control are the same as in Chapter 4.5.1.

4.6.1. Simulation Under One Environmental Condition

In this chapter, the environmental condition is the same as **Table 14**. Ship motion time histories and their statistics are shown in **Fig. 75** and **Table 16**, respectively. In **Fig. 75**, the 'with the wind feed-forward' refers to the PID feed-back system with the wind feed-forward system, and the 'with the proposed motion predictive control' refers to the PID feed-back system with the proposed motion predictive control. In **Table 16**, the std stands for standard deviation, and $x'y'$ refer to $\sqrt{(x')^2 + (y')^2}$. For the statistics of the motion time histories, the first 2000s motion histories are excluded considering the initial training process of the proposed motion predictive control. **Fig. 75** shows that the better station-keeping performance is achieved by the 'with the proposed feed-forward' than 'with the wind feed-forward'. Especially, the 'with the proposed feed-forward' has the smaller peaks which are equal to the smaller drifting ship motions. The better station-keeping performance of the 'with the proposed feed-forward' can also be seen in **Table 16** from the statistical point of view.

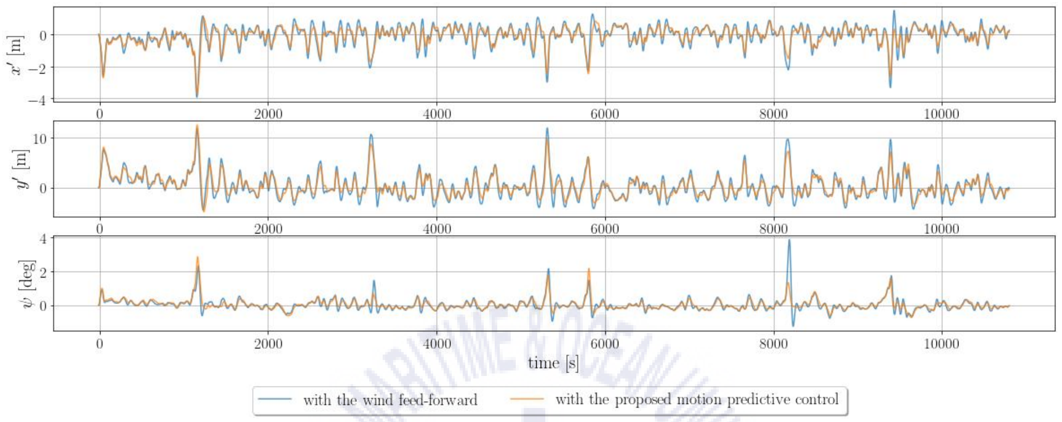


Fig. 75 Ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition)

Table 16 Statistics of the ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition)

	std($x'y'$)	max($x'y'$) [m]	std(ψ)	max($\lvert\psi\rvert$) [deg]
With the wind feed-forward system	1.65	12.30	0.31	3.87
With the proposed motion predictive control	1.32	9.94	0.21	2.17

For the proposed motion predictive control, the prediction accuracy for the drifted ship position in the future is important. To examine the prediction performance, predicted ship motion time histories by the proposed motion predictive control and actual ship motion time histories from the simulation are compared in Fig. 76. For the clear visual comparison, the predicted ship motion time histories are shifted to the left by the prediction length γ to align the timestep with the actual ship motion time histories. Fig. 76 shows that the prediction performance of the proposed motion predictive control is not completely stable yet before around 1700s. It is mainly because the weights of the DNN in the proposed motion predictive control are randomly initialized at 0s and start to be trained from 0s. In other words, because the DNN is not trained enough before around 1700s, the prediction performance is poor in some sections. But from around 1700s to the end of the simulation, the prediction is quite accurate and robust. This accurate prediction is directly reflected in the station-keeping performance as Fig. 75 where most of the peaks of the 'with the proposed motion predictive control' in x', y', and ψ directions are lower than those of the 'with the wind feed-forward' after around 1700s.

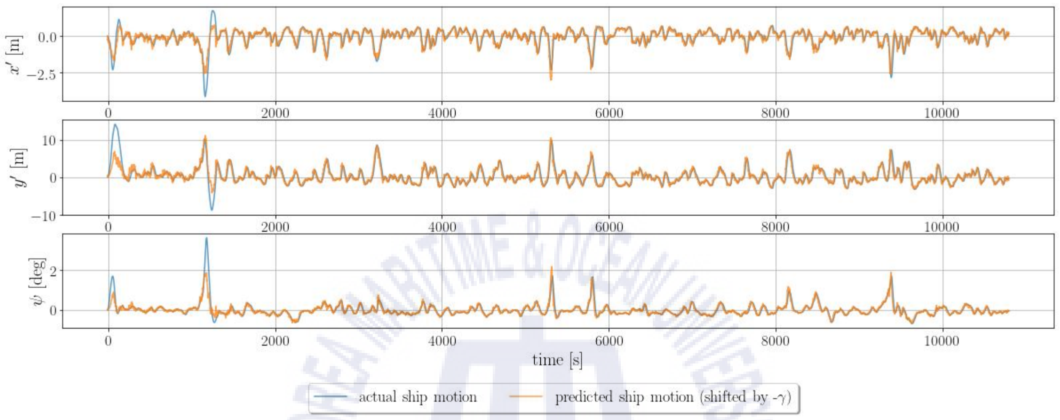


Fig. 76 Comparison between the actual and predicted ship motion time histories (under one environmental condition)

Finally, thrust time histories of the two systems are compared in Fig. 77 where the mean thrust denotes the mean thrust from the six azimuth thrusters and the dotted-line denotes a mean value of the whole thrust time history. It can be observed that the two thrust time histories are quite similar and the mean values of the two thrust histories are almost the same since the two dotted-lines are overlapped. Therefore, it can be said that the PID feed-back system with the proposed motion predictive control is capable of learning to adapt to a given environment from scratch and yields the better the station-keeping performance than the PID feed-back system with the wind feed-forward system.

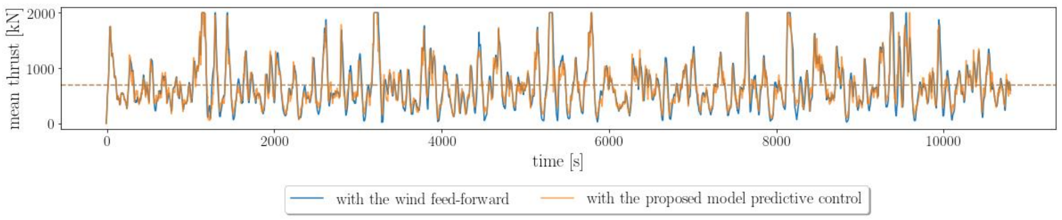


Fig. 77 Thrust time histories with respect to the wind feed-forward and the proposed motion predictive control (under one environmental condition)

4.6.2. Simulation Under Two Different Sequential Environmental Conditions

In Chapter 4.6.1, it is shown that the PID feed-back system with the proposed motion predictive control can adapt to a given environment from scratch, improving its prediction performance as well as the station-keeping performance. The capability of adapting to a given environment is important, but capability of adapting from one environment to another is also equally important because a sea state varies over time.

In this subchapter, two environmental conditions are sequentially placed, where the two environmental conditions are different in both harshness and attack angle. The environmental conditions are shown in Table 17.

Table 17 Two different sequential environmental conditions

Time	Wave spectrum	Hs	Tz	Vw	Vc	Wind, wave, current directions	Attack angle of environments to a ship
0 ~ 10800 s	JONSWAP	4.09 m	7.41 m	12.5 m/s	1 m/s	co-linear	20°
10800 ~ 21600 s	JONSWAP	6.12 m	9.07 m	17.5 m/s	1 m/s	co-linear	40°

First, ship motion time histories are shown in **Fig. 78**, and the actual and predicted ship motion time histories are shown in **Fig. 79** to examine the effects of the prediction accuracy on the station-keeping performance. For the clear visual comparison, the predicted ship motion time histories are shifted to the left by the prediction length γ to align the timestep with the actual ship motion time histories. In **Fig. 78** and **Fig. 79**, the red-dotted line represents the transition point from the first environmental condition to the second.

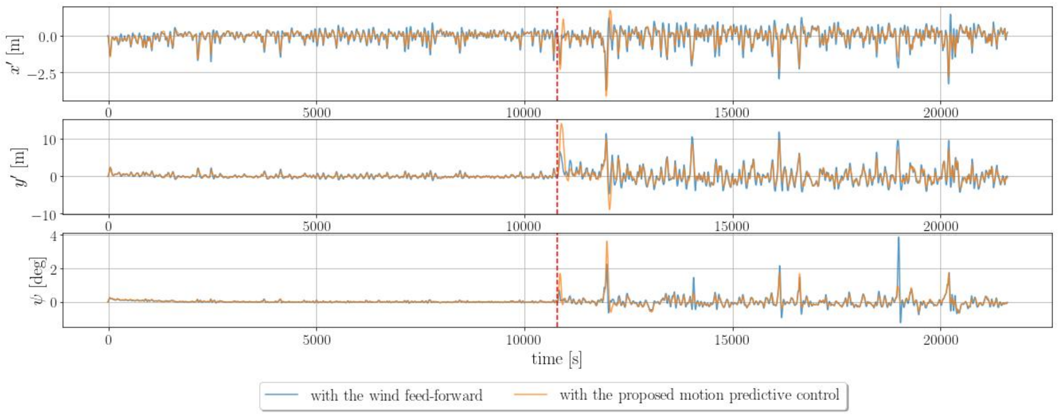


Fig. 78 Ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under two environmental conditions)

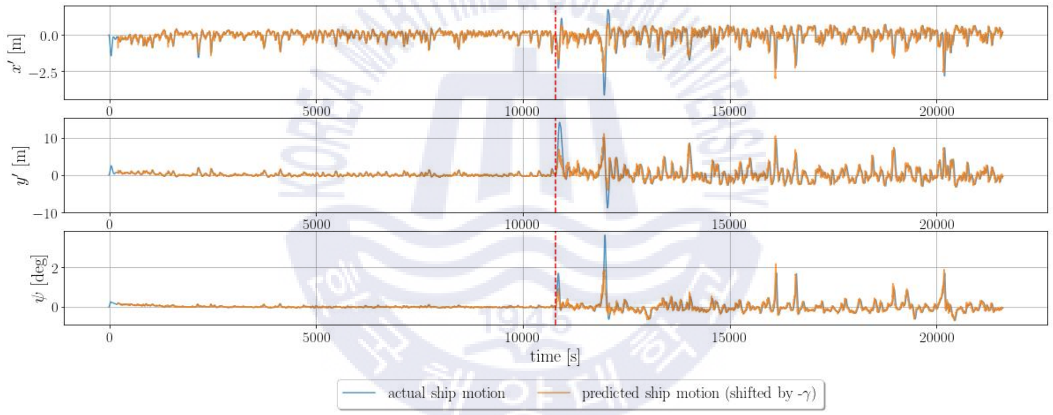


Fig. 79 Comparison between the actual and predicted ship motion time histories (under two environmental conditions)

As shown in Fig. 78, the station-keeping performance of the 'with the proposed motion predictive control' in the first environmental condition is better than the 'with the wind feed-forward' from the beginning unlike Fig. 75. This is because the first environmental condition in Fig. 78 is relatively milder than the environmental condition in Fig. 75 and the training process of the DNN is easier due to the smaller motion amplitudes. It tells that the adaptability of the proposed motion predictive control is more efficient when a sea state is relatively milder. In the early stage of the second environmental condition (around 10800s-12500s), the station-keeping performance of the 'with the proposed motion predictive control' is poorer than 'with the wind feed-forward' as shown in Fig. 78.

This poor station-keeping performance of the ‘with the proposed motion predictive control’ occurs due to the poor prediction accuracy at around 10800s to 12500s as shown in Fig. 79. In detail, since the predicted drifted ship positions are underestimated at the peaks, and it results in the smaller future error, then, the smaller future error causes the smaller thrust and it eventually leads to the further drifting ship motion. This underestimated prediction occurs because the DNN of the proposed motion predictive control trained in the first environmental condition is not accustomed to the second environmental condition at first, therefore, it underestimates the future ship position in the early stage of the second environmental condition. However, as the DNN gets trained with data from the second environmental condition, it starts making very accurate predictions around 1700s after the transition to the second environment. With such accurate predictions by the DNN, the ‘with the proposed motion predictive control’ achieves the better station-keeping performance than ‘with the wind feed-forward’ until the end of the simulation in the second environmental condition.

The statistics of the ship motion time histories in Fig. 78 are shown in Table 18 in which the first 2200s ship motion histories in both environmental conditions are excluded considering the initial training process of the proposed motion predictive control in each environmental condition. Table 18 ensures the better station-keeping performance of the ‘with the proposed motion predictive control’ from the statistical point of view.

Table 18 Statistics of the ship motion time histories with respect to the wind feed-forward and the proposed motion predictive control (under two environmental conditions)

Time	2200 ~ 10800s			
	std($x'y'$)	max($x'y'$ [m]	std(ψ)	max($\lvert\psi\rvert$) [deg]
With the wind feed-forward system	0.38	2.86	0.024	0.17
With the proposed motion predictive control	0.30	2.08	0.019	0.16
Time	13000 ~ 21600s			
	std($x'y'$)	max($x'y'$ [m]	std(ψ)	max($\lvert\psi\rvert$) [deg]
With the wind feed-forward system	1.66	12.29	0.31	3.87

With the proposed motion predictive control	1.31	9.95	0.20	1.72

In this subchapter, the two different environmental conditions are sequentially placed, where the transition from one environmental condition to another occurs instantly. This instant transition of the environmental condition is used to result in the conservative results in terms of the adaptability of the proposed motion predictive control. In other words, if the transition is set to be slow and gradual, it becomes much easier for the proposed motion predictive control to adapt to the new environmental condition and it becomes difficult to assess the maximum adaptability of the proposed motion predictive control. Therefore, by using the instant transition in the simulation, the maximum adaptability of the proposed motion predictive control can be assessed. It should be noted that because the proposed motion predictive control can even adapt to an instantly-varying sea environment, it is expected that the proposed motion predictive control can adapt to a gradually-varying sea environment well without deterioration in the prediction performance.

Chapter 5 Reinforcement Learning-Based Adaptive PID Controller for DPS

5.1. Introduction

Dynamic positioning of a vessel refers to a process of automatically controlling a vessel's thrusters to maintain the vessel at a fixed position and heading. Conventionally, dynamic positioning system (DPS) uses a proportional-integral-derivative (PID) controller (Harbonn, 1971; Nguyen, Sørensen and Tong Quek, 2007; Van 't Veer and Gachet, 2011; Jeon *et al.*, 2017). However, there are two main limitations in a conventional PID controller. First, a PID controller has P, I, and D gains in its structure and they need to be adjusted as parameters. Those gains are usually tuned manually by trial-and-error. However, such manual gain tuning may not result in optimal values for the gains since the gain tuning is based on trial-and-error and human intuition. To automate the PID gain tuning process, many studies have been conducted. The most common and easiest PID tuning method is a Ziegler-Nichols (ZN) method (Ziegler and Nichols, 1942), and this ZN method was adopted in many previous studies for the DPS (Van 't Veer and Gachet, 2011; Koschorrek *et al.*, 2015; Jeon *et al.*, 2017). Aside from the ZN method, other PID gain tuning methods such as Davison method (Davison, 1976), Penttinen method (Penttinen and Koivo, 1980), and Maciejowski method (Maciejowski, 1989) were analyzed for the PID gain tuning for the DPS in (Martin and Katebi, 2005). However, no matter how the PID gains are tuned with the tuning methods introduced above, the tuned gains remain fixed. It is well known that performance of a conventional PID controller is somewhat limited when there is a time-varying load (Teoh and Yee, 1991), which is the second main limitation of a conventional PID controller. In case of the DPS, since it needs to maintain a vessel's position at the open sea, it faces environmental loads such as wind, wave, and current loads which vary over time. Thus, a PID controller with the fixed gains cannot yield efficient station-keeping performance for the DPS. To cope with the time-varying loads, an adaptive PID controller was introduced (Teoh and Yee, 1991). The main advantage of an adaptive PID controller is that it adaptively tunes the gains to cope with the varying loads. For the DPS, many adaptive PID controllers were developed based on fuzzy logic (Yamamoto and Morooka, 2005; Jeon *et al.*, 2017; Xu *et al.*, 2019). In the previous studies of the fuzzy-based adaptive PID controller for the DPS, the controller adaptively tuned the gains considering a position and velocity of a ship at every timestep, and it achieved better station-keeping performance. But since this fuzzy-based adaptive PID controller used fuzzy logic, it had intrinsic limitations of manual definition of fuzzy rules and fuzzy variables of the system, both of which require human expertise in the application domain and can result in different performance by different individuals. Also, the fuzzy-based adaptive PID controller's fuzzy rule was designed to increase the P and D gains when a ship starts being drifted or is far away from a target position. This gain-tuning strategy certainly increases the station-keeping performance, but it comes with the cost of aggressive thrust control (Xu *et al.*, 2019), which is undesirable in terms of control efficiency (Jon and Breivik, 2009).

In this paper, a concept of an adaptive PID controller is used and its gains are tuned by the adaptive fine-tuning system proposed in this study instead of the fuzzy logic. The proposed adaptive fine-tuning system is based on the following two notions: 1) it should not be dependent of human intuition-based adaptive gain-tuning strategy, 2) it should be able to achieve the better station-keeping performance without further DP thrust consumption or a higher rate of the thrust change, which represent the control efficiency. To achieve the first notion, one of the deep reinforcement learning (DRL) algorithms, deep deterministic policy gradient (DDPG) (Lillicrap *et al.*, 2016), is utilized since it does not require any prior knowledge about the dynamics of a ship or DPS to learn an efficient adaptive gain-tuning strategy by interacting with a given environment. To achieve the second notion, the proposed adaptive fine-tuning system is restricted for its available gain range to $[0, k_{\text{base}}]$ where k_{base} represents a base gain. The base gain is a pre-tuned gain by PID gain-tuning methods such as manual tuning method, ZN method, or any PID gain-tuning method. Because the available gain range is restricted, the proposed adaptive fine-tuning system is restricted in overusing the DP thrust to increase the station-keeping performance. With the two notions, the adaptive fine-tuning system is forced to learn how to efficiently tune the adaptive gains to increase the station-keeping performance while its choice for the gain is restricted not to overuse the thrust. The term "fine-tuning" in the proposed system comes from the fine-tuning for a convolutional neural network in deep learning (Guo *et al.*, 2019).

As to the application of the DRL to an adaptive PID controller, there have been many studies conducted for robots (Gloye *et al.*, 2005; Carlucho, De Paula and Acosta, 2019, 2020). In Carlucho et al., (2020, 2019) and Gloye et al. (2005), the robots with the DRL successfully learned to achieve a fast reference tracking with the fast convergence by adaptively tuning the gains. The main differences between those previous works and the work in this paper are a target of the application and the environment. In this work, unlike Carlucho et al., (2020, 2019) and Gloye et al. (2005), a 285m FPSO (Floating Production Storage and Offloading) ship is used as a target of the application which has a much slower response to a control signal than a small robot, and the environmental load is far greater which causes a high magnitude of drifting motions (Van ’t Veer and Gachet, 2011). For the proposed adaptive fine-tuning system, the main goal is the same as the fast convergence to the tracking reference, but instead of converging to a varying reference quickly, it needs to converge to a constant reference position quickly after the ship is drifted by the environmental load.

In the following subchapters, a target ship and the DPS, and the proposed adaptive fine-tuning system are presented. Then, simulation results and discussion are followed.

5.2. Target Ship and DPS

The target ship and DPS introduced here are the ship and DPS used for the simulations in the result subchapter. Properties of the target ship and DPS are presented in detail. Principal dimension of the target ship and its coordinate system are shown in **Table 19** and **Fig. 80**, respectively. In **Fig. 80**, $\{n\}$ and $\{b\}$ are the origins of global and local coordinate systems, respectively. ψ is a heading angle of the ship in degrees. The global and local positions of the ship are in meters and denoted as (x, y) and (x', y'), respectively. The target ship is equipped with six azimuth thrusters. Their arrangement and details are shown in **Fig. 81** and **Table 20**, respectively.

Table 19 Principal dimension of the FPSO ship

Item	Principal dimension
Length Between Perpendiculars (LBP)	285 m
Longitudinal Center of Gravity	142.26 m
Breadth	63 m
Draught	13 m
Weight	225,518 ton

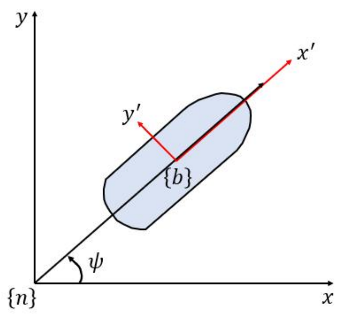


Fig. 80 Global and local coordinate systems

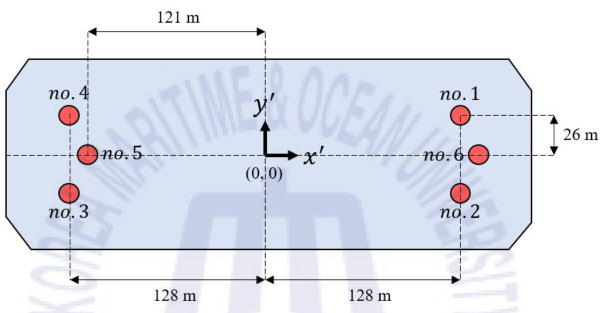


Fig. 81 Arrangement of DP thrusters

Table 20 Details of DP thrusters

Azimuth Thruster	From x' [m]	From y' [m]	Maximum Thrust Limit [kN]
no.1	128	26	2000
no.2	128	-26	2000
no.3	-128	-26	2000
no.4	-128	26	2000
no.5	-121	0	2000
no.6	135	0	2000

5.2.1. PID Control in DPS

A control law of a single PID controller is shown in Eq. (110) where u_t is the low-level controller output at time t, e_t is an error which is defined as a reference position subtracted by a current position, k_p, k_i, and k_d are the proportional, integral, and derivative gains of the PID controller, respectively.

$$u_t = k_p e_t + k_i \int_0^t e_t \, dt + k_d \frac{de_t}{dt} \tag{110}$$

In case of the PID controller in the DPS, since there are three directional motions (surge, sway, and yaw) that need to be maintained by the DPS, three PID controllers are used in the DP's PID control system. The structure of the PID control system in the DPS is shown in **Fig. 82**. In this figure, $\boldsymbol{x}_{\text{ref}}$ is a vector of reference ship positions in the surge, sway, and yaw directions. $\boldsymbol{e}_t$ is a vector of the errors in the surge, sway, and yaw directions at time t, $T_t^{(\cdot)}$ is a required thrust in each direction where the direction is denoted in the superscript parenthesis, and each T_t denotes the same as u_t. $\boldsymbol{T}_t$ is a vector of the allocated thrusts for each thruster, and $\boldsymbol{x}_{\mathbf{t+1}}$ is a vector of current ship positions in the surge, sway, and yaw directions at time $t + 1$. The details of $\boldsymbol{x}_{\text{ref}}$, $\boldsymbol{e}_t$. $\boldsymbol{T}_t$, and $\boldsymbol{x}_{\mathbf{t+1}}$ are shown in Eqs. (111)-(114). In $\boldsymbol{T}_t$, the number next to T represents a thrust number. For $\boldsymbol{x}_{\mathbf{t+1}}$, it should be noted that the ship positions are the positions filtered by a Kalman filter which is used to remove the ship motion components induced by the 1st-order wave load (Fossen and Perez, 2009).

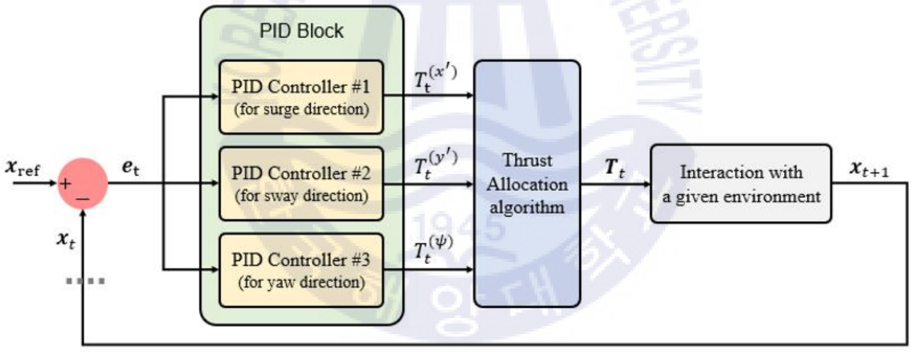

PID Block
PID Controller #1 (for surge direction)
PID Controller #2 (for sway direction)
PID Controller #3 (for yaw direction)
$T_t^{(x')}$
$T_t^{(y')}$
$T_t^{(\psi)}$
Thrust Allocation algorithm
$\boldsymbol{T}_t$
Interaction with a given environment
$\boldsymbol{x}_{\text{ref}}$
$\boldsymbol{e}_t$
$\boldsymbol{x}_t$
$\boldsymbol{x}_{t+1}$
+
−

Fig. 82 Structure of the PID control system in the DPS

$$\boldsymbol{x}_{\text{ref}} = \{x'_{\text{ref}}, y'_{\text{ref}}, \psi_{\text{ref}}\} \tag{111}$$

$$\boldsymbol{e}_t = \boldsymbol{x}_{\text{ref}} - \boldsymbol{x}_t = \{(x'_{\text{ref}} - x'_t), (y'_{\text{ref}} - y'_t), (\psi_{\text{ref}} - \psi_t)\} = \left\{e_t^{(x')}, e_t^{(y')}, e_t^{(\psi)}\right\} \tag{112}$$

$$\boldsymbol{T}_t = \left\{T1_t^{(x')}, T1_t^{(y')}, T2_t^{(x')}, T2_t^{(y')}, \cdots, T6_t^{(x')}, T6_t^{(y')}\right\} \tag{113}$$

$$\boldsymbol{x}_{\mathbf{t+1}} = \{x'_{t+1}, y'_{t+1}, \psi_{t+1}\} \tag{114}$$

In **Fig. 82**, the flow of the control system is as follows: First, $\boldsymbol{e}_t$ is calculated based on $\boldsymbol{x}_{\text{ref}}$ and $\boldsymbol{x}_t$, and it is fed into each of the three PID controllers according to their directions. Then, each PID controller outputs $T_t^{(.)}$ and it is fed into the thrust allocation algorithm to output $\boldsymbol{T}_t$. For the thrust allocation algorithm for the DP thrust, a Lagrange Multiplier method is used (De Wit, 2009).

5.2.2. Hydrodynamics Associated with a Drifting Motion of a Ship

This sub-subchapter is present to provide an understanding of the environmental loads that contribute to the ship's drifting motion, and how the DP thrust counteract the environmental loads.

The mathematical model of the ship motion can be expressed as Eq. (115) (Fossen and Perez, 2009),

$$M(p,a) + C(p,v) + K(p) = F(p,v,t) \tag{115}$$

where $M(p,a)$ is the system inertia load, $C(p,v)$ is the system damping load, $K(p)$ is the system stiffness load, $F(p,v,t)$ is the external load, p, v and a are the position, velocity and acceleration, respectively, and t is time. For the DPS, the external load $F(p,v,t)$ is equal to Eq. (116),

$$F(p,v,t) = \tau_{\text{wind}} + \tau_{\text{current}} + \tau_{\text{wave}} - \tau_{\text{thrusters}} \tag{116}$$

where τ represents external loads and subscripts represent sources of the external loads.

There are three types of environmental loads contributing to the drifting motion of a ship such as wind, current, and wave drift loads. The environmental loads contributing to the drifting motion are presented in Fig. 83. The wave load is divided into the 1st order and 2nd order wave loads. The difference between these two loads is shown by ship motions affected by these two loads in Fig. 84 (Hassani and Pascoal, 2015).

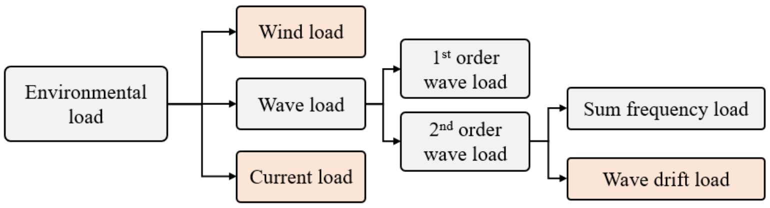


Fig. 83 Environmental loads contributing to the drifting motion

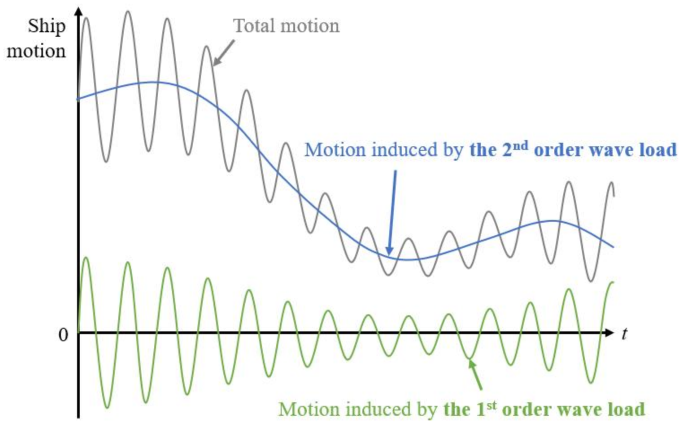


Fig. 84 Difference between the 1st order and 2nd order wave loads

For the DPS to compensate for the drifting motion caused by the wind, current, and wave drift loads, the DP thrusters need to counteract the environmental loads. This relation can be expressed as Eq. (117) in which the DPS basically maintains a ship's position by trying to satisfy this equation.

$$(\tau_{\text{wind}} + \tau_{\text{current}} + \tau_{\text{wave drift}}) - \tau_{\text{thrusters}} = 0 \tag{117}$$

The wind and current loads can be expressed as Eqs. 오류! 참조 원본을 찾을 수 없습니다.-(121) (Fossen and Perez, 2009), and the wave drift load can be expressed as Eq. (122) (Orcina, 2019).

$$\tau_{\text{wind}} = \frac{1}{2}\rho_{\text{air}}V_r^2 \begin{bmatrix} C_{\text{surge}}A_{\text{surge}} \\ C_{\text{sway}}A_{\text{sway}} \\ C_{\text{yaw}}A_{\text{yaw}} \end{bmatrix} \tag{118}$$

$$A_{\text{yaw}} = L_{BP}A_{sway} \tag{119}$$

$$\tau_{\text{current}} = \frac{1}{2}\rho_{\text{water}}V_r^2 \begin{bmatrix} C_{\text{surge}}A_{\text{surge}} \\ C_{\text{sway}}A_{\text{sway}} \\ C_{\text{yaw}}A_{\text{yaw}} \end{bmatrix} \tag{120}$$

$$A_{\text{yaw}} = L_{BP}^2 D \tag{121}$$

where τ_{wind} and τ_{current} are wind and current loads, respectively, $C_{\text{surge}}, C_{\text{sway}}, C_{\text{yaw}}$ are the surge, sway and yaw coefficients with respect to the wind or current direction relative to a ship heading, ρ denotes density, V_r is a relative velocity of wind or current past a ship, $A_{\text{surge}}, A_{sway}$ are the exposed areas above the waterline for τ_{wind} and the submerged areas for τ_{current}. L_{BP} is length between perpendiculars, and D is a draft.

$$\tau_{\text{wave drift}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \text{Re}\{\text{QTF}_{\text{wave drift}}(\beta_i, \beta_j, T_i, T_j) a_i a_j \exp[i(\omega_i - \omega_j)t - (\phi_i - \phi_j)]\} \tag{122}$$

where n is the number of the regular wave components for describing irregular sea state, Re denotes the real part of a complex number, $\text{QTF}_{\text{wave drift}}$ is the wave drift quadratic transfer function which can be obtained from a diffraction analysis in the frequency domain on the target ship, β is the direction of the regular wave component relative to the ship's heading, T, a, ω, ϕ are the wave period, amplitude, frequency, and phase lag of the regular wave component. Eq. (122) shows that $\text{QTF}_{\text{wave drift}}$ is applied to each pair of the regular wave components to yield each pair's contribution to the wave drift load.

5.3. Proposed Adaptive Fine-Tuning System for PID Gains in DPS

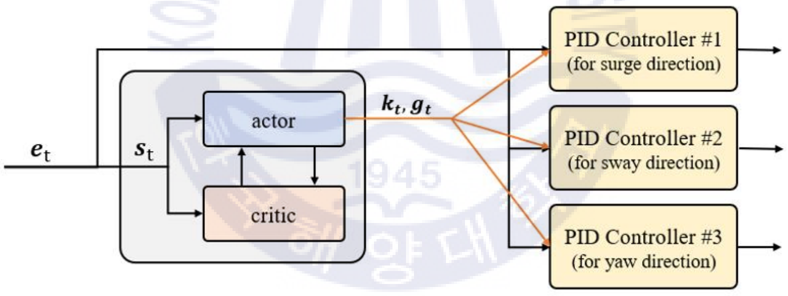


Fig. 85 Overall structure of the proposed adaptive fine-tuning system

An overall structure of the proposed adaptive fine-tuning system is shown in Fig. 85. For the intuitive understanding, Fig. 85 is drawn on the top of **오류! 참조 원본을 찾을 수 없습니다.**. In this figure, the state $\boldsymbol{s}_t$ is an input vector for the actor and critic, $\boldsymbol{k}_t$ is a vector of the adaptive PID gains, and $\boldsymbol{g}_t$ is an update-gate for the integral of the errors. A configuration of $\boldsymbol{s}_t$ is shown in Eq. (123). The previous motion time histories are used in $\boldsymbol{s}_t$ to consider a memory effect in which a fluid load is determined in part by previous motions at any instant (Yeung, 1983). The integrals of the errors are also included in $\boldsymbol{s}_t$ since they are influential terms for the mean station-keeping position, and also needed to provide enough information to output the update-gate for the integral of the errors $\boldsymbol{g}_t$. During the training of the actor and critic, a normalized $\boldsymbol{s}_t$ is used to ensure the training stability (Lee,

Lee and Seo, 2020b). For the normalization of $\boldsymbol{s}_t$, the error terms in $\boldsymbol{s}_t$ are normalized by Eq. (124) where $\mathcal{S}$ is all the states stored in the replay buffer and the std stands for standard deviation, and the integral of the error terms in $\boldsymbol{s}_t$ are normalized by dividing them by 100 to make them small enough for the stable neural network training. The integral of the error terms are differently normalized because these terms usually increase and gradually decrease over time unlike the error terms which go around their mean values with certain standard deviations since the ship position goes around its reference position by the DPS.

$$\boldsymbol{s}_t = \left\{ \begin{matrix} e_{t-9}^{(x')}, e_{t-8}^{(x')}, \cdots, e_{t}^{(x')}, \\ e_{t-9}^{(y')}, e_{t-8}^{(y')}, \cdots, e_{t}^{(y')}, \\ e_{t-9}^{(\psi)}, e_{t-8}^{(\psi)}, \cdots, e_{t}^{(\psi)} \\ \sqrt{\left| \int_0^t e_t^{(x')} dt \right|}, \sqrt{\left| \int_0^t e_t^{(y')} dt \right|}, \sqrt{\left| \int_0^t e_t^{(\psi)} dt \right|} \end{matrix} \right\} \tag{123}$$

$$\boldsymbol{s}'_t = \left(\boldsymbol{s}_t - \text{mean}(\mathcal{S})\right) / \text{ std}(\mathcal{S}) \tag{124}$$

The actor basically outputs $\boldsymbol{k_t}$ and $\boldsymbol{g_t}$, at every timestep. $\boldsymbol{k_t}$ consist of the adaptive PID gains for the surge, sway, and yaw directions. The details of $\boldsymbol{k_t}$ is shown in Eq. (125) where each $\boldsymbol{k}_t^{(.)}$ consist of the adaptive P, I, and D gains in each direction as shown in Eq. (126). $\boldsymbol{k}_{\text{base}}$ is a vector of the base gains determined based on the ZN method, and $\boldsymbol{a}_{\text{t}}$ is the output vector from the actor's output layer. In this paper, the actor uses a sigmoid activation function in its output layer. The sigmoid activation function is shown in Eq. (127). Therefore, ranges of each component in $\boldsymbol{a}_{\text{t}}$ and $\boldsymbol{k}_{\text{t}}$ are $[0,1]$ and $[0, \boldsymbol{k}_{\text{base}}]$, respectively. With $\boldsymbol{k_t}$, the PID controller with the adaptive gains is expressed as Eq. (128) where the direction is denoted in the superscript parenthesis.

$$\boldsymbol{k}_t = \left\{ \boldsymbol{k}_t^{(x')}, \boldsymbol{k}_t^{(y')}, \boldsymbol{k}_t^{(\psi)} \right\} = \boldsymbol{a}_{\text{t}} \times \boldsymbol{k}_{\text{base}} \tag{125}$$

$$\boldsymbol{k}_t^{(.)} = \left\{ k_p^{(.)}(t), k_i^{(.)}(t), k_d^{(.)}(t) \right\} \tag{126}$$

$$\phi(z) = 1/(1 + e^{-z}) \tag{127}$$

$$T_t^{(.)} = k_p^{(.)}(t) e_t^{(.)} + k_i^{(.)}(t) \int_0^t e_t^{(.)} \, dt + k_d^{(.)}(t) \frac{de_t}{dt} \tag{128}$$

The update-gate for integral of the errors $\boldsymbol{g_t}$ determines how much error to update for the integral of the errors which is in the integral terms in the PID controllers. The update equation of the integral of errors by $\boldsymbol{g_t}$ is shown in Eq. (129) where $\boldsymbol{g_t} \in [0,1]$. The concept of $\boldsymbol{g_t}$ was inspired by gated recurrent units (GRU) (Chung *et al.*, 2014).

$$\int_0^t \boldsymbol{e}_t \, dt = \int_0^{t-1} \boldsymbol{e}_t \, dt + (\boldsymbol{g_t} \times \boldsymbol{e}_t) \tag{129}$$

$$\boldsymbol{g}_t = \left\{ g_t^{(x')}, g_t^{(y')}, g_t^{(\psi)} \right\} \tag{130}$$

$\boldsymbol{g}_t$ is designed to prevent a large rebound motion of a ship. When a sea state is severe, a ship experiences large drifting motions on a drifting side mostly due to aggressive wave drift loads. An illustration of the drifting side and an example of the wave drift load time history are shown in **Fig. 86** and Fig. 87, respectively. While a ship is drifted greatly due to the very large wave drift load, the integral of the error increases drastically as well. Due to the drastically-increased integral of the error over many drifting motions, the large rebound ship motions are caused. The large rebound ship motions caused by the increased integral of the error are illustrated in Fig. 88.

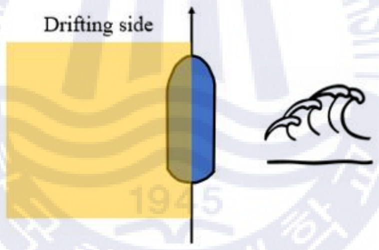


Fig. 86 Illustration of the drifting side (direction of an environmental load: right to left)

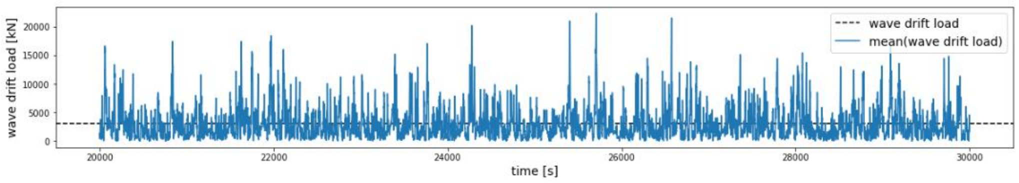


Fig. 87 Example of the wave drift load time history (sea state: very rough, environmental direction to a ship: 40°)

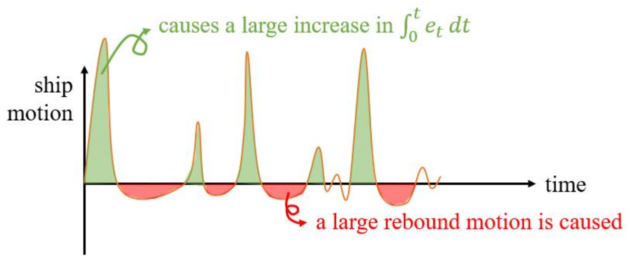

causes a large increase in $\int_0^t e_t\,dt$
ship motion
time
a large rebound motion is caused

Fig. 88 Illustration of the large rebound motion caused by the drastic increase of the integral of error

As to the adaptive gains from the actor, the I gain is fixed when a ship is on the drifting side, while the P and D gains are adaptively selected by the actor. The I gain is fixed on the drifting side to ensure the convergence of the station-keeping. On the other side of the drifting side, the I gain is adaptively selected. The drifting side can easily be identified by the integral of the error.

Architectures of the actor and critic are shown in Fig. 89 where the HL stands for a hidden layer, the BN stands for batch normalization (BN) proposed by Ioffe and Szegedy (2015), and the numbers below the HL refer to the size of the HL. Unlike the original DDPG implementation from Lillicrap et al. (2016) that uses the BN in both the actor and critic, the BN is used in the critic only for the proposed adaptive fine-tuning system. The choice of application of the BN to the actor and critic was made based on the station-keeping performance assessment in simulations. Given the study by Bhatt et al. (2019), it is natural that application of the BN does not always increase the RL learning performance but the performance may increase or decrease depending on application domains.

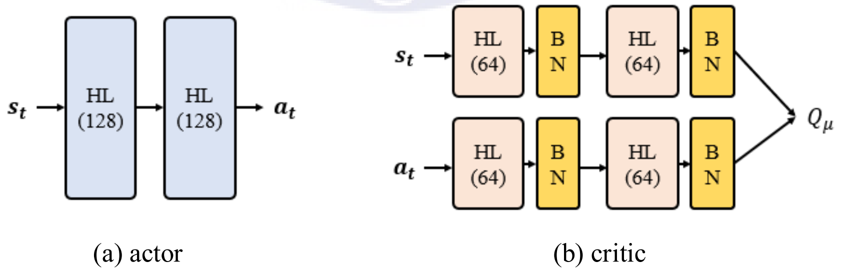


Fig. 89 Architectures of the actor and critic

The reward function that is needed to train the actor and critic is defined as Eq. (131). It should be noted that the station-keeping in the yaw direction is implicitly designed since the good station-keeping of the surge and sway motions requires the good station-keeping of the yaw motion. To further stabilize the RL learning, r_t is clipped to $[0,2]$ and r_t is normalized as Eq. (132) where $\mathcal{R}$ denotes all the rewards stored in the replay buffer. For the action noise, a Gaussian action noise with the std of 0.1 is used instead of an Ornstein-Uhlenbeck process since they result in the similar learning performance (Plappert *et al.*, 2018). Finally, the other hyper-parameters for the DDPG are set the same as in the original DDPG implementation (Lillicrap *et al.*, 2016).

$$r_t = -\sqrt{\left(e_t^{(x')}\right)^2 + \left(e_t^{(y')}\right)^2} \tag{131}$$

$$r_t' = \left(r_t - \text{mean}(\mathcal{R})\right)/\text{std}(\mathcal{R}) \tag{132}$$

5.4. Simulation Results

In the simulations, the equation of ship motion is solved by using a marine dynamic simulation software, OrcaFlex, developed by Orcina. As to the sea states, because the station-keeping performance by the gain change does not vary much in calm or moderate sea states due to small drifting motions, the simulations were conducted in ‘very rough’, ‘high’, and ‘very high’ sea states according to Table 21 (Nguyen, Sørensen and Tong Quek, 2007). For the station-keeping, its reference positions (surge x', sway y', yaw ψ) are set to (0, 0, 0).

Table 21 Definition of sea states

Sea states	Significant wave height H_s [m]
Calm	0 - 0.1
Moderate	1.25 - 2.5
Rough	2.5 – 4.0
Very rough	4.0 – 6.0
High	6.0 – 9.0
Very high	9.0 – 14.0

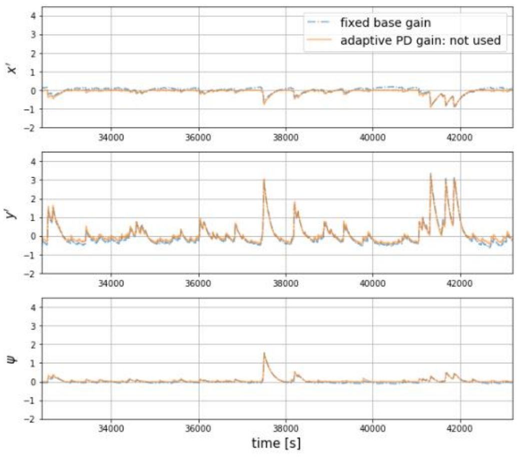
5.4.1. Effectiveness of the Proposed Adaptive Fine-Tuning System

In the simulations in this sub-subchapter, the sea state is set to ‘very rough’. The details of the environmental condition of the sea state are shown in Table 22 where T_z, V_w, and V_c denote a zero-crossing period, mean current speed, and mean wind speed, respectively. The specific values for the environmental condition are determined by a IMCA standard (IMCA, 2000).

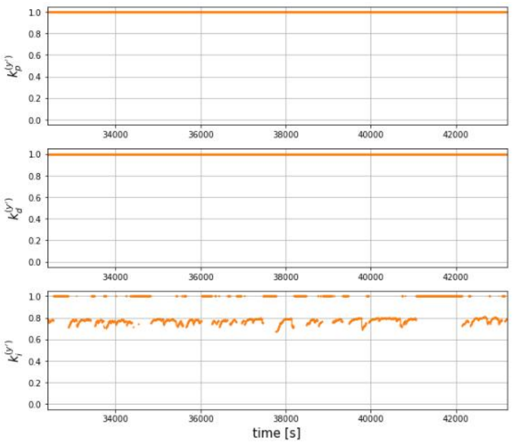
Table 22 Environmental condition for the effectiveness test

Sea state	Wave spectrum	H_s [m]	T_z [m]	V_w [m/s]	V_c [m/s]	Wind, wave, current direction	Environmental direction to the ship
Very rough	JONSWAP	5.0	8.16	14.75	1	co-linear	40°

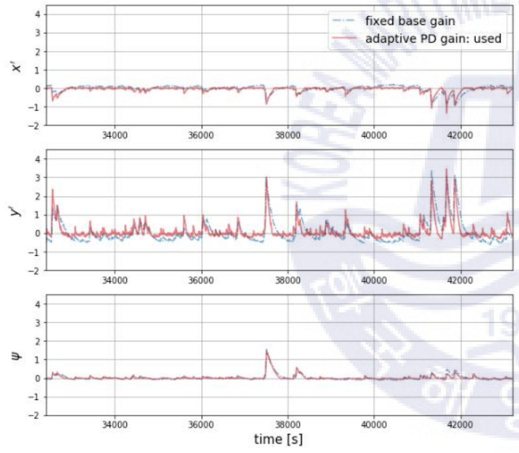
First, the effects of the use of the adaptive P, D gains are presented with ship motion and the gain time histories in Fig. 90 where a PID controller with the base gain (fixed) and a PID controller with the proposed adaptive fine-tuning system are compared. For the adaptive fine-tuning system in Fig. 90, one adaptive fine-tuning system is not allowed to tune the PD gains but the I gain only, and the other is allowed to tune all the PID gains. For the gain histories, since a significant difference lies in the sway motion in Fig. 90, the gain histories regarding the sway direction are presented only, and the gain histories are normalized by division by $\boldsymbol{k}_{\text{base}}$. For the training of the adaptive fine-tuning systems, they are trained for 12-hour timesteps in the simulation. In Fig. 90, the 'fixed base gain' represents the PID controller with the fixed base gain, the 'adaptive PD gain: not used' represents the PID controller with the adaptive fine-tuning system that tunes the I gain only, and the 'adaptive PD gain: used' represents the PID controller with the adaptive fine-tuning system that tunes all the PID gains. In Fig. 90, the distinct difference between 'adaptive PD gain: not used' and 'adaptive PD gain: used' is the convergence speed. In case of the 'adaptive PD gain: used', it tends to approach the reference position faster than 'adaptive PD gain: not used' and it has smaller rebound motions, especially in the sway direction. As observed in Fig. 90 (c)-(d), in case of the 'adaptive PD gain: used', when the ship is drifted, the PD gains are increased to prevent the drifting motion, and while the ship approaches the reference position, the P gain is drastically decreased and the D gain is slightly decreased, which makes the fast convergence possible. Because the drastic reduction of the P gain contributes to the prevention for the rebound motion while the D gain which is relatively lower than the base gain by ranging from 0.4 to 0.6 of the base gain allows the fast approach to the reference position.



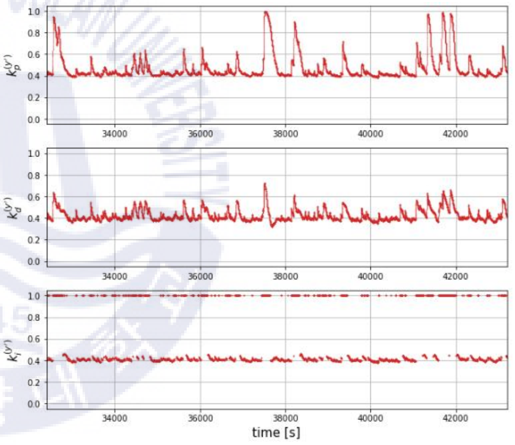
(a) adaptive PD gain: not used



(b) gain histories of 'adaptive PD gain: not used'



(c) adaptive PD gain: used

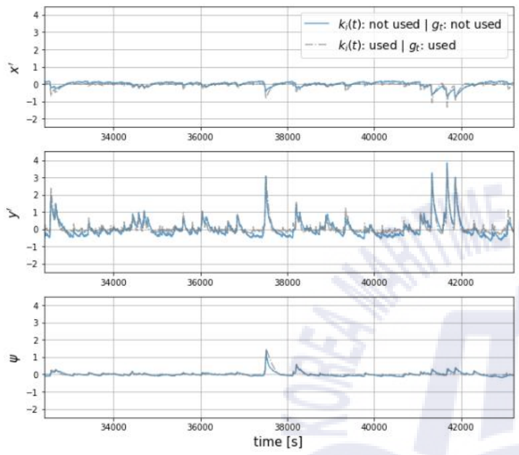


(d) gain histories of 'adaptive PD gain: used'

Fig. 90 Effects of the adaptive PD gains

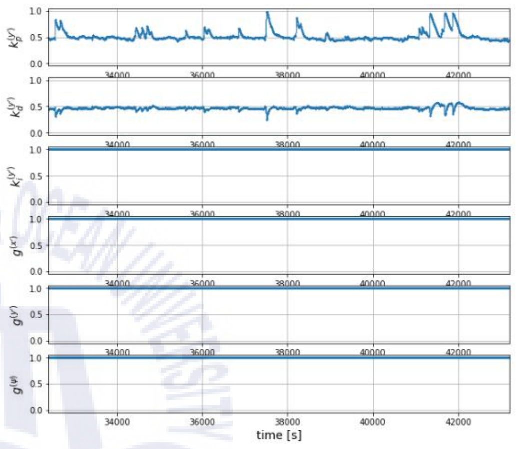
Second, the effects of the adaptive I gain and the update-gate for the integral of the errors $\boldsymbol{g}_t$ are analyzed in **Fig.** ***91*** where $k_i(t)$ represents the adaptive I gain. In **Fig.** ***91***, the ship motion time histories are presented on the left side, and the time histories of the adaptive gains and $\boldsymbol{g}_t$ are presented on the right side. Same as in **Fig. 90**, the gain histories are normalized by division by $\boldsymbol{k}_{\text{base}}$, and the training of the adaptive fine-tuning systems is conducted for 12-hour timesteps in the simulation. In **Fig.** ***91*** (a), it is clearly seen that the rebound motion is reduced when $k_i(t)$ and $\boldsymbol{g}_t$ are used. In **Fig.** ***91*** (c) and (g), the pattern of $k_i^{(y')}$ can be observed, in which $k_i^{(y')}$ are tuned to around

0.5 while the ship experiences the rebound motion. This is a reasonable control for $k_i^{(y')}$ because the drifting side is in the positive direction in the sway motion, then its integral of the error is negative. Thus, the small $k_i^{(y')}$ helps the ship to reach the reference position faster while the ship is in the rebound side (negative) by reducing the negative thrust contributed by the integral of the error. **Fig. *91*** (e) shows that the adaptive fine-tuning system does not learn the adaptive gain-tuning strategy well when $k_i(t)$ is not used and $\boldsymbol{g}_t$ are used.



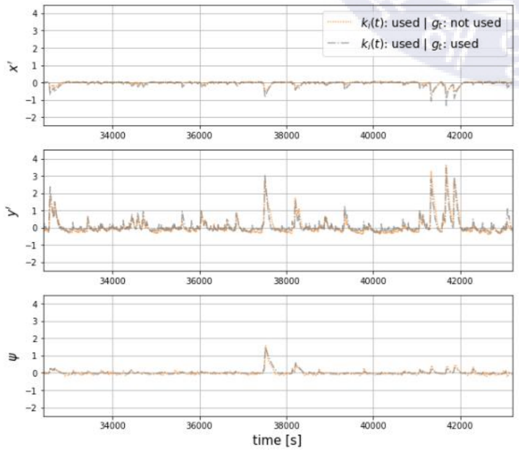
(a) ship motion time histories

[with respect to $k_i(t)$ and $\boldsymbol{g}(t)$]



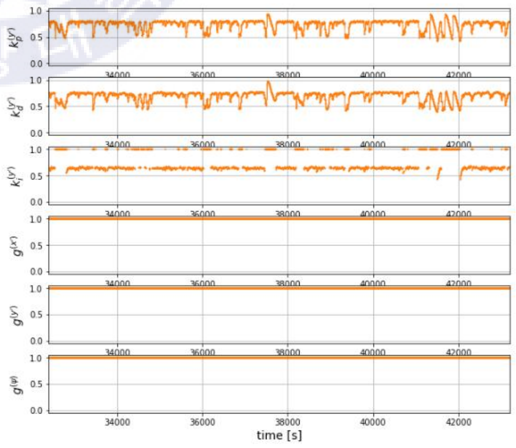
(b) time histories of the adaptive gain and $\boldsymbol{g}_t$

[$k_i(t)$: not used | $\boldsymbol{g}(t)$: not used]



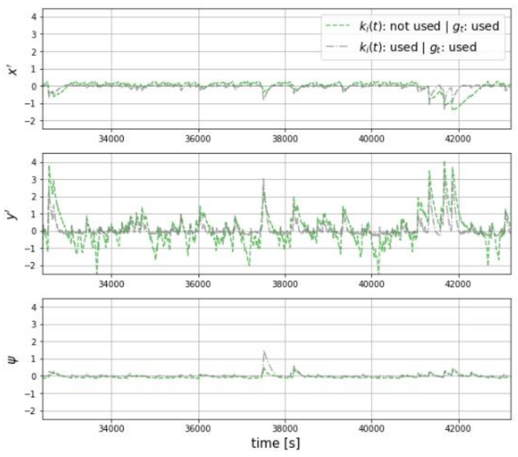
(c) ship motion time histories

[with respect to $\boldsymbol{g}(t)$]



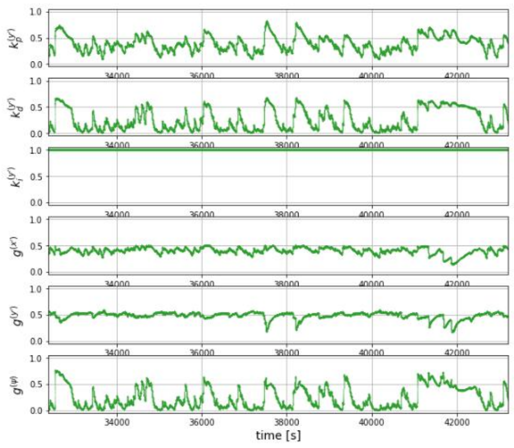
(d) time histories of the adaptive gain and $\boldsymbol{g}_t$

[$k_i(t)$: used | $\boldsymbol{g}(t)$: not used]



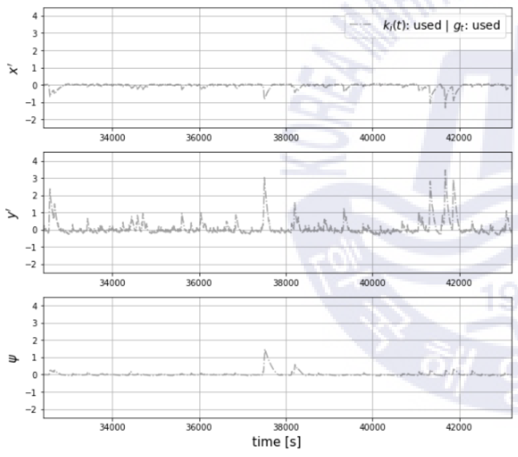
(e) ship motion time histories

[with respect to $k_i(t)$]



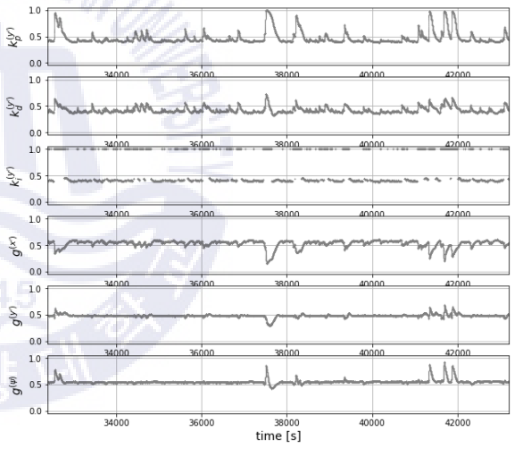
(f) time histories of the adaptive gain and $\boldsymbol{g}_t$

[$k_i(t)$: not used | $\boldsymbol{g}(t)$: used]



(g) ship motion time histories

[$k_i(t)$: used | $\boldsymbol{g}(t)$: used]



(h) time histories of the adaptive gain and $\boldsymbol{g}_t$

[$k_i(t)$: used | $\boldsymbol{g}(t)$: used]

Fig. 91 Effects of the adaptive I gain and the update gate of the integral of error

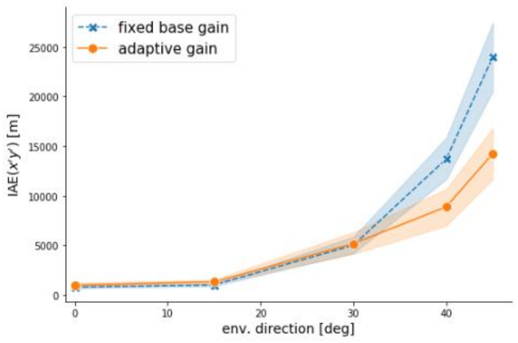
5.4.2. Overall Performance Assessment

The overall performance of the adaptive fine-tuning system is presented with respect to the station-keeping performance and the control efficiency. The higher control efficiency means the lower thrust consumption and a lower rate of the thrust change. The sea states considered here are ‘very rough’, ‘high’, and ‘very high’. The details of the environmental conditions are shown in **Table 23**. For the environmental directions, the maximum environmental directions in each sea state are determined based on a watch circle with its radius of 10m (Weingarth, 2006).

Table 23 Environmental condition for the overall performance test

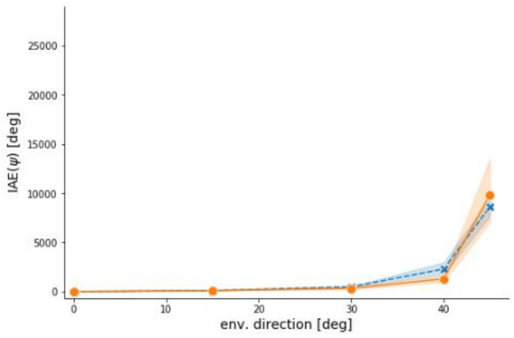
Sea state	Wave spectrum	H_s [m]	T_z [m]	V_w [m/s]	V_c [m/s]	Wind, wave, current direction	Environmental direction to the ship
Very rough	JONSWAP	5.0	8.16	14.75	1	co-linear	$0°, 15°, 30°, 40°, 45°$
High	JONSWAP	7.5	9.87	20.0	1	co-linear	$0°, 15°, 30°, 35°$
Very high	JONSWAP	9.06	11.04	23.75	1	co-linear	$0°, 15°, 30°, 35°$

The performance assessment is conducted with respect to each environmental condition considering the sea states and environmental directions to examine the proposed system's learning capability on each environmental condition. Then, a total number of the environmental conditions for the performance assessment is 13 (= 5 + 4 + 4). For the training of the adaptive fine-tuning system, the training is conducted for 12-hour timesteps in each environmental condition individually. The overall performance assessment of the adaptive fine-tuning system compared to the PID controller with the fixed base gain is presented in **Fig. 92**. The station-keeping performance assessment is presented with the line graphs with the IAE performance index. The IAE stands for the integral absolute error and it is a traditional performance index (Jon and Breivik, 2009). The control efficiency assessment is presented with violin graphs to present the distributions of the thrust and a rate of thrust change. In **Fig. 92**, the 'fixed base gain' and 'adaptive gain' refer to the PID controller with the fixed base gain and the PID controller with the adaptive fine-tuning system, respectively. IAE$(x'y')$ is equal to IAE$\left(\sqrt{(x')^2 + (y')^2}\right)$. $\bar{T}$ and $d\bar{T}/dt$ denote a mean absolute value of $\boldsymbol{T}$ from Eq. (113) and a time derivative of $\bar{T}$, respectively. The performances indices IAE$(x'y')$, IAE(ψ), $\bar{T}$, and $d\bar{T}/dt$ indicate the station-keeping performance in the surge and sway directions, the station-keeping performance in the yaw direction, the DP thrust consumption, and a rate of the thrust change, respectively. The results in **Fig. 92** are obtained from five 1h-simulations with different seeds. Finally, in the line graphs for the IAE, the dots denote the mean values and shaded area denotes a 95% confidence interval.



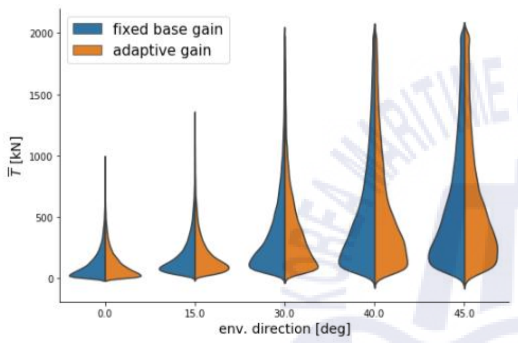
(a) IAE$(x'y')$

[sea state: 'very rough']



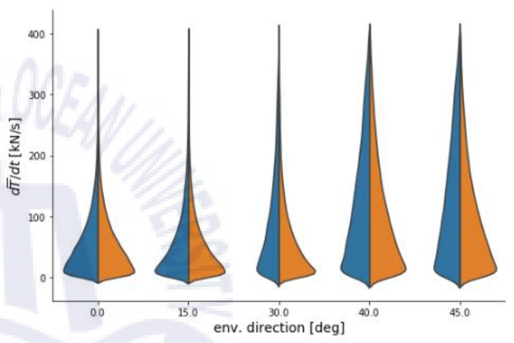
(b) IAE(ψ)

[sea state: 'very rough']



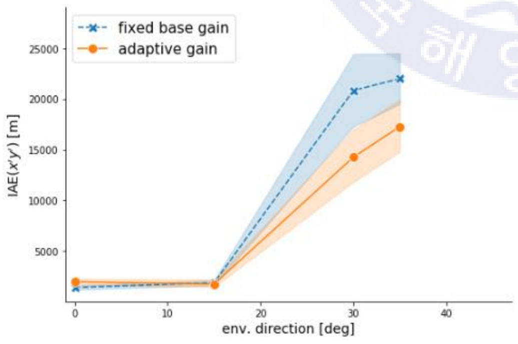
(c) distribution of $\overline{T}$

[sea state: 'very rough']



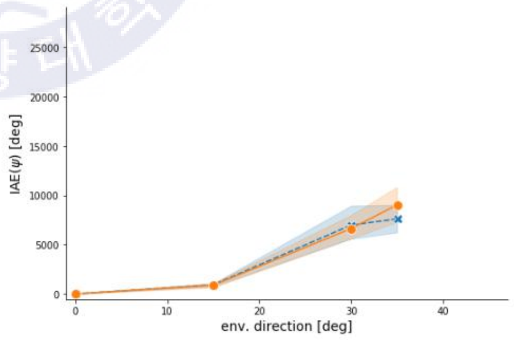
(d) distribution of $d\overline{T}/dt$

[sea state: 'very rough']



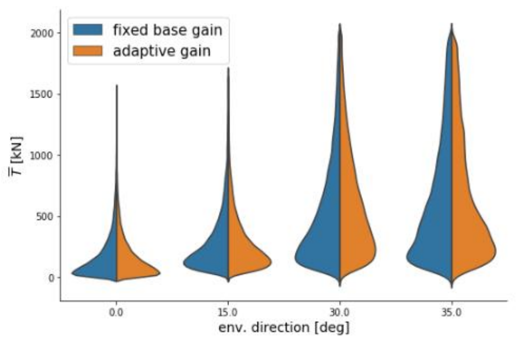
(e) IAE$(x'y')$

[sea state: 'high']



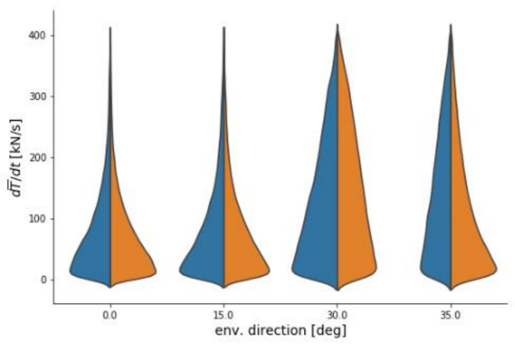
(f) IAE(ψ)

[sea state: 'high']



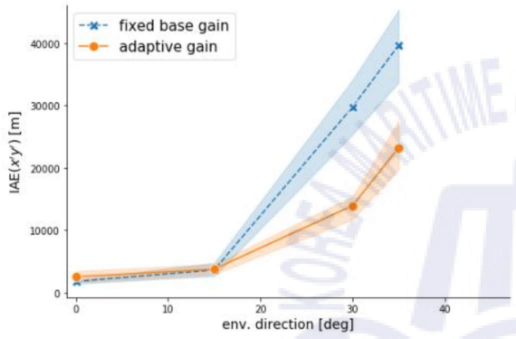
(g) distribution of $\overline{T}$

[sea state: 'high']



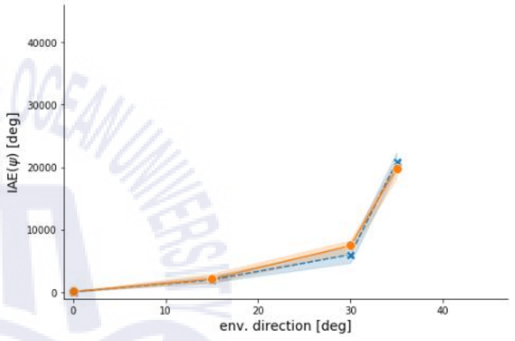
(h) distribution of $d\overline{T}/dt$

[sea state: 'high']



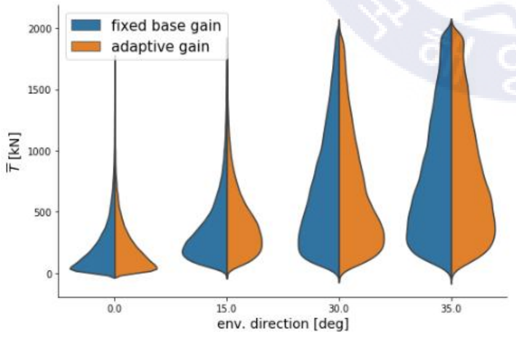
(i) IAE($x'y'$)

[sea state: 'very high']



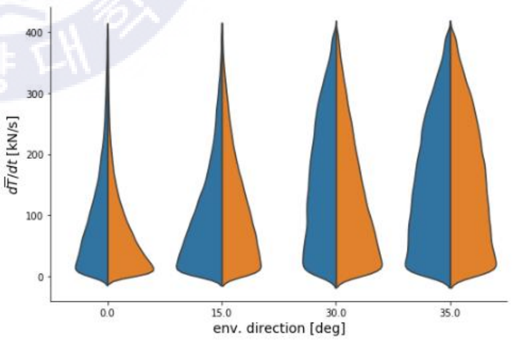
(j) IAE(ψ)

[sea state: 'very high']



(k) distribution of $\overline{T}$

[sea state: 'very high']



(l) distribution of $d\overline{T}/dt$

[sea state: 'very high']

Fig. 92 The overall performance assessment of the adaptive fine-tuning system

There are two distinct characteristics in the results in Fig. 92. First, when the environmental direction is below around 15°, there is almost no performance difference between the 'fixed base gain' and ‘adaptive gain’. This is because when the environmental direction to the ship is small, the ship experiences the relatively small wave drift loads than when the environmental direction to the ship is large, and the small wave drift loads result in the small drifting motions. The small drifting motions are equal to the small error. Therefore, no matter how the gains are tuned by the adaptive fine-tuning system, the output DP thrust does not change much, which corresponds to the no performance change. For the intuitive understanding of the wave drift load with respect to the environmental directions, the wave drift loads with respect to the environmental directions in the ship’s equilibrium position in the ‘high’ sea state are presented in Fig. 93 where $F_{x'}$, $F_{y'}$, and M_{ψ} denote the load in the surge direction, the load in the sway direction, and the load in the yaw direction, respectively.

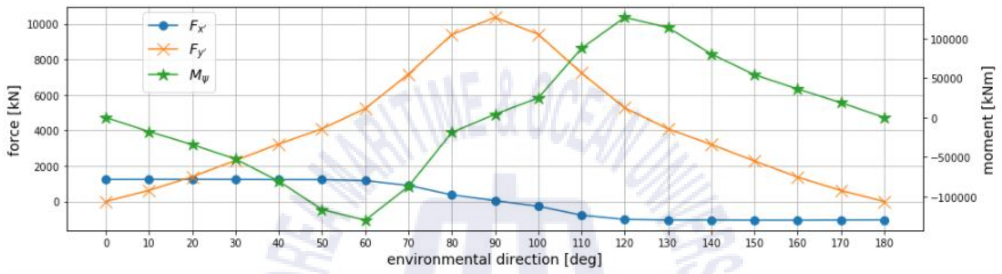


Fig. 93 Wave drift loads with respect to the environmental directions (sea state: ‘high’)

Second, when the environmental direction is over around 15°, the difference in the station-keeping performance becomes apparent. Overall, when the environmental direction is over around 15 deg, the adaptive fine-tuning system results in the better station-keeping performance in the surge and sway directions and the equal station-keeping performance in the yaw direction with the similar thrust consumption distribution and the similar rate of the thrust change. Since the goal of the proposed system is to learn the efficient adaptive gain-tuning strategy to increase the station-keeping performance without deterioration in the control efficiency, the proposed system achieves its goal.

5.5. Discussion

The proposed adaptive fine-tuning system achieves its goal of learning the efficient adaptive gain-tuning strategy to increase the station-keeping performance without deterioration in the control efficiency. However, while the station-keeping performance increases in the surge and sway directions, the performance in the yaw direction remains almost the same. We originally thought that it is because the reward function does not have explicit information on the station-keeping for the yaw motion. Therefore, we tried adding a penalty term for the yaw motion error in the existing reward function. However, it only made the yaw motion somewhat aggressively oscillating, which made the station-keeping in the surge and sway directions unstable as well. To solve that issue, we needed to add time derivates of the surge, sway, and yaw motion errors, and additional efforts for the weight tuning for each term. At that point, the reward function became too complicated. Therefore, we kept the reward function as simple as possible as presented in this paper.

The proposed adaptive fine-tuning system allows the ship to converge to the reference position quickly after drifted. However, the magnitudes of the 'peaky' drifting motions remain the same. These peaky drifting motions are caused when the wave drift load is suddenly strong as shown in Fig. 87. To reduce the magnitude of the peak drifting motions, we tried adopting a machine learning-based anomaly detection algorithm to detect the moment when the peaky drifting motion is about to happen, and when detected, the base gain multiplied by a multiplying factor of 10 was set to be used to reduce the magnitude of the peaky drifting motion as a part of the adaptive fine-tuning system. Although the anomaly detection algorithm worked well for the detection, when the wave drift loads were suddenly strong, this approach did not help much. We suppose that utilization of a feed-forward system would be much more effective to reduce the magnitude of the peaky drifting motion.

The proposed adaptive fine-tuning system utilizes a PID controller and thrust allocation algorithm to yield the DP thrusts from the six azimuth thrusters, which is designed to cope with the varying environmental loads more efficiently than using the fixed gains. Then, one might have this question: 'what if the deep reinforcement learning algorithm learns to control the DP thrusts directly so that the ship can cope with the varying environmental loads efficiently as an end-to-end learning system?' We had this exact idea and experimented it in simulations. However, the end-to-end learning did not perform well. Here are the reasons why we suppose did not go well: 1) In the end-to-end learning, a number of actor's output increases in proportion to a number of thrusters. In other words, one thruster needs to be controlled by a magnitude of thrust and thruster angle, therefore, six thrusters require 12 actions. But if we impose the PID structure, a number of the output is fixed to nine. (three gains for the surge motion, three gains for the sway motion, and three gains for the yaw motion). This gives the end-to-end learning a higher dimension in the action space. 2) When a PID structure is used with a thrust allocation algorithm, the thrust allocation algorithm can take care of the change rates of thrust and angle of each thruster. However, in case of the end-to-end learning, the deep reinforcement learning algorithm needs to learn not to drastically change the thrust and an angle of each thruster, or the change rate limit can be imposed on the action of the actor but this would discourage the exploration process. 3) When a PID structure is used with a thrust allocation algorithm, the allocation algorithm can ensure that a target total thrust is well distributed over the available thrusters not to give one thruster a lot of load and let other thrusters to be idle. However, in case of the end-to-end learning, the deep reinforcement learning algorithm needs to learn the thrust allocation from scratch, which adds the complexity in learning. 4) With a PID structure, the directionality in the control system for the thrust is simpler as shown in Fig. 94 and Eqs. (133)-(134), where the red star denotes a reference position and $\mathrm{PID}(.)$ represents a PID controller. It should be noted that only P gain is considered in the sway direction for the example's simplicity in Fig. 94. In case of the PID controller, the same thrust with its proper direction can be output with the same gain as shown in Eq. (133). But in case of the end-to-end learning, the deep reinforcement learning algorithm needs to learn the directionality of the thrust, which adds the additional complexity in learning as shown in Eq. (134).

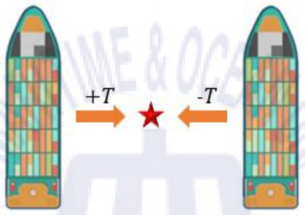


Fig. 94 Directionality of thrust in the control system

$$\vec{T} = \mathrm{PID}(e, k_p) \tag{133}$$

$$\vec{T} = \begin{cases} +T = [T\cos(\theta), T\sin(\theta)] \\ -T = [T\cos(\theta + 2\pi), T\sin(\theta + 2\pi)] \end{cases} \tag{134}$$

Finally, this study shows that the adaptive fine-tuning system can learn the effective adaptive gain-tuning strategy in each environmental condition. Therefore, it is important that the proposed system should be further developed to perform well in various environmental conditions. We suggest the following two feasible approaches: 1) training the proposed system on all the environmental conditions at once, 2) utilizing a switching control system. The first approach might be possible, but since there can be a lot of environmental conditions depending on a sea site, the training might be difficult. The second approach was proposed by Nguyen et al. (2007). In the scheme with the second approach, multiple adaptive fine-tuning systems with respect to the sea states are trained first. Then, they are integrated by using the switching control system which determines which adaptive fine-tuning system to use in a current state. Since the learning complexity can be distributed over several adaptive fine-tuning systems with the second approach, the second approach seems more feasible than the first one.

Chapter 6 Application of Recent Developments in Deep Learning To ANN-based Automatic Berthing System

6.1. Introduction

When a ship approaches a port or harbor, which is called a "berth operation," slow speeds cause nonlinear characteristics in a ship's motion, and the ship undergoes sudden changes in its rudder angle and engine revolutions per second (RPS). For this reason, unlike ship navigation, where ships sail almost at a constant speed, the ship berthing problem is a challenging one. In order to overcome these difficulties, many intelligent control algorithms have been proposed, and one of them is the artificial neural network (ANN) or simply the neural network (NN). With the rapid development of deep learning (DL), systems based on neural networks have drastically increased and demonstrate extraordinary performance. Therefore, many studies on the application of neural networks to ship berthing systems have been conducted.

Im and Hasegawa (2001) proposed the "Parallel Neural Network," which makes the neural network focus on each task by changing the ordinary neural network architecture. Im and Hasegawa (2002) attempted to use another neural network to identify ship motions so that the neural network system could cope with wind disturbances. Im (2007) proposed an algorithmic method using a neural network to allow ships to berth safely, even when they approached from unusual directions. Bae et al. (2008) compared two neural networks with different input configurations and showed that both input configurations led to a similar performance. Ahmed and Hasegawa (2013) used a virtual window concept and nonlinear programming method to generate teaching data consistently. Im and Nguyen (2018) tried solving the problem in which a neural network trained with a ship berthing dataset at one port can only be used at that particular port but not at others. The researchers adopted a head-up coordinate system instead of a north-up coordinate system, and the head-up coordinate system allowed the neural network trained with a ship berthing dataset at one port to be used at another port as well.

However, although many studies on ANN-based automatic berthing have been conducted to date, they have missed to address several important issues. First, they neither considerd to use recent activation functions nor mentioned about weight initialization methods. Second, they used an inefficient optimizer, although, there are other optimizers with higher performance that have been proposed and verified. Third, there are no consistent method for input data scaling, and some previous studies scaled their input data in inefficient ways. Fourth, no regularization methods have been considered. Fifth, owing to the overfitting problem caused by the lack of a regularization method, poor berthing performance occurred when neural networks were given extrapolated initial positions. Therefore, most of the previous studies showed berthing trajectories with initial positions from training datasets and interpolated or slightly extrapolated initial positions only.

In this study, recent activation functions, weight initialization methods, optimizers, input scaling methods, neural networks with a higher number of hidden layers, and the batch normalization (BN) are applied to the ANN-based automatic berthing. Then, their effectiveness is verified based on the loss functions, berthing performance histories, and berthing trajectories.

6.2. Mathematical Model of Ship Maneuvering

To build a mathematical model of ship maneuvering, a definition of a coordinate system and principal particulars of a ship are first presented in Fig. 95 and

Table 24, respectively. Then, a mathematical model of ship maneuvering and modeling of a propeller and rudder are presented. In Fig. 95, x and y are the actual positions of the ship in meters, and η and ξ are the normalized positions of x, y by the length of the ship.

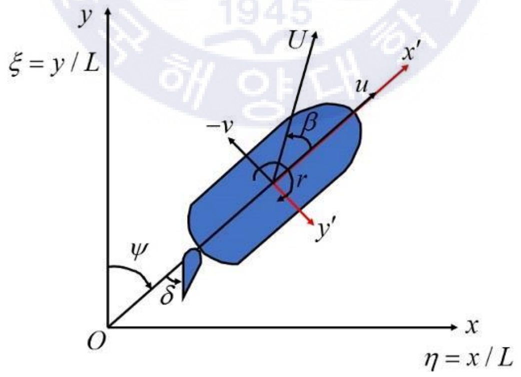


Fig. 95 Coordinate system

Table 24 Principal particulars of a target ship

HULL		
Length overall	L_{OA} [m]	188.0
Length between perpendiculars	L [m]	175.0

Breath	B [m]	25.4
Draft	d [m]	8.50
Block coefficient	C_B	0.559
RUDDER		
Height	H_R	7.70
Area ratio	$A_R/(Ld)$	1/45.8
Aspect ratio	λ	1.827
PROPELLER		
Diameter	D [m]	6.5
Pitch ratio	P/D	1.055
Expanded area ratio		0.730

6.2.1. Mathematical Model for Ship-Maneuvering Problem

In the ship-maneuvering problem, a planar motion includes only surge, sway and yaw motions, which are considered out of six-degree-of-freedom motions. The basic equations of motion for ship maneuvering can be expressed as follows:

$$m(\dot{u} - vr) = F_{surge} \tag{135}$$

$$m(\dot{v} + ur) = F_{sway} \tag{136}$$

$$I_z\dot{r} = M_{yaw} \tag{137}$$

where m is the mass, and u, v, r are the velocities in the surge, sway, yaw directions, respectively. I_z is the moment of inertia with respect to the z-axis, and F_{surge}, F_{sway}, F_{yaw} are the surge and sway forces and yaw moment, respectively. The dot represents the time derivative. The right terms in Eqs. (135)-(137) can be dealt with using the established procedure from (Newman, 2019), and Eqs. (135)-(137) can be expressed as Eqs. (138)-(140).

$$(m + m_x)\dot{u} - (m + m_y)vr = X \tag{138}$$

$$(m + m_y)\dot{v} + (m + m_x)ur = Y \tag{139}$$

$$(I_z + J_z)\dot{r} = N - x_G Y \tag{140}$$

where m_x, m_y, J_z are the added mass in the surge and sway directions and the added moment of inertia, respectively. X, Y are the hydrodynamic forces, and N is the hydrodynamic moment. x_G is the distance from the midship to the center of gravity. The hydrodynamic forces and moment are generally expressed as Eqs. (141)-(143), which were proposed by the Maneuvering Modeling Group (MMG).

$$X = X_H + X_P + X_R + X_W + X_T \tag{141}$$

$$Y = Y_H + Y_P + Y_R + Y_W + Y_T \tag{142}$$

$$N = N_H + N_P + N_R + N_W + N_T \tag{143}$$

where subscripts H, P, R, W, T refer to the hull, propeller, rudder, wind and tug, respectively. However, because this study focuses on the application of recent developments in DL, the external forces such as the wind and tug are not considered.

A ship's maneuvering displacements and velocities can be calculated numerically by integrating Eqs. (144)-(146) using the Runge–Kutta 4th-order method.

$$\dot{u} = \frac{X + (m + m_y)vr}{(m + m_x)} \tag{144}$$

$$\dot{v} = \frac{Y - (m + m_x)ur}{(m + m_y)} \tag{145}$$

$$\dot{r} = \frac{N - x_G Y}{I_z + J_z} \tag{146}$$

6.2.2. Modeling of Propeller and Rudder

Owing to the kinematics of the mechanical system of the rudder and engine, time delays between commands of operation and the resultant mechanical response may occur. The equations from (Sohn, 1992) to consider this physical time-delay effect are used here. First, the equation to consider the time-delay effect for the propeller RPS is Eq. (147).

$$\dot{n} = (n^* - n)/T_n \tag{147}$$

where n is the RPS, $\dot{n}$ is a derivative of RPS with respect to time, and n^* is the target RPS. T_n is a time-delaying constant for the RPS, and this variable is set to 15 s in this study. Next, equations to consider the time-delay effect for a rudder are Eqs. (148)-(149).

$$\dot{\delta} = (\delta^* - \delta)/T_E;\ \text{if}\ |\delta^* - \delta| \le T_E\left|\dot{\delta}_{\max}\right| \tag{148}$$

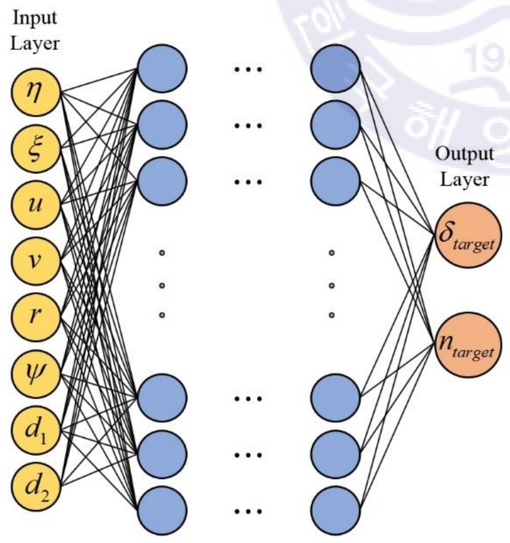
$$\dot{\delta} = \text{sign}(\delta^* - \delta)|\dot{\delta}_{\max}|;\ \text{if}\ |\delta^* - \delta| > T_E|\dot{\delta}_{\max}| \tag{149}$$

where δ^* is the target rudder angle, T_E is the time-delaying constant for the rudder, and $\dot{\delta}_{\max}$ is the maximum angular velocity of the rudder. In this study, T_E is set to 2.5 s, and $\dot{\delta}_{\max}$ is set to 3.0°/s.

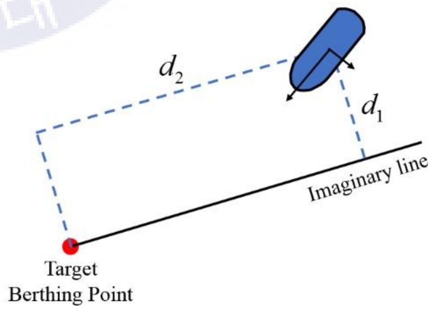
6.3. Artificial Neural Network and Important Factors in Training the Network

6.3.1. Artificial Neural Network

The basic architecture of the neural network used in this study is illustrated in Fig. 96, η is $x\,/L$, and ξ is $y\,/L$, in which $x,\ y$ are the relative positions of the ship from the target berthing point, and L is the LBP of the ship. $u,\ v,\ r$ are the velocities of the ship in the surge, sway, and yaw directions, respectively, and ψ is the heading angle of the ship. In the output layer, $\delta_{\text{target}},\ n_{\text{target}}$ are the target rudder angle and target RPS, respectively. This input layer configuration has been used in several studies such as (Im and Hasegawa, 2001; Ahmed and Hasegawa, 2013). The number of hidden layers and the size of the neural network will be addressed in Chapter 6.3.4.



(a)



(b)

Fig. 96 Basic architecture of neural network, and concept of imaginary line

The imaginary line shown in Fig. 96 is used as a guideline during berthing. It is known that as the imaginary line provides the additional input variables of d_1 and d_2, which can improve the berthing performance by the neural network. The angle between the x-axis and the imaginary line is set to 20° in this study.

For the teaching data preparation, the patterns of the berthing trajectories, rudder angle, and RPS control from (Bae et al., 2008; Nguyen, Do and Im, 2018) are referred. The teaching dataset used in this study is shown in Fig. 97, where the red crosses are the initial positions in the teaching data, the target berthing facility is drawn in blue, and the orange straight line is the imaginary line.

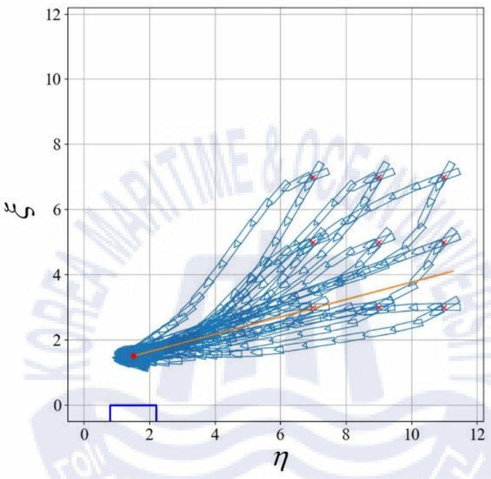


Fig. 97 Teaching dataset

The end of the berthing operation is defined as a status meeting and the following three conditions according to Ahmed and Hasegawa (2012). First, the aim is to berth at a distance of around the length of 1.5 times farer than where the ship is from the berthing facility. Second, the relative heading angle to the berthing facility should be smaller than 30°. Third, the velocity of the ship should be slower than 0.2 m/s.

The overall flowchart for the automatic ship berthing with a neural network is shown in Fig. 98, the left side of the dotted line is the preparation step for the neural network model, and the right side is a processing loop with the trained neural network. In this study, building and training the neural networks are carried out with the open-source software library Tensorflow.

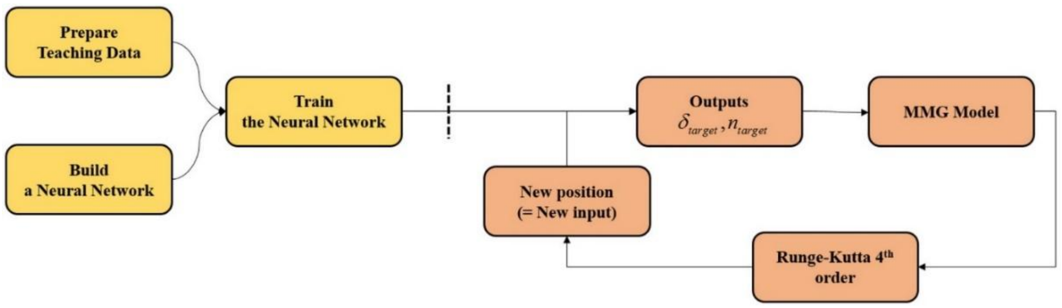


Fig. 98 Flowchart for ANN-based ship berthing system

6.3.2. Optimizer

In previous studies, such as (Im and Hasegawa, 2001; Im and Hasegawa, 2002; Ahmed and Hasegawa, 2013; Im and Nguyen, 2018) adopted the Levenberg–Marquardt (Levenberg, 1944) as an optimizer. However, according to (Bazzi, Ismail and Zohdy, 2018), that analyzed the training performance of several popular optimizers including the Levenberg–Marquardt, the RMSProp turned out to be faster and computationally more efficient, and required fewer hyperparameters than Levenberg–Marquardt. Furthermore, Kingma and Ba (2015) showed that the Adam optimizer outperforms the RMSprop. Therefore, the Adam is adopted as an optimizer in this study. The learning rate of the Adam is set to 1e-3, which results in robust training performance in general. For the loss function, the MSE is used.

6.3.3. Input Data Scaling

Table 25 List of input data-scaling methods

	Scaling method	
Scaling #1	0 to 1 Min-Max Scaling	Scales inputs to range of 0 to 1
Scaling #2	-1 to 1 Min-Max Scaling	Scales inputs to range of -1 to 1
Scaling #3	Standard Scaling	Standardizes inputs by removing mean and scaling to unit variance

Because the input data of a ship berthing dataset for neural network contains variables of different units, the weights are likely to be updated in an unstable way during training when input data scaling is not applied because of the different scales of each input variable. However, among the previous studies on ANN-based automatic berthing, some did not employ a scaling method for their input data, and some applied a scaling method on some input variables only. A few of the studies scaled the input data to between 0 and 1. The commonly used input data-scaling methods are presented in **Table 25**. Equations for the Scaling #1 and the Scaling #2 are shown in Eq. (150) and Eq. (151), respectively.

$$x' = (x - \min(X))/(\max(X) - \min(X)) \tag{150}$$

$$x' = (x - 2\min(X))/(\max(X) - \min(X)) \tag{151}$$

where x is the sampled data, x' is the scaled sampled data, and X is a set of data.

6.3.4. Number of Hidden Layers

The number of hidden layers has a close relationship with the neural network model's system capacity. It is commonly known that if a neural network model has a large number of hidden layers, it has a large capacity to understand the complexity of a given training dataset and to have a complex system. However, simply using many hidden layers does not guarantee the neural network's good performance. Numbers of hidden layers of 5, 10, 20, 30, and 40 are considered and investigated in Chapter 6.4. The hidden layer size is also an important factor in the neural network architecture, but in order to build a neural network model for a complex system, the number of hidden layers is more influential. Generally, the hidden layer size is set to 2^n. n is 3 for relatively simple systems, 6 for slightly complex systems, and 8–9 for complex systems. In this study, the hidden layer size is set to 2^6.

6.3.5. Overfitting Prevention

Overfitting may occur when a neural network is trained with a given training dataset over many epochs because the weights become too fit to the given dataset only. Then, the performance of the neural network fails with new input data. In an automatic berthing using a neural network, if the overfitting occurs, the neural network will show poor berthing performance when a new initial position is given. The poor performance with the overfitting issue occurs more clearly when new input data is extrapolated rather than interpolated by the neural network. In previous studies on ANN-based automatic berthing, no regularization methods to prevent overfitting were considered. Some studies such as (Bae et al., 2008) showed that poor berthing performance occurs owing to an extrapolated initial position. In order to deal with the poor performance caused by the extrapolation, the BN is adopted in this study.

6.4. Application of Recent Developments in Deep Learning to Automatic Berthing

In this subchapter, the effects of the activation function, weight initialization method, input data-scaling method, number of hidden layers, and the BN are investigated in terms of the neural network's training performance and berthing performance.

As to the activation functions and weight initialization methods, nine different neural network models with nine combinations from three activation functions of the sigmoid, tangent hyperbolic (tanh), and ReLU; and three weight initialization methods of the normal distribution, Xavier, and He are built. In order to show the training progress over epochs, a loss function history graph is presented in Fig. 99. Details of the training models are listed in Table 26 for better reproducibility.

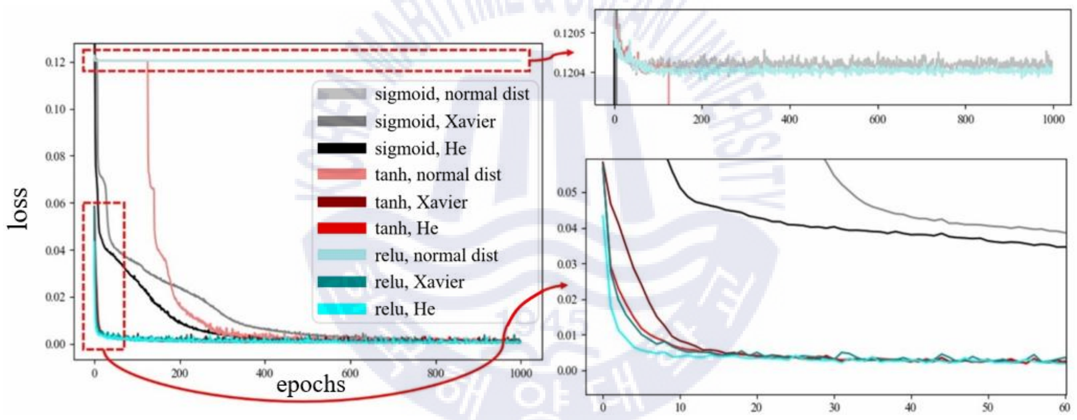


Fig. 99 Loss function histories with respect to activation and weight initialization methods

Table 26 Details of training models with respect to activation and weight initialization methods

Hidden layer size	Number of hidden layers	Optimizer, learning rate	Number of epochs	Batch size	Activation function	Weight initialization	Input data-scaling method
2^6	10	Adam, 1e-3	1000	2^7	sigmoid, tanh, ReLU	normal distribution, Xavier, He	-1 to 1 Min-Max Scaling

The number of hidden layers is set to 10 to make use of the DNN. The effect of the number of hidden layers is investigated later in this subchapter. The batch size is set to 2^7 because this is usually set to 2^n, where n is generally 5 to the minimum and 10 to the maximum.

In Fig. 99, the models with weight initialization by normal distribution led to a slow training speed and poor convergence. The models using the sigmoid function show a slow training speed as well. It is shown that the traditional activation function and weight initialization method are inefficient in terms of training the DNN for automatic ship berthing. The best combination of the activation function and weight initialization method is ReLU-He.

For the input data-scaling methods, the effect of each input data-scaling method is analyzed with a loss function history. The loss function history is shown in Fig. 100, and details of the training models with respect to the input data-scaling method are listed in Table 27.

Table 27 Details of training models with respect to activation and weight initialization methods

Hidden layer size	Number of hidden layers	Optimizer, learning rate	Number of epochs	Batch size	Activation function	Weight initialization	Input data-scaling method
2^6	10	Adam, 1e-3	1000	2^7	ReLU	He	No Scaling, $x\,y$-Only Scaling, 0 to 1 Min-Max Scaling, -1 to 1 Min-Max Scaling, Standard Scaling

In Table 27, the training model with No Scaling means that the input data of $\{x,\ y,\ u,\ v,\ r,\ \psi,\ d_1,\ d_2\}$, where x, y are the actual $x,\ y$ coordinates of the ship from the berthing facility, are used for the neural network. The training model with the $x\,y$-Only Scaling means that the input data of $\{\eta,\ \xi,\ u,\ v,\ r,\ \psi,\ d_1,\ d_2\}$ are used for the model.

In Fig. 100, the significance of application of the input data-scaling method is clearly seen by its training speed and convergence. Standard Scaling shows the fastest training speed and best convergence with the least oscillation, while No Scaling resulted in poor training performance.

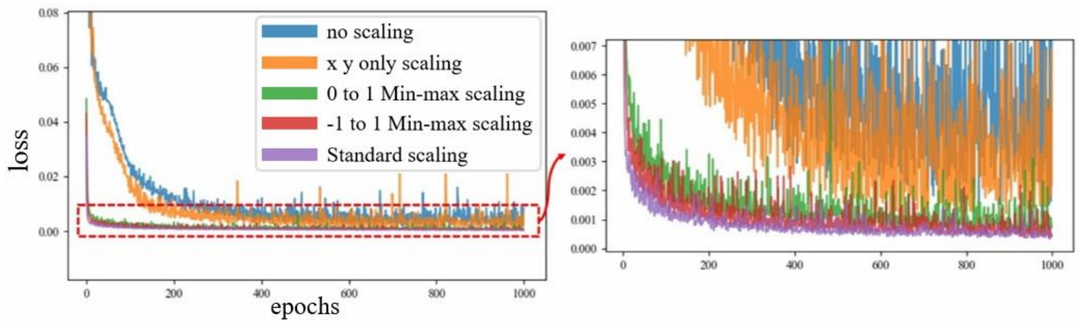


Fig. 100 Loss function histories with respect to input data-scaling methods

Next, the effect of the number of hidden layers is analyzed based on the loss function history and berthing performance history with initial positions from the training dataset. The berthing performance history is obtained as in **Table 28**.

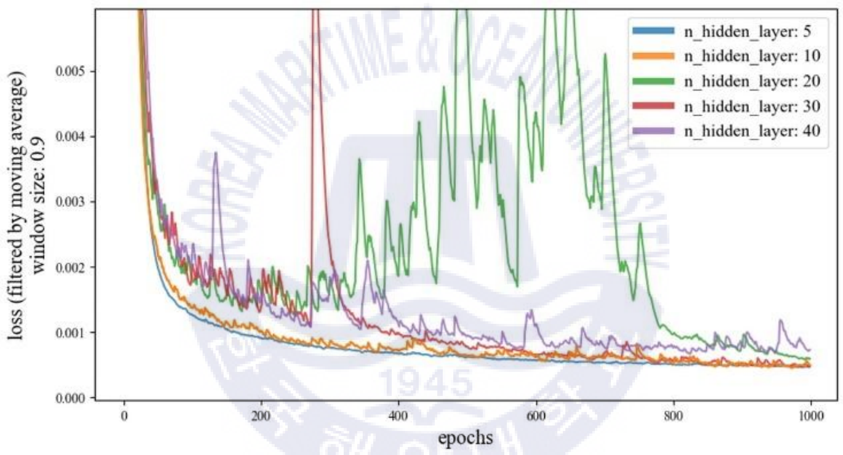
Table 28 Procedure for obtaining berthing performance history

```
Set a period for a berthing performance test ← $Period_{\text{performance test}}$
Set initial positions of the ship for a berthing performance test ← $P_{init}$
Initialize an array to store berthing performance history ← $Arr_{perf}$
for epoch = 1, 2, ... do
    Train the neural network with a given training dataset
        if epoch % $Period_{\text{performance test}}$ = 0 then
        Test the neural network with $P_{init}$
        Measure the distance between the target berthing point and the arrival
        position of the ship
        Store the distance value in $Arr_{perf}$
    end if
end for
return $Arr_{perf}$
```

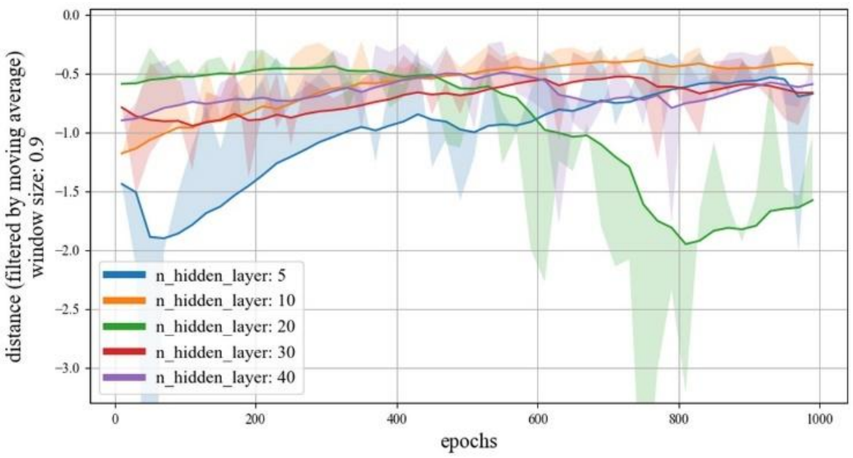
The loss function history and berthing performance history with initial positions of the ship from the training dataset with respect to the number of hidden layers are shown in Fig. 101. Details of the training models are listed in **Table 29**. In Fig. 101 (a), n_hidden_layer of 5 and 10 result in the most stable training, whereas the others show somewhat unstable training with fluctuations. In Fig. 101 (b), the *y*-axis of the distance represents the distance gap between the target berthing point and arrival position of the ship, as shown in **Table 28**. Thus, when the distance is zero, this indicates the highest value and thus the highest performance. The edges of the shaded areas in Fig. 101 (b) represent the actual data. It is shown that the training model with the n_hidden_layer of 10 shows the highest performance, and an unnecessarily complex neural network model with many hidden layers decreases the performance.

Table 29 Details of training models with respect to number of hidden layers

Hidden layer size	Number of hidden layers	Optimizer, learning rate	Number of epochs	Batch size	Activation function	Weight initialization	Input data-scaling method
2^6	5, 10, 20, 30, 40	Adam, 1.0e-3	1000	2^7	ReLU	He	Standard Scaling



(a) Loss function histories



(b) Berthing performance histories with initial positions from training dataset

Fig. 101 Loss function and berthing performance histories with respect to number of hidden layers

Finally, to verify the model's universal berthing performance, the effect of using the BN to prevent overfitting and to improve the neural network's performance is analyzed by the loss function history and berthing performance history with initial positions from the training dataset and a set of new initial positions that are not included in the training dataset. The set of new initial positions is shown in Fig. 102, where the red crosses are the initial positions in the teaching data and the blue ships represent the new initial positions.

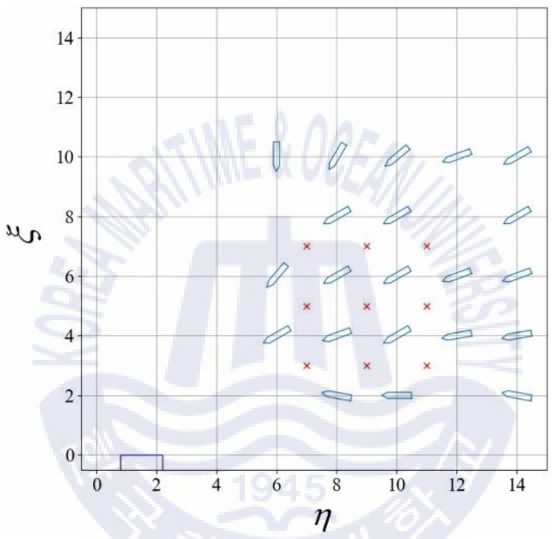
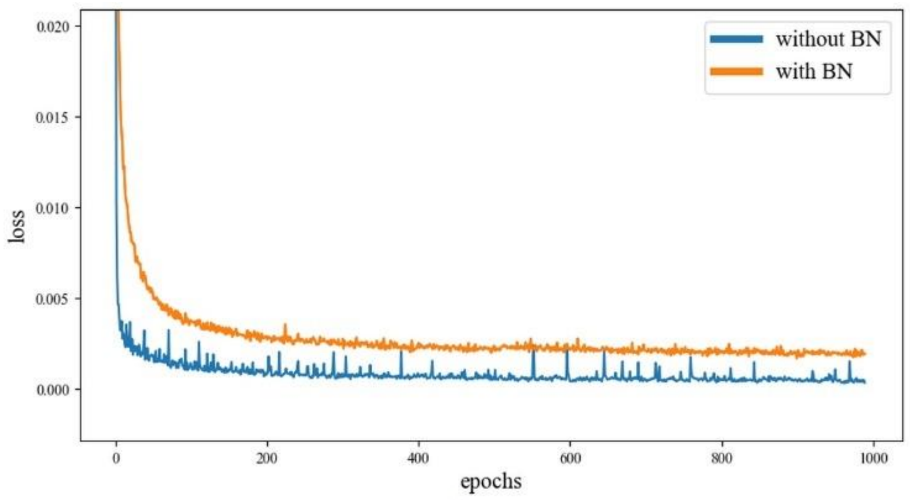


Fig. 102 Set of new initial positions for berthing performance history to verify universal berthing performance

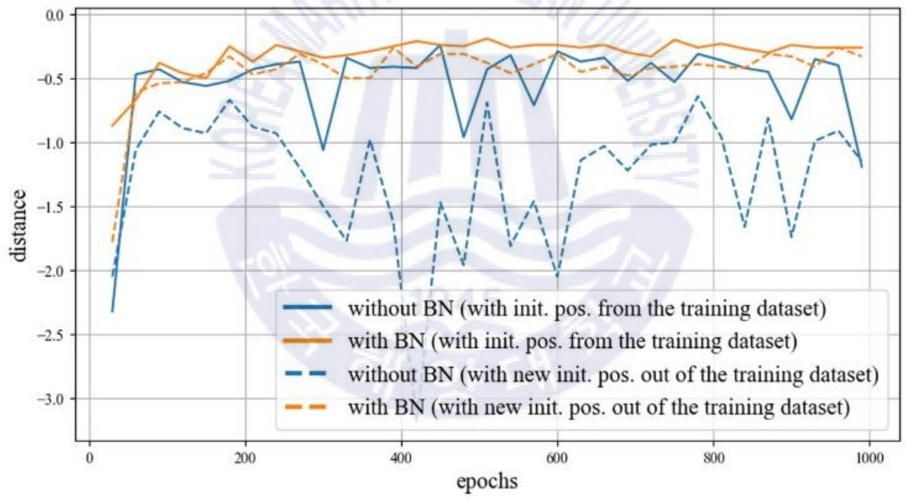
The loss function and berthing performance histories, and details of the training models with respect to the use of the BN, are shown in Fig. 103 and **Table 30**, respectively.

Table 30 Details of training models when BN is used

hidden layer size	number of hidden layers	optimizer, learning rate	number of epochs	batch size	activation function	weight initialization	input data-scaling method	use of BN
2^6	10	Adam, 1e-3	1000	2^7	ReLU	He	Standard Scaling	True, False



(a) Loss function histories



(b) Berthing performance histories

Fig. 103 Loss function and berthing performance histories when BN is used

In Fig. 103 (a), it can be seen that the BN prevents overfitting and stabilizes the training process. The performances in Fig. 103 (b) show that the model with the BN outperforms the model without the BN not only for the initial positions from the training dataset but also for the new initial positions that are not included in the training dataset.

Table 31 Trained model selection algorithm for a neural network for automatic ship berthing

```
Initialize an array to store harmonic mean values ← $Arr_{m}$
Set the berthing performance history data with the initial positions from the training dataset ← $X$
Set the berthing performance history data with the new initial positions out of the training dataset ← $Y$
**for** each $x \in X$ and $y \in Y$ **do**
    Concatenate $X$ and $Y$ ← $XY$
    $x' = \frac{x - \min(XY)}{\max(XY) - \min(XY)}$
    $y' = \frac{y - \min(XY)}{\max(XY) - \min(XY)}$
    $hm = (2x'y')/(x' + y')$    // harmonic mean
    Store $hm$ in $Arr_{hm}$
**end for**
**return** argmax( $Arr_{hm}$)
```

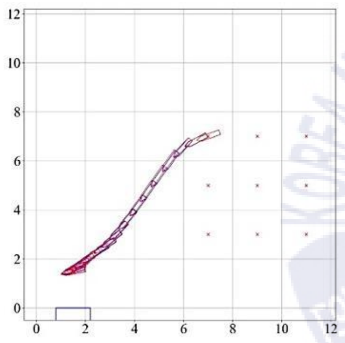
Although the trained model with the BN shows good and stable performance over epochs, it is ideal to select the best-trained model of them all. In order to select the best-trained model, a trained model selection algorithm for neural network models for automatic berthing using the harmonic mean is proposed. The trained model selection algorithm is shown in Table 31.

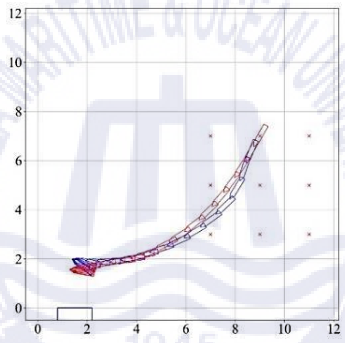
We adopted the harmonic mean to penalize cases where the gap between the berthing performances with the initial positions from and out of the training dataset is high, meaning that the model is overfit to the given training dataset and has a poor universal berthing performance. It has been shown that the use of the recent activation functions, weight initialization method, and input data-scaling methods resulted in faster training speeds and better convergence of the loss function. In addition, the BN improved the actual berthing performance for universal use and stabilized the training process. With the trained model selection algorithm, the two best-trained models were selected for the final simulation in Chapter 6.5. One is from the trained models with the BN, and the other is from the trained models without the BN. The significance of application of the BN is shown and discussed in the next subchapter.

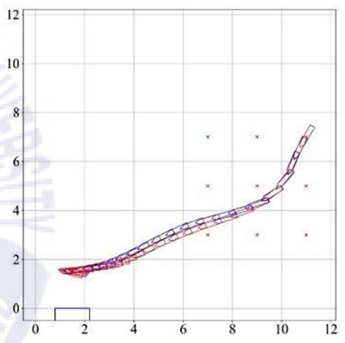
6.5. Simulation and Results Discussion

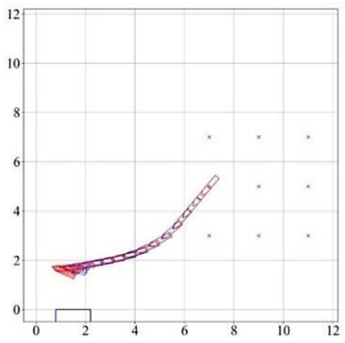
In this subchapter, the berthing trajectories by two different trained models with and without the BN are presented to show how the BN improves the neural network for ship berthing and solves the extrapolation problem. First, berthing trajectories with the initial positions from the training dataset are presented. And then, the berthing trajectories with interpolated and extrapolated initial positions are presented next, respectively. In the berthing simulations in this subchapter, the initial u is set to 1.5, and the initial n is set to 0.5 for all cases.

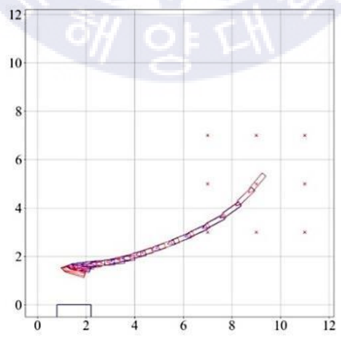
To validate if the two trained neural network models perform efficiently when initial positions from a training dataset are given, berthing trajectories with initial positions from a training dataset are presented in Fig. 104. As expected from the berthing performance history in Fig. 103 (b), similar berthing performance and trajectories are shown with the initial positions from the training dataset as shown in Fig. 104 (a)-(i), meaning that both trained models are trained well for the given training dataset.

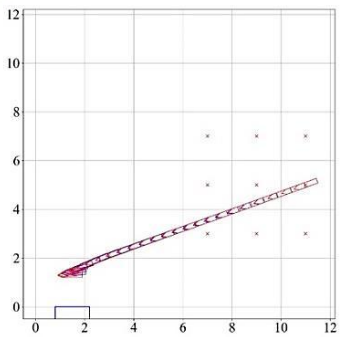

(a) (η, ξ, ψ): (7, 7, 250°)

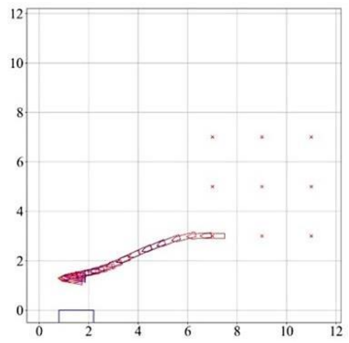

(b) (η, ξ, ψ): (9, 7, 210°)

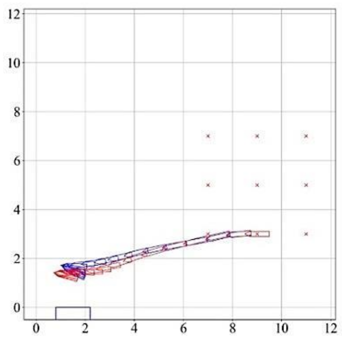

(c) (η, ξ, ψ): (11, 7, 250°)


(d) (η, ξ, ψ): (7, 5, 220°)


(e) (η, ξ, ψ): (9, 5, 220°)


(f) (η, ξ, ψ): (11, 5, 250°)


(g) (η, ξ, ψ): (7, 3, 270°)


(h) (η, ξ, ψ): (9, 3, 270°)

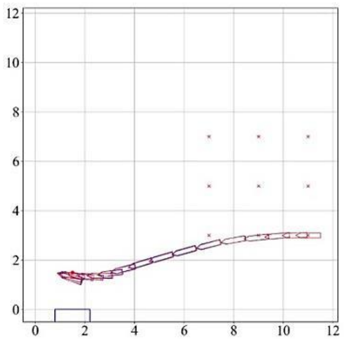
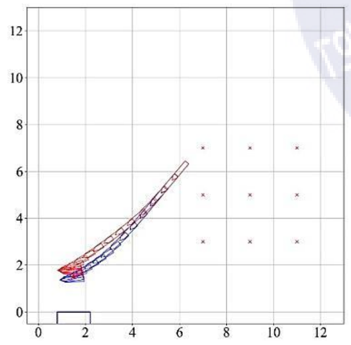
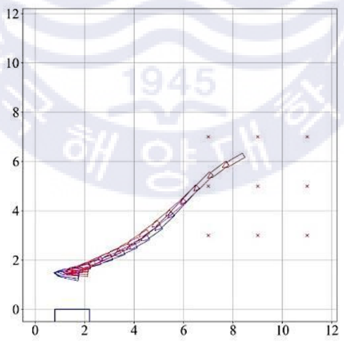
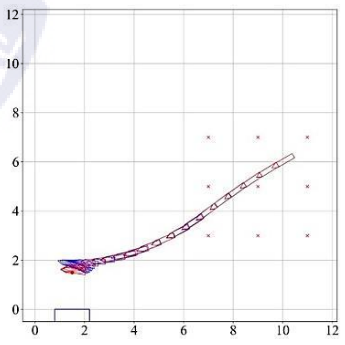

(i) (η, ξ, ψ): (11, 3, 270°)

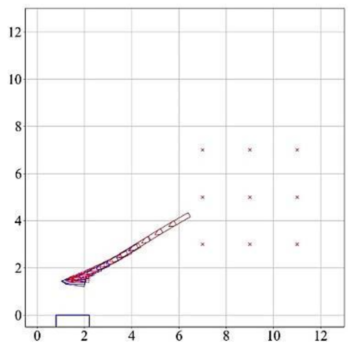
Fig. 104 Berthing trajectories with initial positions from training dataset (red: with BN, blue: without BN)

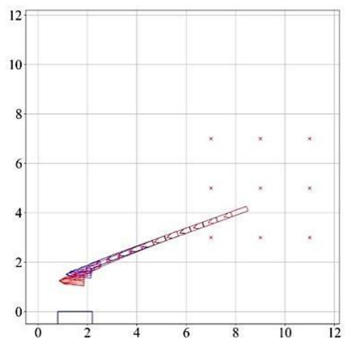
To validate if the two trained neural network models perform efficiently when interpolated initial positions are given, berthing trajectories with the interpolated initial positions are presented in Fig. 105. It can be observed that when the interpolated initial positions are given, both trained models showed successful berthing performances with similar berthing trajectories as shown in Fig. 105 (a)-(f).


(a) (η, ξ, ψ): (6, 6, 220°)


(b) (η, ξ, ψ): (8, 6, 240°)


(c) (η, ξ, ψ): (10, 6, 240°)


(d) (η, ξ, ψ): (6, 4, 240°)


(e) (η, ξ, ψ): (8, 4, 250°)

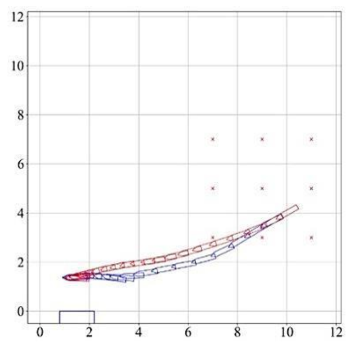
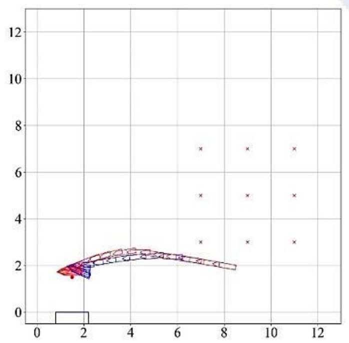
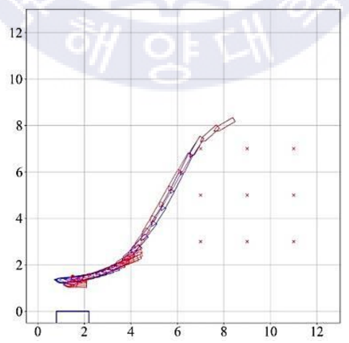

(f) (η, ξ, ψ): (10, 4, 240°)

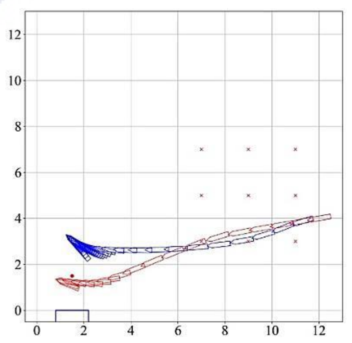
Fig. 105 Berthing trajectories with interpolated initial positions (red: with BN, blue: without BN)

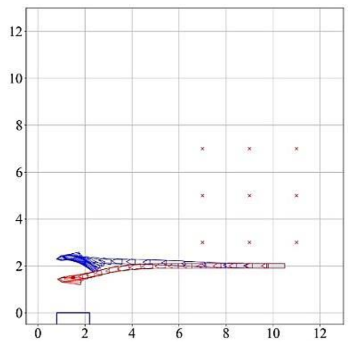
In the following simulations, the trained models are given three types of extrapolated initial positions. The types of extrapolated initial positions differ by the magnitude of extrapolation.

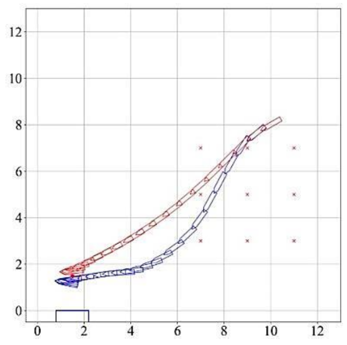
Berthing trajectories with the first type of extrapolated initial position are shown in Fig. 106. It can be observed that although the trained model without the BN shows good berthing performance in a few cases such as Fig. 106 (a) and Fig. 106 (b), it also shows poor performance as shown in Fig. 106 (c)-(d), while the model with the BN performs successfully in all the cases. In addition, it can be seen that the two trained models control quite differently to reach the target berthing point, as seen in Fig. 106 (e)-(f).


(a) (η, ξ, ψ): (8, 2, 280°)


(b) (η, ξ, ψ): (8, 8, 240°)


(c) (η, ξ, ψ): (12, 4, 260°)


(d) (η, ξ, ψ): (10, 2, 270°)


(e) (η, ξ, ψ): (10, 8, 240°)

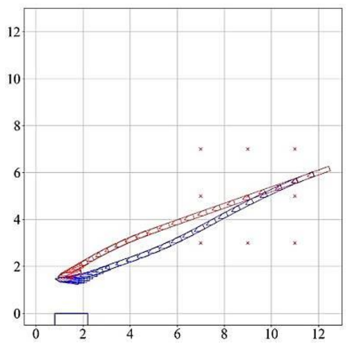
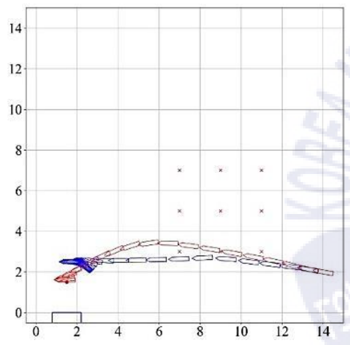
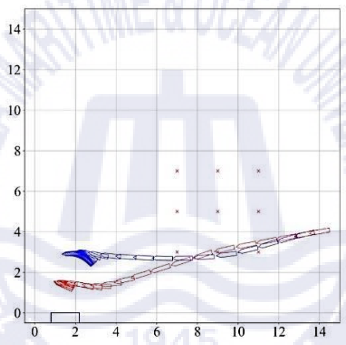
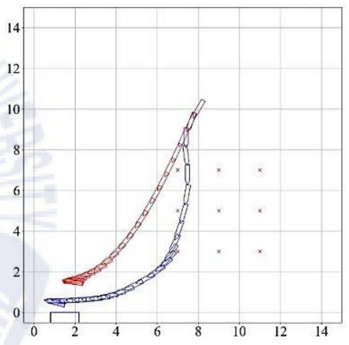

(f) (η, ξ, ψ): (12, 6, 250°)

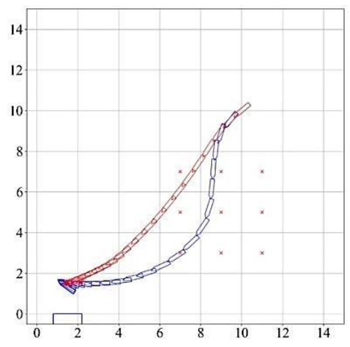
Fig. 106 Berthing trajectories with extrapolated initial positions, Type 1

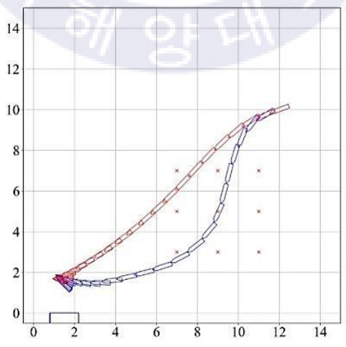
(red: with BN, blue: without BN)


(a) (η, ξ, ψ): (14, 2, 280°)


(b) (η, ξ, ψ): (14, 4, 260°)


(c) (η, ξ, ψ): (8, 10, 210°)


(d) (η, ξ, ψ): (10, 10, 230°)


(e) (η, ξ, ψ): (12, 10, 250°)

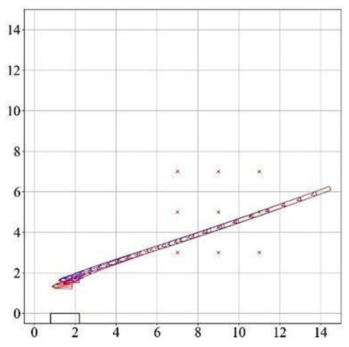

(f) (η, ξ, ψ): (14, 6, 250°)

Fig. 107 Berthing trajectories with extrapolated initial positions, Type 2

(red: with BN, blue: without BN)

Next, berthing trajectories with the second type of extrapolated initial position are shown in Fig.

107. As the more extrapolated initial positions are given in the cases in Fig. 107, the trained model without the BN suffers from the extrapolation issue even more as shown in Fig. 107 (a)-(e). The only case where the trained model without the BN performs well is when it berths in a straight line as shown in Fig. 107 (f).

Finally, berthing trajectories with the third type of extrapolated initial position are shown in Fig. 108, in which the extrapolation of initial positions is the greatest in order to clearly see the effect of the BN. In Fig. 108. The model without the BN performs poorly overall owing to the extrapolation problem. The effectiveness of application of the BN is clear, as the BN improves the neural network model for more general use.

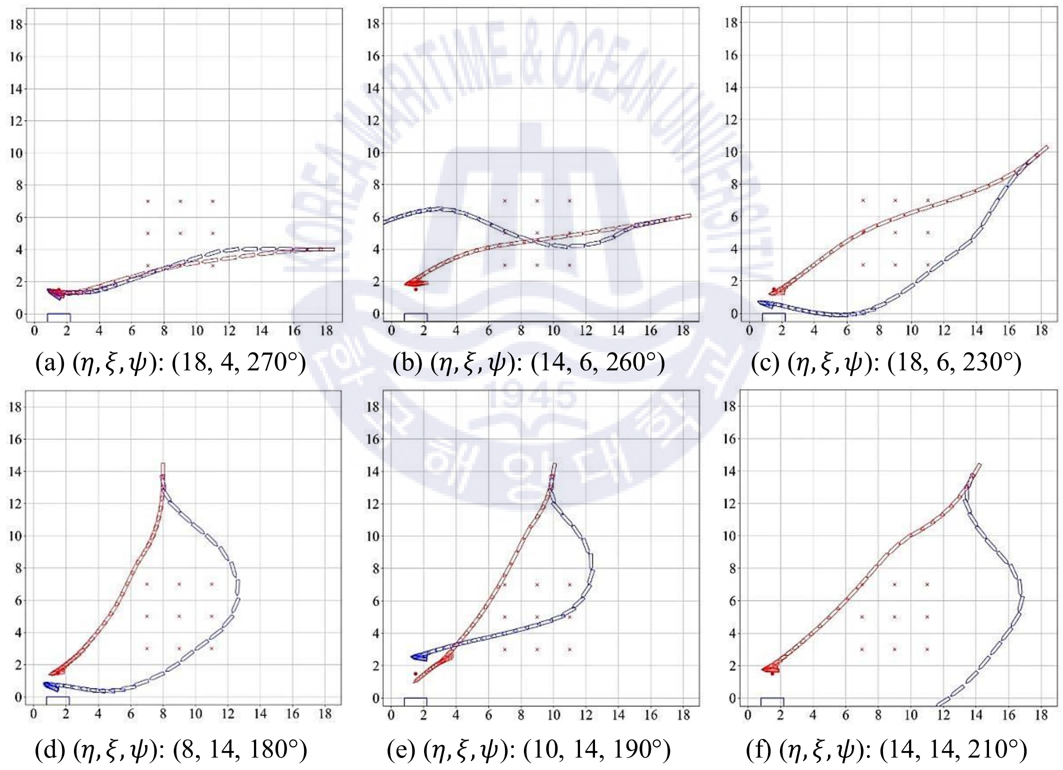

(a) (η, ξ, ψ): (18, 4, 270°) (b) (η, ξ, ψ): (14, 6, 260°) (c) (η, ξ, ψ): (18, 6, 230°)
(d) (η, ξ, ψ): (8, 14, 180°) (e) (η, ξ, ψ): (10, 14, 190°) (f) (η, ξ, ψ): (14, 14, 210°)

Fig. 108 Berthing trajectories with extrapolated initial positions, Type 3
(red: with BN, blue: without BN)

Chapter 7 Conclusion

In this paper, the following four studies were conducted: 1) machine learning-based mooring line tension prediction system, 2) feed-forward system for DPS using predicted drifted ship position based on deep learning and replay buffer, 3) reinforcement learning-based adaptive PID controller for DPS, 4) application of recent developments in deep learning to ANN-based automatic berthing system.

7.1. Machine Learning-Based Mooring Line Tension Prediction System

The machine learning-based mooring line tension prediction system mainly consists of the proposed K-means-based sea state selection method and the proposed hybrid neural network architecture.

The proposed K-means-based sea state selection method provides a standardized way of the sea state selection to obtain the training dataset by utilizing the K-means algorithm. The proposed sea state selection method eliminates the manual sea state selection which was a limitation in the previous studies, and it can reduce a number of sea states for the dynamic simulations to obtain the training dataset by efficiently selecting the representative sea states, which saves a large amount of computational cost. In the simulation, it was shown that a number of sea states could be reduced by 3.5 times by using the proposed K-means-based sea state selection method while maintaining the tension prediction performance on the test sea states.

The proposed hybrid neural network architecture provides the excellent tension prediction performance across the entire sea states. This high prediction performance was achieved by the use of BN and the fusion of the overall Hs-focused NN model and the low Hs-focused NN model with the proposed soft-binary attention module in which the soft-binary attention module pays the appropriate amount of attention to each NN model given its input at every timestep. In the simulation, it was shown that the use of BN clearly improved the tension prediction performance and the soft-binary attention module could learn to efficiently output the attention values, which results in the excellent prediction performance.

Since the proposed machine learning-based mooring line tension prediction system shows the excellent prediction performance on various sea states in one main environmental direction, a further study can be conducted by considering multiple environmental directions for a broader use such as a mooring line tension monitoring system that can replace physical sensors for the mooring line tension as a digital twin system. When multiple environmental directions are considered, a number of directional sea states increases drastically, which significantly increases the computational cost for the dynamic simulation. However, it is strongly supposed that this problem can be resolved by the proposed K-means-based sea state selection method while the proposed hybrid neural network architecture may provide the excellent prediction performance for various directional sea states.

7.2. Motion Predictive Control for DPS Using Predicted Drifted Ship Position Based on Deep Learning and Replay Buffer

The conventional PID feed-back system for the DPS is integrated with the proposed motion predictive control to improve the station-keeping performance by reducing magnitudes of the ship drifting motions. The proposed motion predictive control is designed based on the DL layer, dropout, online machine learning system, and the replay buffer.

The DL layers and dropout form the DNN, and the DNN is used to output the predicted drifted ship motion in the future. By using the DNN, the prediction accuracy for the predicted drifted ship motions is ensured, which guarantees the station-keeping performance improvement of the proposed motion predictive control.

The online machine learning system provides the proposed motion predictive control with the adaptability to a new/varying sea environment, therefore, it can eliminate necessity of training the DNN for all the possible environmental conditions before deployment, which saves a lot of computational time and large load on the DNN for having to generalize all the given data from all the environmental conditions at once.

The replay buffer allows the DNN to be trained with a variety of data from various timesteps to make the time-independent learning possible which helps the DNN have a better understanding of ship dynamics. In the comparative simulation with respect to the replay buffer, it was shown that the use of the replay buffer resulted in the better the station-keeping performance throughout the entire simulation, which makes the replay buffer one of the crucial components in the proposed motion predictive control.

The real-time normalization method ensures robustness of the neural network training process by adaptively normalizing training data. The main difference between the ordinary normalization method and the proposed real-time normalization method is that the ordinary normalization method calculates normalizing factors before the training and uses the factors for the neural network training as constants.

However, the proposed real-time normalization method constantly updates the normalizing factors with the incoming data to the replay buffer, which makes the normalization be able to normalize training data properly and adaptively with new data distribution from varying environmental conditions.

Finally, the PID feed-back system with the wind feed-forward system and the PID feed-back system with the proposed motion predictive control were compared in the simulations. The simulation results showed that the proposed motion predictive control resulted in the better station-keeping performance than the wind feed-forward system with the same amount of thrust consumption.

7.3. Reinforcement Learning-Based Adaptive PID Controller for DPS

In this paper, an adaptive PID controller for the DPS is developed with the proposed adaptive fine-tuning system for the PID gains. The main goal of the proposed adaptive fine-tuning system is to learn an efficient adaptive gain-tuning strategy to increase the station-keeping performance without deterioration in the control efficiency. To achieve the goal, one of the deep reinforcement learning algorithms, deep deterministic policy gradient (DDPG), was adopted. The main characteristics of the proposed adaptive fine-tuning system is as follows:

First, a target of the application is a 285m FPSO ship and the environment is the open sea where the sea state can become severe. The ships have a much slower response to a control signal unlike small robots, and the environmental loads a ship undergoes when the sea state is severe are far greater than wind disturbance that small robots experience.

Second, the input is previous ship motions, and the output is the adaptive PID gains and the update-gate for the integral of the errors. For the input, previous ship motions are used instead of a ship motion at a current timestep to help the neural networks understand the ship dynamics better by considering the memory effect. The adaptive PID gains allow the PID controller to cope with the varying environmental loads better, and the update-gate for the integral of the errors allows to prevent the unconditional accumulation for the integral of the errors, which leads to the smaller rebound ship motions.

Third, the proposed adaptive fine-tuning system is likely to learn an adaptive-gain tuning strategy that simply increases the gains to increase the station-keeping performance if there is no restriction in the available adaptive gain range. However, this strategy is inefficient because it comes with the cost of higher thrust consumption and aggressive thrust control. To encourage the proposed system to learn the adaptive-gain tuning strategy that results in the better station-keeping performance without deterioration in the control efficiency, its available adaptive gain range is set to [0, k_{base}] where k_{base} refers to the base gain pre-tuned by a PID gain-tuning method.

In the simulation results, it was shown that the proposed adaptive fine-tuning system could learn the efficient adaptive gain-tuning strategy which results in the better station-keeping performance without deterioration in the control efficiency. In the discussion subchapter, the advantages and limitations of the proposed system are discussed, and the room for further study is also discussed.

7.4. Application of Recent Developments in Deep Learning to ANN-Based Automatic Berthing System

In the study of the application of recent developments in deep learning to ANN-based automatic berthing system, it was shown that the faster training speed and better training convergence could be achieved with the application of recent activation functions, weight initialization methods, input data-scaling methods, and a higher number of hidden layers. In order to observe the progress of the berthing performance over epochs and select the best-trained model, the algorithm for obtaining the berthing performance history and the model selection algorithm were proposed. Lastly, the use of the BN could stabilize the training process and solve the extrapolation problem by preventing overfitting. A neural network model with the BN was able to perform successfully, not only with interpolated and slightly extrapolated initial positions but also with greatly extrapolated initial positions. This could make the neural network models more universal for automatic berthing.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Seung-Jae Lee for always guiding me to the right direction for my researches at every step of the way with his insightful advice and kindness, and for being my role model as a researcher and a person. I also would like to thank Professor Seong-Uk Lee for conveying his knowledge to me with his lecture and advice and for being always welcoming for any question.

A special thanks to my wife Senida Kakeš for her unchanged support towards my decisions and for giving me a strength everyday with her love and smile. Many thanks for my parents, my father Cheol-Ho Lee, my mother Sook-Ja Kim. Lastly, my old friend and colleague Byeong-Cheon Kim should be thanked for sharing his inspiration and passion towards the further academic career as a researcher.

References

Van 't Veer, R. and Gachet, M. (2011) 'Dynamic Positioning - Early design, capability and offsets, a novel approach', in *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering - OMAE*, pp. 755–764. doi: 10.1115/OMAE2011-49354.

Aalbers, A. *et al.* (2004) 'Wave feed forward DP and the effect on shuttle tanker operation', in *Dynamic Positioning Conference*. Available at: http://dynamic-positioning.com/proceedings/dp2004/abstract_design_aalbers.pdf (Accessed: 19 December 2019).

Ahmed, Y. A. and Hasegawa, K. (2012) 'Automatic ship berthing using artificial neural network based on virtual window concept in wind condition', in *Proceedings of 13th IFAC Symposium on Control in Transportation Systems*, pp. 286–291. doi: 10.3182/20120912-3-BG-2031.00059.

Ahmed, Y. A. and Hasegawa, K. (2013) 'Automatic ship berthing using artificial neural network trained by consistent teaching data using nonlinear programming method', *Engineering Applications of Artificial Intelligence*, 26(10), pp. 2287–2304. doi: 10.1016/j.engappai.2013.08.009.

Bae, C.-H. *et al.* (2008) 'A Study of the Automatic Berthing System of a Ship Using Artificial Neural Network', *Journal of Korean navigation and port research*, 32(8), pp. 589–596. doi: 10.5394/kinpr.2008.32.8.589.

Bazzi, T., Ismail, R. and Zohdy, M. (2018) 'Comparative Performance of Several Recent Supervised Learning Algorithms', *International Journal of Computer and Information Technology*, 7(2). Available at: www.ijcit.com49 (Accessed: 22 April 2020).

Bhatt, A. *et al.* (2019) 'CrossNorm: Normalization for off-policy TD reinforcement learning', in *arXiv preprint arXiv:1902.05605*. Available at: http://arxiv.org/abs/1902.05605 (Accessed: 11 December 2019).

Carlucho, I., De Paula, M. and Acosta, G. G. (2019) 'Double Q-PID algorithm for mobile robot control', *Expert Systems with Applications*, 137, pp. 292–307. doi: 10.1016/j.eswa.2019.06.066.

Carlucho, I., De Paula, M. and Acosta, G. G. (2020) 'An adaptive deep reinforcement learning approach for MIMO PID control of mobile robots', *ISA Transactions*. doi: 10.1016/j.isatra.2020.02.017.

Cho, K. *et al.* (2014) 'Learning phrase representations using RNN encoder-decoder for statistical machine translation', in *arXiv preprint arXiv:1406.1078*, pp. 1724–1734. doi: 10.3115/v1/d14-1179.

Chung, J. *et al.* (2014) ‘Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling’, pp. 1–9. Available at: http://arxiv.org/abs/1412.3555.

Davison, E. J. (1976) ‘Multivariable tuning regulators: the feedforward and robust control of a general servomechanism problem’, *IEEE Transactions on Automatic Control*, 21(4), p. 631. doi: 10.1109/TAC.1976.1101251.

DNV-GL (2018) *Class Guideline (DNVGL-CG-0130): Wave loads*.

Duchi, J., Hazan, E. and Singer, Y. (2011) ‘Adaptive subgradient methods for online learning and stochastic optimization’, *Jornal of Machine Learning Research*.

Fossen, T. I. and Perez, T. (2009) ‘Kalman filtering for positioning and heading control of ships and offshore rigs: Estimating the effects of waves, wind, and current’, *IEEE Control Systems*, 29(6), pp. 32–46. doi: 10.1109/MCS.2009.934408.

Glorot, X. and Bengio, Y. (2010) ‘Understanding the difficulty of training deep feedforward neural networks’, in *Journal of Machine Learning Research*, pp. 249–256. Available at: http://www.iro.umontreal. (Accessed: 21 April 2020).

Gloye, A. *et al.* (2005) ‘Learning to drive and simulate autonomous mobile robots’, in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, pp. 160–171. doi: 10.1007/978-3-540-32256-6_13.

Guarize, R. *et al.* (2007) ‘Neural networks in the dynamic response analysis of slender marine structures’, *Applied Ocean Research*, 29(4), pp. 191–198. doi: 10.1016/j.apor.2008.01.002.

Guo, Y. *et al.* (2019) ‘Spottune: Transfer learning through adaptive fine-tuning’, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 4800–4809. doi: 10.1109/CVPR.2019.00494.

Hahnioser, R. H. R. *et al.* (2000) ‘Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit’, *Nature*. Nature Publishing Group, 405(6789), pp. 947–951. doi: 10.1038/35016072.

Harbonn, J. (1971) ‘The Terebel dynamic positioning system - Results of five years of field work and experiments’, in *Proceedings of the Annual Offshore Technology Conference*, pp. 813–821. doi: 10.4043/1499-ms.

Hassani, V. and Pascoal, A. M. (2015) ‘Wave filtering and dynamic positioning of marine vessels using a linear design model: Theory and experiments', *Operations Research/ Computer Science Interfaces Series*. Springer New York LLC, 58, pp. 315–343. doi: 10.1007/978-3-319-16133-4_17.

He, K. *et al.* (2015) 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification', in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034. doi: 10.1109/ICCV.2015.123.

Hinton, G., Srivastava, N. and Swersky, K. (2012) *Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude*, *COURSERA: Neural Networks for Machine Learning*. Available at: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf%0Ahttp://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (Accessed: 21 April 2020).

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*. MIT Press Journals, 9(8), pp. 1735–1780. doi: 10.1162/neco.1997.9.8.1735.

Im, N.-K. and Hasegawa, K. (2002) 'Motion Identification using Neural Networks and Its Application to Automatic Ship Berthing under Wind', *Jornal of Ship and Ocean Technology*, 6(1), pp. 16–26. Available at: http://www.naoe.eng.osaka-u.ac.jp/~hase/cv/papers/075.pdf (Accessed: 22 April 2020).

Im, N. (2007) 'All Direction Approach Automatic Ship Berthing Controller Using ANN (Artificial Neural Networks)', *Journal of Institute of Control, Robotics and Systems*. Available at: http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=JOJDCV_2007_v13n4_304 (Accessed: 22 April 2020).

Im, N. and Hasegawa, K. (2001) 'Automatic Ship Berthing Using Parallel Neural Controller', *IFAC Proceedings Volumes*. Elsevier BV, 34(7), pp. 51–57. doi: 10.1016/s1474-6670(17)35058-9.

Im, N. K. and Nguyen, V. S. (2018) 'Artificial neural network controller for automatic ship berthing using head-up coordinate system', *International Journal of Naval Architecture and Ocean Engineering*. Elsevier Ltd, 10(3), pp. 235–249. doi: 10.1016/j.ijnaoe.2017.08.003.

IMCA (2000) *Specification for DP capability plots*. Available at: www.imca-int.comAB (Accessed: 19 December 2019).

Ioffe, S. and Szegedy, C. (2015) 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', in *32nd International Conference on Machine Learning, ICML 2015*. International Machine Learning Society (IMLS), pp. 448–456.

Jeon, M.-R. *et al.* (2017) 'A Study on the Dynamic Positioning Control Algorithm Using Fuzzy Gain Scheduling PID Control Theory', *Journal of the Society of Naval Architects of Korea*. The Society of Naval Architects of Korea, 54(2), pp. 102–112. doi: 10.3744/snak.2017.54.2.102.

Jon, A. and Breivik, M. (2009) 'Autotuning Aspects for Dynamic Positioning Systems', in *8th IFAC International Conference on Manoeuvring and Control of Marine Craft*. Elsevier, pp. 334–339. doi: 10.3182/20090916-3-br-3001.00054.

Kingma, D. P. and Ba, J. L. (2015) 'Adam: A method for stochastic optimization', in *3rd International Conference on Learning Representations*. International Conference on Learning Representations, ICLR.

Koschorrek, P. *et al.* (2015) 'Dynamic positioning with active roll reduction using Voith Schneider Propeller', in *IFAC-PapersOnLine*. Elsevier, pp. 178–183. doi: 10.1016/j.ifacol.2015.10.277.

Lee, D., Lee, S. J. and Seo, Y. J. (2020a) 'Application of recent developments in deep learning to ANN-based automatic berthing systems', *International Journal of Engineering and Technology Innovation*, 10(1), pp. 75–90. doi: 10.46604/ijeti.2020.4354.

Lee, D., Lee, S. J. and Seo, Y. J. (2020b) 'Application of recent developments in deep learning to ANN-based automatic berthing systems', *International Journal of Engineering and Technology Innovation*, 10(1), pp. 75–90. Available at: https://sci-hub.si/http://search.proquest.com/openview/8d72093a7a3a6859cb91e4c16c8d0190/1?pq-origsite=gscholar&cbl=4365211 (Accessed: 20 March 2020).

Lee, S. J. (2008) *The effects of LNG-Sloshing on the global responses of LNG-Carriers*. Texas A&M University.

Levenberg, K. (1944) 'A method for the solution of certain non-linear problems in least squares', *Quarterly of Applied Mathematics*, 2(2), pp. 164–168. doi: 10.1090/qam/10666.

Lillicrap, T. P. *et al.* (2016) 'Continuous control with deep reinforcement learning', in *4th International Conference on Learning Representations*. Available at: https://goo.gl/J4PIAz (Accessed: 11 December 2019).

Maciejowski, J. M. (1989) *Multivariable feedback design*, *Electronic Systems Engineering Series*. Available at: https://sci-hub.st/https://ui.adsabs.harvard.edu/abs/1989mfd..book.....M/abstract (Accessed: 7 July 2020).

Macqueen, J. (1967) ‘Some methods for classification and analysis of multivariate observations’, in *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*.

Martin, P. and Katebi, R. (2005) ‘Multivariable PID tuning of dynamic ship positioning control systems’, *Journal of Marine Engineering and Technology*, 4(2), pp. 11–24. doi: 10.1080/20464177.2005.11020190.

Mnih, V. *et al.* (2013) ‘Playing atari with deep reinforcement learning’, in *arXiv preprint arXiv:1312.5602*.

Newman, J. N. (2019) *Marine Hydrodynamics*, *Marine Hydrodynamics*. MIT press. doi: 10.7551/mitpress/4443.001.0001.

Nguyen, T. D., Sørensen, A. J. and Tong Quek, S. (2007) ‘Design of hybrid controller for dynamic positioning from calm to extreme sea conditions’, *Automatica*. Pergamon, 43(5), pp. 768–785. doi: 10.1016/j.automatica.2006.11.017.

Orcina (2019) *Vessel Theory : Wave Drift and Sum Frequency Loads*. Available at: https://www.orcina.com/webhelp/OrcaFlex/Content/html/Vesseltheory,Wavedriftandsumfrequencyloads.htm (Accessed: 12 July 2020).

Penttinen, J. and Koivo, H. N. (1980) ‘Multivariable tuning regulators for unknown systems’, *Automatica*, 16(4), pp. 393–398. doi: 10.1016/0005-1098(80)90023-0.

Plappert, M. *et al.* (2018) ‘Parameter space noise for exploration’, in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) ‘Learning representations by back-propagating errors’, *Nature*. Nature Publishing Group, 323(6088), pp. 533–536. doi: 10.1038/323533a0.

Sidarta, D. E. *et al.* (2017) ‘Prediction of offshore platform mooring line tensions using artificial neural network’, in *Proceedings of the 27th International Ocean and Polar Engineering Conference*, pp. 1–11.

Silver, D. *et al.* (2014) ‘Deterministic policy gradient algorithms’, in *31st International Conference on Machine Learning, ICML 2014*, pp. 605–619.

Simões, M. G., Tiquilloca, J. L. M. and Morishita, H. M. (2002) ‘Neural-Network-based prediction of mooring forces in floating production storage and offloading systems’, *IEEE Transactions on*

Industry Applications, 38(2), pp. 457–466. doi: 10.1109/28.993167.

Sohn, K.-H. (1992) 'Hydrodynamic Forces and Maneuvering Characteristics of Ships at Low Advance Speed', *Journal of the Society of Naval Architects of Korea*, 29(3), pp. 90–101. Available at: https://sci-hub.si/http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=DHJSCN_1992_v29n3_90 (Accessed: 22 April 2020).

Song, S. S. *et al.* (2016) 'A study on the feedforward control algorithm for dynamic positioning system using ship motion prediction', *Journal of the Korean Society of Marine Environment and Safety*, 22(1), pp. 129–137. doi: 10.7837/kosomes.2016.22.1.129.

Srivastava, N. *et al.* (2014) 'Dropout: A simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, 15, pp. 1929–1958.

Sutton, R. S. *et al.* (2000) 'Policy gradient methods for reinforcement learning with function approximation', in *Advances in Neural Information Processing Systems*, pp. 1057–1063.

Teoh, E. K. and Yee, Y. E. (1991) 'Implementation of adaptive controllers using digital signal processor chips', in *IFAC Symposia Series*, pp. 109–113. doi: 10.1016/s1474-6670(17)51305-1.

Watkins, C. J. C. H. and Dayan, P. (1992) 'Q-learning', *Machine Learning*. Springer Science and Business Media LLC, 8(3–4), pp. 279–292. doi: 10.1007/bf00992698.

Weingarth, L. (2006) 'Refining the DP Watch Circle', in *MTS Dynamic Position Conference*. Available at: https://sci-hub.st/http://dynamic-positioning.com/proceedings/dp2006/abstract_op_procedures_weingarth.pdf (Accessed: 13 July 2020).

De Wit, C. (2009) *Optimal thrust allocation methods for dynamic positioning of ships, Delft University of Technology, Netherlands*.

Xu, S. *et al.* (2019) 'A fuzzy rule-based PID controller for dynamic positioning of vessels in variable environmental disturbances', *Journal of Marine Science and Technology (Japan)*. Springer Tokyo. doi: 10.1007/s00773-019-00689-2.

Yamamoto, M. and Morooka, C. K. (2005) 'Dynamic positioning system of semi-submersible platform using fuzzy control', *Journal of the Brazilian Society of Mechanical Sciences and Engineering*. Brazilian Society of Mechanical Sciences and Engineering, 27(4), pp. 449–455. doi:

10.1590/S1678-58782005000400014.

Yeung, R. W. (1983) *International Workshop on Ship and Platform Motions*. Available at: https://books.google.co.kr/books?id=11hJAQAAIAAJ&printsec=frontcover&hl=ko#v=onepage&q&f=false (Accessed: 1 July 2020).

Ziegler, J. G. and Nathaniel, N. B. (1942) 'Optimum settings for automatic controllers', *ASME*. Available at: www.driedger.ca...IIthankktheeASMEEforrallowinggthissimportanttworkktoobeesoofreelyyavailabl e... (Accessed: 19 December 2019).