Development of a Ship Autopilot using Neural Network

Stefanos Gouletas

Diploma Thesis

School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Professor G. Papalambrou

Committee Members:

Professor K. Spyrou
Associate Professor C. Papadopoulos

February 2020

Abstract

This Thesis concerns the development of a neural network autopilot which will be used for vessel controlling. The main purpose of this project is to use alternative and more sophisticated technologies like artificial intelligence in contrast with the commonly used PID controllers. The Neural Network controller was constructed using Python libraries (Tensorflow and Keras), which have been used for the easier and more efficient training and building. For the representation of the ship system, we used two different mathematical models, one linear (Nomoto Model) and one more complicated based on the hydrodynamic coefficients, in order to make the system more realistic (4DOF Model). For testing, we executed a simple course keeping simulation to check the suitability of the controller. Even then in order to observe the efficiency of the neural network controller, we use it in a path following scenario in which the vessel should follow a predefined course. The performance of the neural network controller is compared to classical PID control results which are implemented for the same simulation scenarios. We conclude that artificial intelligence and more specifically Neural Networks is a very important tool which could be used for more efficient and effective controllers that could process more informations as sea state, weather and fuel optimization.

Contents

List of Figures

List of Tables

List Of Abbreviations

ANN	Artificial Neural Networks
AP	Aft Per Per-Pendicular
API	Application Programming Interface
BL	Base Line
CG	Center of Gravity
CNN	Convolutional Neural Networks
CNS	Central Nervous System
CP	Center of Pressure
DOF	Degrees of Freedom
FRF	Frequency Response Function
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical user interface
INS	Inertial Navigation System
LOS	Line of Sight
LSTM	Long/Short Term Memory
MLP	Multilayer Perceptrons
NN	Neural Networks
PDE	Partial Differential Equation
PID	Proportional Integral Derivative
PVA	Position, Velocity and Acceleration
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
TPU	Tensorflow Processing Unit

Chapter 1

Introduction

1.1 Literature Review

A ship requires the continuous attention of the captain or other crew member to sail safely. As vessels range increases and voyages become larger, the constant attention leads to serious fatigue. An autopilot is designed to perform some of the tasks of the navigator. In recent years the number of autopilots which are used on ships is on increase. Autopilot is a system that can control the vessel (steer the rudder automatically, follow desired heading angle or predefined path) without constant manual control by a human operator being required. As a general rule, human operators are not replaced by autopilots, but instead they assist them in controlling the ship. So far, the PID based autopilot is the most common system that it is used in market. This structure of autopilot is so common because it is very simple to be built and maintained. Moreover, PID is an efficient controller owing to its reliability and its low creation cost. Unfortunately, despite its advantages, it has a major drawback in making it unsatisfactory. In nature there are plenty of variable factors like weather, waves and the non-linearity of ship. A simple PID based autopilot cannot deal with these factors and fails. In recent years, artificial neural networks have gained a widely attention in control application in order to cope with the failure of PID. The artificial neural networks have the ability to model non-linear systems and can be exploited to be constructed non-linear controllers. They can handle the variability of system and its disturbances such as currents, waves and air forces. Artificial neural networks (ANN) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The structure of neural networks allows them to manipulate many data which are non-linear. These ability is very important for ship's controller. Thus, the big challenge is the commercial launching of an autopilot which have been developed on artificial intelligence foundation. Artificial intelligence / machine learning is one of the biggest trends in our time, however, less than a decade after breaking the Nazi encryption machine Enigma and helping the Allied Forces win World War II, mathematician Alan Turing changed history a second time with a simple question: "Can machines think?". Turing's paper "Computing Machinery and Intelligence" (1950), and it's subsequent Turing Test, established the fundamental goal and vision of artificial intelligence. AI is a computer system able to perform tasks that ordinarily require human intelligence, videlicet, artificial intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans.

Vessel's autopilot systems have been thoroughly framed and studied in the Handbook of Marine Craft Hydrodynamics and Motion Control published by Thor I. Fossen. The whole process of designing and manufacturing an autopilot is described in this research. One of the most important point is to choose a mathematical model which can properly depict the vessel system in order to be used for prediction or real-time simulation. There are plenty of mathematical models for a vessel in which the complexity and the number of differential equations vary accordingly the purpose of study. The most easy and simple model is 1 DOF which is usually based on the model representation of Nomoto. This model is linear and using for heading control. 4DOF is a more complicated model using for better representation of vessel's response. Autopilot is constructed as three independent blocks denoted as the guidance, navigation and control (GNC) systems. Using these blocks the autopilot can use the vessel's coordinates in order to calculate the more effective rudder angle which is able to lead the vessel on a predefined path.

1.2 Problem Formulation

The main objective of this Thesis is the development of a Neural Network Control System capable to calculate the optimal rudder command in order to the vessel can follow a predefined path or a fixed desired heading angle. The Neural Network must be properly trained, otherwise, the controller will not be capable to achieve the mentioned objective. In this thesis, we will take advantage of neural networks abilities in order to build a more effective controller in comparison with the other ordinary controllers like PID.

1.3 Outline

In this thesis, a multi layer neural network with one hidden layer is built to replace a common PID ship controller. Its purpose is to keep ship's course and to make the ship capable of following a path. The neural network will be trained with a method known as supervised, which is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. As there is no real small scale vessel for data acquisition, a mathematical model of the ship should be used. In this thesis, there are two different representation of the ship's system, one using a simply linear mathematical model, Nomoto, and one using a more analytical non-linear model with four degrees of freedom. Both are used to acquire data and simulate certain scenarios. The implementation will be executed on PYTHON programming language due to its simplicity and performance.

The theoretical part of neural networks is presented in the Chapter (2) of this paper. This section is divided into the biological and artificial neural networks. The Chapter (3) contains the description of the autopilot and its division to the Guidance, Navigation and Control System. The used programming tools for the implementation of the mentioned neural network controller will be described in Chapter (4). The analysis of the two mathematical representations of the ship system (Nomoto and 4DOF models) is presented in Chapter (5). Finally, in the last Chapter (6) there is a description of the training and simulation implementation (method), as also the results of them.

Chapter 2

Neural Network Theory

2.1 Biological Neural Networks

Artificial Neural Networks (NN) draw much of their inspiration from the biological nervous system. It is therefore very useful to have some knowledge of the way this system is organized [3]. Most living creatures, which have the ability to adapt to a changing environment, need a controlling unit which is able to learn. Higher developed animals and humans use very complex networks of highly specialized neurons to perform this task. The control unit - or brain - can be divided in different anatomic and functional sub-units, each having certain tasks like vision, hearing, motor and sensor control. The brain is connected by nerves to the sensors and actors in the rest of the body. The brain consists of a very large number of neurons, about 10^{11} in average. These can be seen as the basic building bricks for the central nervous system (CNS). The neurons are interconnected at points called synapses. The complexity of the brain is due to the massive number of highly interconnected simple units working in parallel, with an individual neuron receiving input from up to 10000 others. The neuron contains all structures of an animal cell. The complexity of the structure and of the processes in a simple cell is enormous. Even the most sophisticated neuron models in artificial neural networks seem comparatively toy-like. Structurally the neuron can be divided in three major parts: the cell body (soma), the dentrites, and the axon, as illustrated on Figure 2.1.

The cell body contains the organelles of the neuron and also the 'dentrites' are originating there. These are thin and widely branching fibers, reaching out in different directions to make connections to a larger number of cells within the cluster. Input connection are made from the axons of other cells to the dentrites or directly to the body of the cell. These are known as axondentrititic and axonsomatic synapses. There is only one axon per neuron. It is a single and long fiber, which transports the output signal of the cell as electrical impulses (action potential) along its length. The end of the axon may divide in many branches, which are then connected to other cells. The branches have the function to fan out the signal to many other inputs. There are many different types of neuron cells found in the nervous system. The differences are due to their location and function. The neurons perform basically the following function: all the inputs to the cell, which may vary by the strength of the connection or the frequency of the incoming signal, are summed up. The input sum is processed by a threshold function and produces an output signal. The processing time of about 1ms per cycle and transmission speed of the neurons of about 0.6 to 120 ms are comparingly slow to a modern

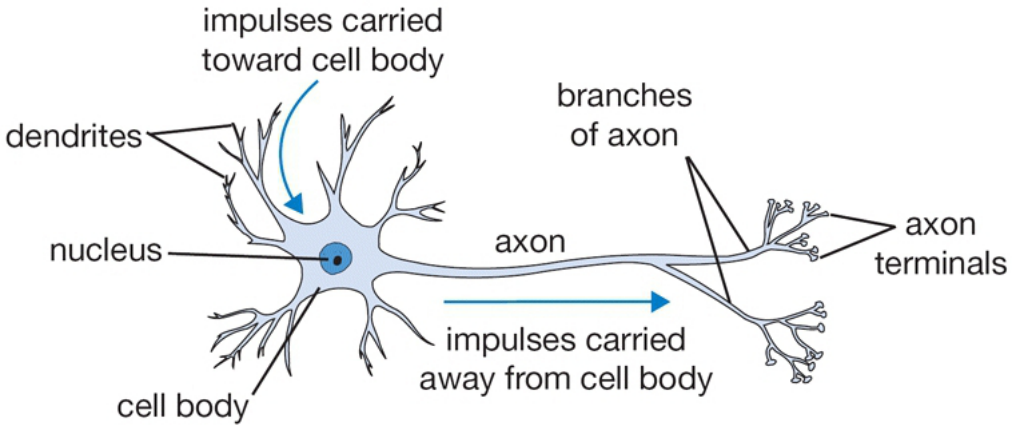

Figure 2.1: Neuron Structure [4]

computer. The brain works in both a parallel and serial way. The parallel and serial nature of the brain is readily apparent from the physical anatomy of the nervous system. That there is serial and parallel processing involved can be easily seen from the time needed to perform tasks. For example a human can recognize the picture of another person in about 100 ms. Given the processing time of 1 ms for an individual neuron this implies that a certain number of neurons, but less than 100, are involved in serial; whereas the complexity of the task is evidence for a parallel processing, because a difficult recognition task can not be performed by such a small number of neurons. This phenomenon is known as the 100-step-rule.

Biological neural systems usually have a very high fault tolerance. Experiments with people with brain injuries have shown that damage of neurons up to a certain level does not necessarily influence the performance of the system, though tasks such as writing or speaking may have to be learned again. This can be regarded as re-training the network.

2.2 Artificial Neural Networks

Artificial Neural Network (ANN) is an efficient computing system which based on the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel [3]. Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units. The ANN structure is illustrated in Figure 2.2.

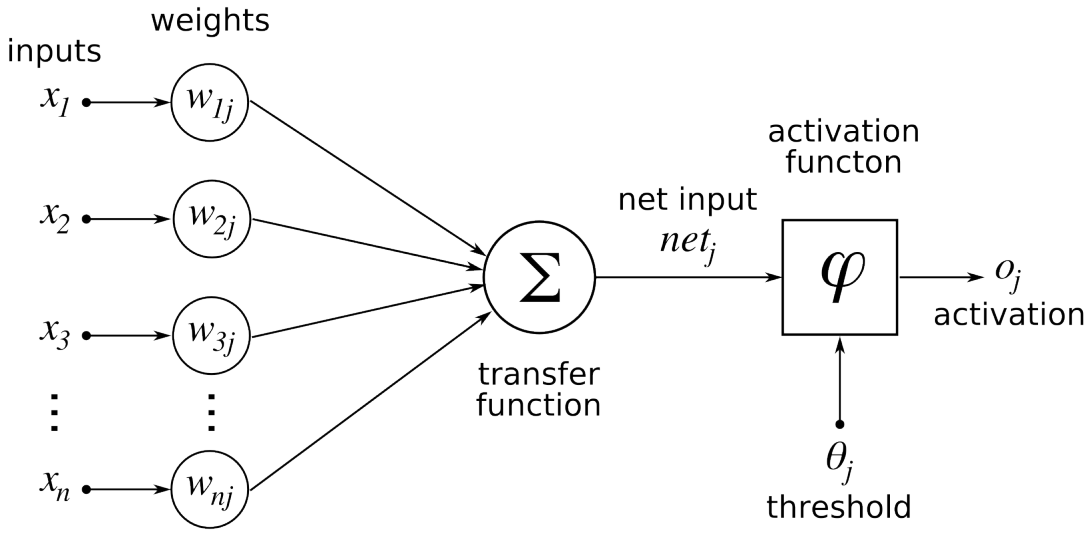

Figure 2.2: Artificial Neural Network Structure

2.2.0.1 Network Topology

There are several kinds of artificial neural networks such as the deep feed forward, the Recurrent (RNN), the Long/Short Term Memory (LSTM) and the Convolutional (CNN) [4]. These types of networks are implemented based on the mathematical operations and a set of parameters required to determine the output. In this thesis a deep feed forward (multilayer) neural network was chosen as the vessel's controller because of its simplicity.

Feedforward Network

Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons(MLPs), are the quintessential deep learning models. The goal of a feedforward network

is to approximate some function f. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks. An example of a neural network with 2 hidden layers is depicted on Figure 2.3.

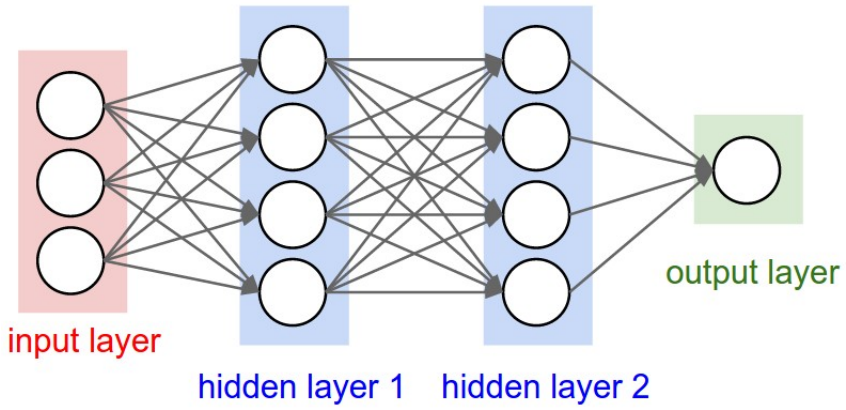

Figure 2.3: Feedforward neural network with 2 hidden layers [4]

Single-Layer Perceptron

The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights as shown on Figure 2.4. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called artificial neurons or linear threshold units. In the literature the term perceptron often refers to networks consisting of just one of these units. A perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two.

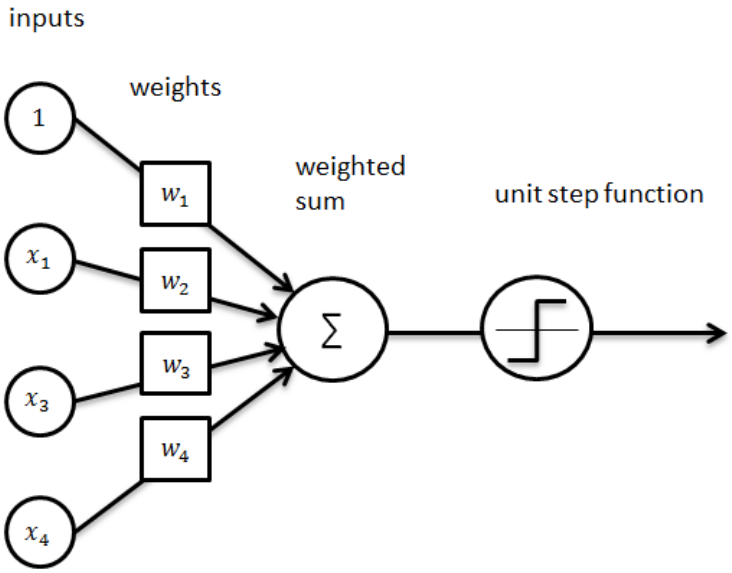

Figure 2.4: The Perceptron

Multi-Layer Perceptron

A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function. Multilayer perceptrons are often applied to supervised learning problems: they train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE).

2.2.0.2 Fundamentals of Learning

Activation Functions

Activation function is just a node that is added to the output end of any neural network. A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. There are a great number of activation functions, so only the most common ones are to be described.

Step function The simplest activation function is the binary step function, which can only take on two values. If the input is above a certain threshold, the function changes from one value to another, but otherwise remains constant. This implies that the function is not differentiable at the threshold and for the rest the derivative is 0. The equation of this function is illustrated below. See Figure 2.5.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases} \tag{2.1}$$

Linear function It is a simple linear function of the form as the below equation. Basically, the input passes to the output without any modification. See Figure 2.6

$$f(x) = x \tag{2.2}$$

Sigmoid function It takes a real-valued number and squashes it into a range between 0 and 1. It is also used in the output layer where our end goal is to predict probability. It converts large negative numbers to 0 and large positive numbers to 1. The illustration of this function is on Figure 2.7. Mathematically it is represented as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Tanh function is also known as the hyperbolic tangent activation function. Similar to sigmoid, tanh also takes a real-valued number but squashes it into a range between -1 and 1. Unlike sigmoid, tanh outputs are zero-centered since the scope is between -1 and 1. You can think of a tanh function as two sigmoids put together. In practice, tanh is preferable over sigmoid. The negative inputs considered as strongly negative, zero input values mapped near zero, and the positive inputs regarded as positive. The function illustrated on Figure 2.8.

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.4}$$

Relu function ReLU is half-rectified from the bottom as you can see from the Figure 2.9. This means that when the input x < 0 the output is 0 and if x > 0 the output is x. This activation makes the network converge much faster. Mathematically, it is given by this simple expression

$$f(x) = max(0, x) \tag{2.5}$$

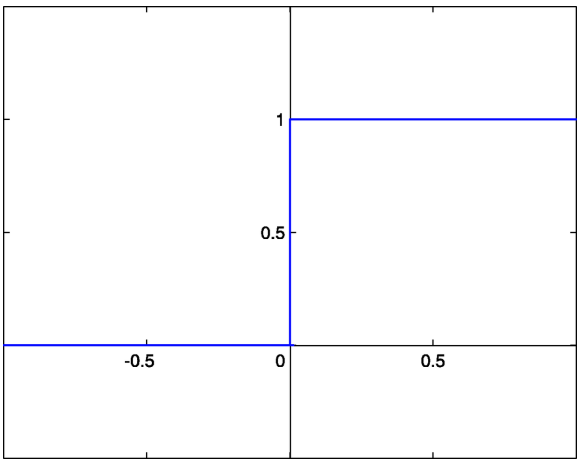

Figure 2.5: Step Function

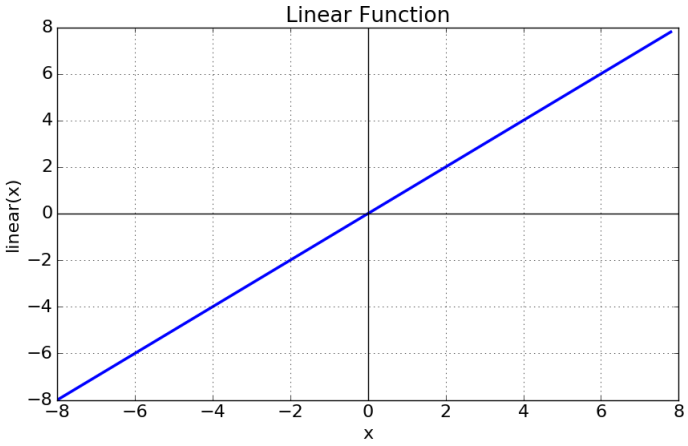

Figure 2.6: Linear Function

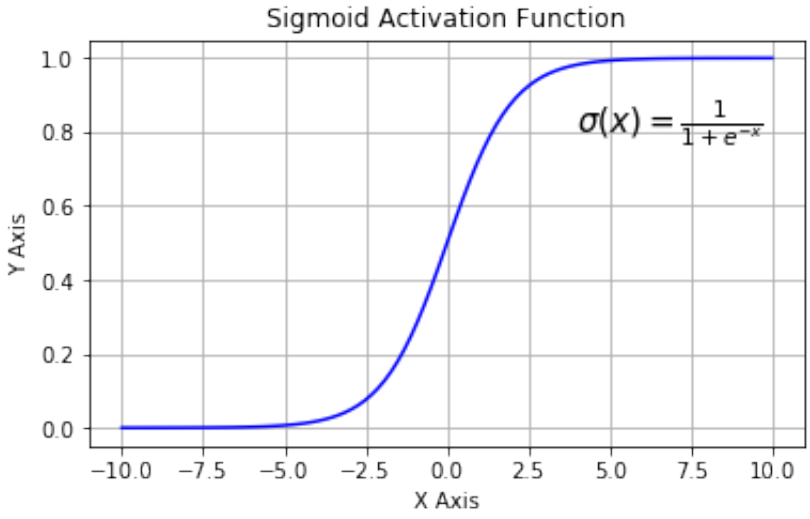

Figure 2.7: Sigmoid Function

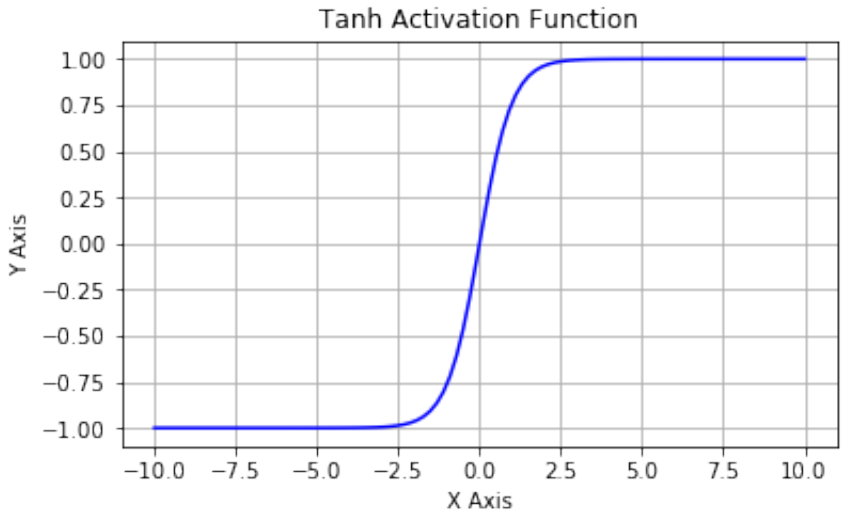

Figure 2.8: Tanh Function

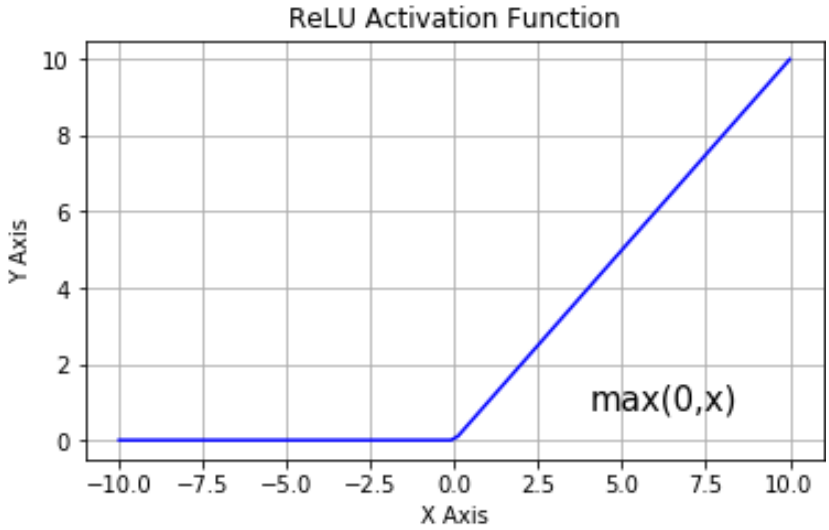

Figure 2.9: Relu Function

Back-Propagation

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks [3]. Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to other cell's dendrites. The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state. Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. Backpropagation can be used for both classification and regression problems, but we will focus on classification in this thesis.

The Back-Propagation is one of the most popular NN algorithm. After choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. For understanding purposes, we will use a neural network with three layers in order to represent the relative procedure. The neural network consists of one input layer with two inputs neurons, one hidden layer with two neurons and one output layer with a single neurons as illustrated Figure 2.10.

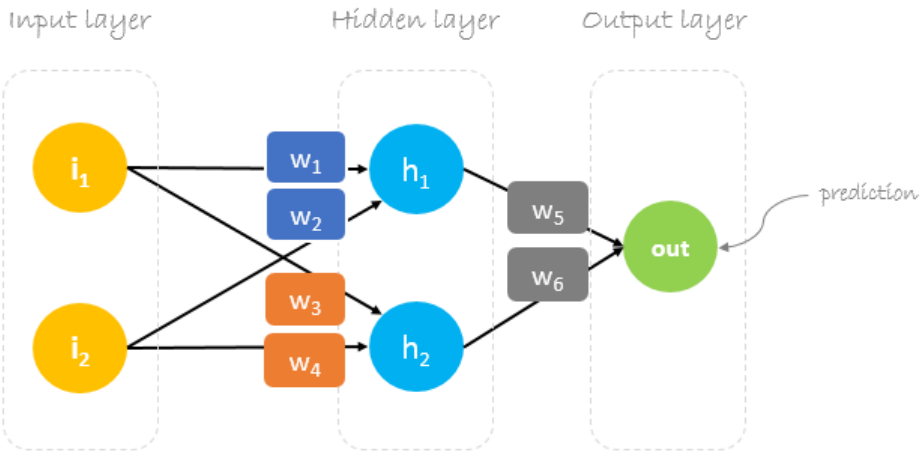

Figure 2.10: Neural Network with three layers [3]

Feed-forward computation Feed forward computation or forward pass is two step process. First part is getting the values of the hidden layer nodes and second part is using those values from hidden layer to compute value or values of output layer.

Back propagation Now, the performance of neural network evaluated by calculating the difference between the actual output and predicted one. During the first attempt the

prediction will not be close to actual output. We will define as error function the following formula in order to it is always positive because of the square.

$$Error = \frac{1}{2}(prediction - actual)^2 \tag{2.6}$$

Our main goal of the training is to reduce the error or the difference between prediction and actual output. Since actual output is constant, "not changing", the only way to reduce the error is to change prediction value. The question now is, how to change prediction value? By decomposing prediction into its basic elements we can find that weights are the variable elements affecting prediction value. In other words, in order to change prediction value, we need to change weights values.

$$prediction = (h_1)w_5 + (h_2)w_6 \tag{2.7}$$

$$prediction = (i_1w_1 + i_2w_2)w_5 + (i_1w_3 + i_2w_4)w_6 \tag{2.8}$$

Backpropagation, short for "backward propagation of errors", is a mechanism used to update the weights using gradient descent. It calculates the gradient of the error function with respect to the neural network's weights. The calculation proceeds backwards through the network.

Gradient descent *is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize th error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.*

$${w_x}^* = w_x - a(\frac{\vartheta Error}{\vartheta w_x}) \tag{2.9}$$

For example, to update w_6, we take the current w_6 and subtract the partial derivative of error function with respect to w_6. Optionally, we multiply the derivative of the error function by a selected number to make sure that the new updated weight is minimizing the error function; this number is called learning rate α.

$${w_6}^* = w_6 - a(\frac{\vartheta Error}{\vartheta w_6}) \tag{2.10}$$

The derivation of the error function is evaluated by applying the chain rule as following:

$$\frac{\vartheta Error}{\vartheta w_6} = \frac{\vartheta Error}{\vartheta prediction} \times \frac{\vartheta prediction}{\vartheta w_6}$$

$$\frac{\vartheta Error}{\vartheta w_6} = \frac{\frac{1}{2}(prediction - actual)^2}{\vartheta prediction} \times \frac{\vartheta(i_1w_1 + i_2w_2)w_5 + (i_1w_3 + i_2w_4)w_6}{\vartheta w_6}$$

$$\frac{\vartheta Error}{\vartheta w_6} = 2 \times \frac{1}{2}(prediction - actual)\frac{(prediction - actual)}{\vartheta prediction} \times (i_1w_3 + i_2w_4)$$

$$\frac{\vartheta Error}{\vartheta w_6} = (prediction - actual) \times (h_2)$$

$$\frac{\vartheta Error}{\vartheta w_6} = \Delta h_2$$

where $\Delta = prediction - actual$.

So to update any other weights existing between the output and the hidden layer as w_5 and w_6 we can apply the following formula:

$$w_5{}^* = w_5 - a\Delta h_1 \tag{2.11}$$
$$w_6{}^* = w_6 - a\Delta h_2 \tag{2.12}$$

However, when moving backward to update $w_1, w_2, w_3 and w_4$ existing between input and hidden layer, the partial derivative for the error function with respect to w_1, for example, will be as following.

$$\frac{\vartheta Error}{\vartheta w_1} = \frac{\vartheta Error}{\vartheta prediction} \times \frac{\vartheta prediction}{\vartheta h_1} \times \frac{\vartheta h_1}{\vartheta w_1}$$
$$\frac{\vartheta Error}{\vartheta w_1} = \frac{\vartheta \frac{1}{2}(prediction - actual)^2}{\vartheta prediction} \times \frac{\vartheta (h_1)w_5 + (h_2)w_6}{\vartheta h_1} \times \frac{\vartheta i_1 w_1 + i_2 w_2}{\vartheta w_1}$$
$$\frac{\vartheta Error}{\vartheta w_1} = 2 \times \frac{1}{2}(prediction - actual)\frac{(prediction - actual)}{\vartheta prediction} \times (w_5) \times (i_1)$$
$$\frac{\vartheta Error}{\vartheta w_1} = (prediction - actual) \times (w_5 i_1)$$
$$\frac{\vartheta Error}{\vartheta w_1} = \Delta w_5 i_1$$

where $\Delta = prediction - actual$.

In summary, the update formulas for all weights will be as following:

$$w_6{}^* = w_6 - a(h_2 \times \Delta) \tag{2.13}$$
$$w_5{}^* = w_5 - a(h_1 \times \Delta) \tag{2.14}$$
$$w_4{}^* = w_4 - a(i_2 \times \Delta w_6) \tag{2.15}$$
$$w_3{}^* = w_3 - a(i_1 \times \Delta w_6) \tag{2.16}$$
$$w_2{}^* = w_2 - a(i_2 \times \Delta w_5) \tag{2.17}$$
$$w_1{}^* = w_1 - a(i_1 \times \Delta w_5) \tag{2.18}$$

Using derived formulas we can find the new weights and next we will repeat the forward passed and the prediction must be closer to actual output than the previously predicted one. The same process have to be repeated of backward and forward pass until error is close or equal to zero.

Learning Methods

Supervised Learning Supervised learning typically begins with an established set of data and a certain understanding of how that data is classified. Supervised learning is intended to find patterns in data that can be applied to an analytics process. This data has labeled features that define the meaning of data. For example, there could be millions of

images of animals and include an explanation of what each animal is and then you can create a machine learning application that distinguishes one animal from another. By labeling this data about types of animals, you may have hundreds of categories of different species. Because the attributes and the meaning of the data have been identified, it is well understood by the users that are training the modeled data so that it fits the details of the labels. When the label is continuous, it is a regression; when the data comes from a finite set of values, it known as classification. In essence, regression used for supervised learning helps you understand the correlation between variables. An example of supervised learning is weather forecasting. By using regression analysis, weather forecasting takes into account known historical weather patterns and the current conditions to provide a prediction on the weather. Supervised learning procedure is shown on Figure 2.11. The algorithms are trained using preprocessed examples, and at this point, the performance of the algorithms is evaluated with test data. Occasionally, patterns that are identified in a subset of the data can't be detected in the larger population of data. If the model is fit to only represent the patterns that exist in the training subset, you create a problem called overfitting. Overfitting means that your model is precisely tuned for your training data but may not be applicable for large sets of unknown data. To protect against overfitting, testing needs to be done against unforeseen or unknown labeled data. Using unforeseen data for the test set can help you evaluate the accuracy of the model in predicting outcomes and results. Supervised training models have broad applicability to a variety of business problems, including fraud detection, recommendation solutions, speech recognition, or risk analysis.

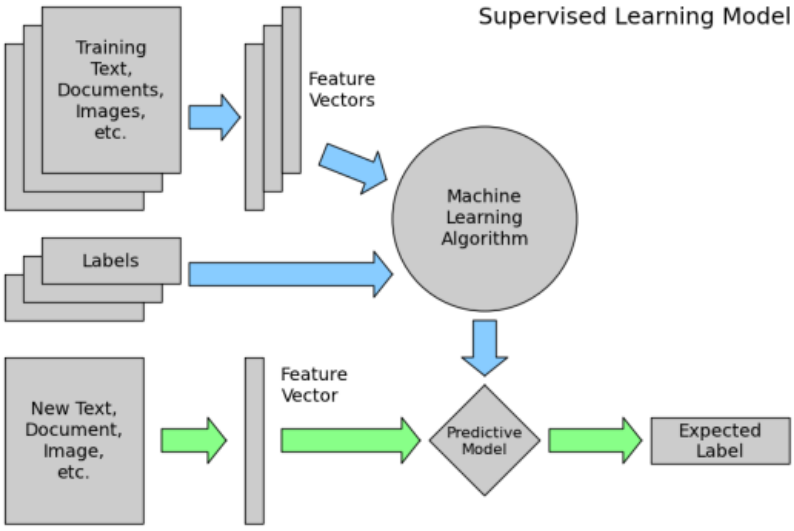

Figure 2.11: Supervised learning process [4]

Unsupervised Learning Unsupervised learning is best suited when the problem requires a massive amount of data that is unlabeled [5]. For example, social media applications, such as Twitter, Instagram, Snapchat, and so on all have large amounts of unlabeled data. Understanding the meaning behind this data requires algorithms that can begin to understand the meaning based on being able to classify the data based on the patterns or clusters it finds.

Therefore, the supervised learning conducts an iterative process of analyzing data without human intervention. Unsupervised learning is used with email spam-detecting technology. There are far too many variables in legitimate and spam emails for an analyst to flag unsolicited bulk email. Instead, machine learning classifiers based on clustering and association are applied in order to identify unwanted email. Unsupervised learning algorithms segment data into groups of examples (clusters) or groups of features. The unlabeled data creates the parameter values and classification of the data. In essence, this process adds labels to the data so that it becomes supervised. Unsupervised learning can determine the outcome when there is a massive amount of data. In this case, the developer doesn't know the context of the data being analyzed, so labeling isn't possible at this stage. Therefore, unsupervised learning can be used as the first step before passing the data to a supervised learning process. Figure 2.12 illustrates the unsupervised learning process. Unsupervised learning algorithms can help businesses understand large volumes of new, unlabeled data. Similarly to supervised learning (see the preceding section), these algorithms look for patterns in the data; however, the difference is that the data is not already understood. For example, in healthcare, collecting huge amounts of data about a specific disease can help practitioners gain insights into the patterns of symptoms and relate those to outcomes from patients. It would take too much time to label all the data sources associated with a disease such as diabetes. Therefore, an unsupervised learning approach can help determine outcomes more quickly than a supervised learning approach.

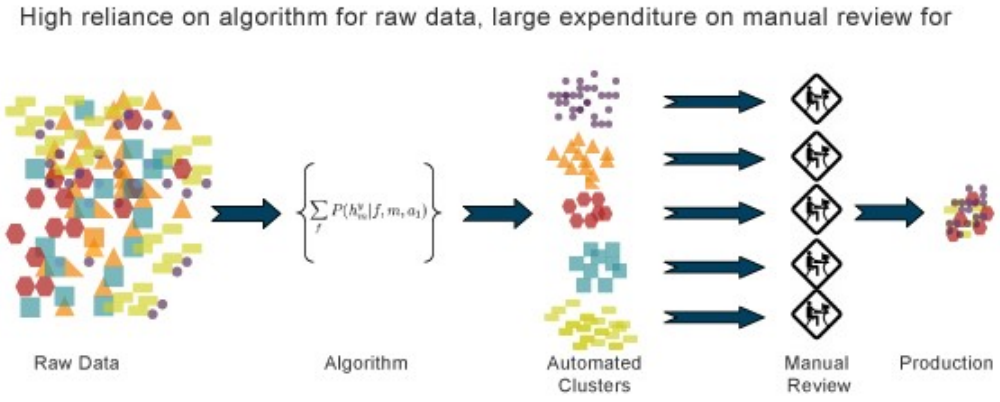

Figure 2.12: Unsupervised learning process [5]

Reinforcement Learning Reinforcement learning is a behavioral learning model [6]. The algorithm receives feedback from the analysis of the data so the user is guided to the best outcome. Reinforcement learning differs from other types of supervised learning because the system isn't trained with the sample data set. Rather, the system learns through trial and error. Therefore, a sequence of successful decisions will result in the process being "reinforced" because it best solves the problem at hand. One of the most common applications of reinforcement learning is in robotics or game playing. Take the example of the need to train a robot to navigate a set of stairs. The robot changes its approach to navigating the terrain based on the outcome of its actions. When the robot falls, the data is recalibrated so the steps are navigated differently until the robot is trained by trial and error to understand how to climb

stairs. In other words, the robot learns based on a successful sequence of actions. The learning algorithm has to be able to discover an association between the goal of climbing stairs successfully without falling and the sequence of events that lead to the outcome. Reinforcement learning is also the algorithm that is being used for self-driving cars. In many ways, training a self-driving car is incredibly complex because there are so many potential obstacles. If all the cars on the road were autonomous, trial and error would be easier to overcome. However, in the real world, human drivers can often be unpredictable. Even with this complex scenario, the algorithm can be optimized over time to find ways to adapt to the state where actions are rewarded. One of the easiest ways to think about reinforcement learning is the way an animal is trained to take actions based on rewards. If the dog gets a treat every time he sits on command, he will take this action each time.

Chapter 3

Autopilot Theory

The autopilot or automatic pilot is a device or a mechanical, electrical or hydraulic system which can maintain a vessel on a predetermined (set) course without the need for human intervention. A common autopilot consist of a Guidance, Navigation and Control System that they are known as GNC system [7], Figure 3.1.

The earliest automatic pilots could do no more than maintain a fixed heading and they are still used to relieve the pilot on smaller boats during routine cruising. For ships, course-keeping capabilities were the first applications. Modern autopilots can, however, execute complex maneuvers like turning, docking operations, or make possible the control of inherently unstable vessels (such as submarines and some large oil tankers). Autopilots are used to steer surface ships, submarines, torpedoes, missiles, rockets, and spacecraft among others.

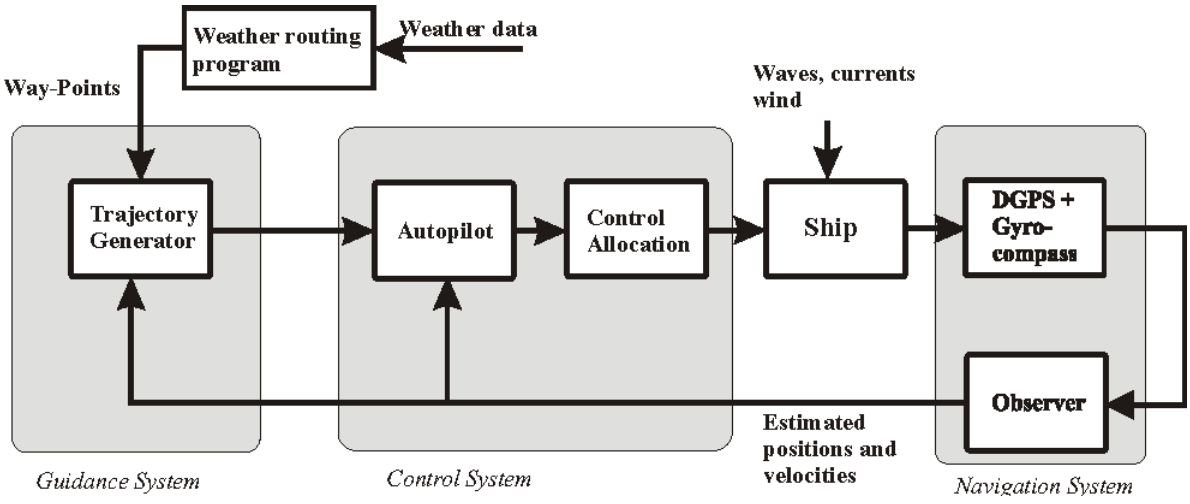

Figure 3.1: Guidance, Navigation and Control System [7]

3.1 Definitions

3.1.1 Definition of Guidance, Navigation and Control

In its most advance form, the GNC blocks represent three interconnected subsystems [7]:

Guidance is the action or the system that continuously computes the reference (desired) position, velocity and acceleration of a vessel to be used by the control system. These data are usually provided by human operator and the navigation system. The basic components of a guidance system are motion sensors, external data like weather data (wind speed and direction, wave height and slope, current speed and direction etc.) and a computer. The computer collects and processes the information, and then feeds the results to the vessel's control system. In many cases, advanced optimization techniques are used to compute optimal trajectory or path for the vessel to follow. This might include sophisticated features like fuel optimization, minimum time navigation, weather routing, collision avoidance, formation control and schedule meetings.

Navigation is the science of directing a craft by determining its position, course, and distance traveled. In some cases, velocity and acceleration are determined as well. This is usually done by using a satellite navigation system combined with motion sensors like accelerometers and gyros. The most advanced navigation system for marine applications is the inertial navigation system (INS). Navigation is derived from the Latin navis, "ship," and agere, "to drive." It originally denoted the art of ship driving, including steering and setting the sails. The skill is even more ancient than the word itself, and it has evolved over the course of many centuries into a technological science that encompasses the planning and execution of safe, timely, and economical operation ships, underwater vehicles, aircraft, and spacecraft.

Control is the action of determining the necessary control forces and moments to be provided by the vessel in order to satisfy a certain control objective. The desired control objective is usually seen in conjunction with the guidance system. Examples of control objectives are minimum energy, set-point regulation, trajectory tracking, path following, maneuvering etc. Constructing the control algorithm involves the design of feedback and feedforward control laws. The outputs from the navigation system, position, velocity and acceleration, are used for feedback control while feedforward control is implemented using signals available in the guidance system and other external sensors.

3.1.2 Definition of Degrees of Freedom and Motions of a Marine Vessel

In maneuvering, a marine vessel experiences motion in 6 degrees of freedom (DOF). The motion in the horizontal plane is referred to as surge (longitudinal motion, usually superimposed on the steady propulsive motion) and sway (sideways motion). Heading, or yaw (rotation about the vertical axis) describes the course of the vessel. The remaining three DOFs are roll (rotation about the longitudinal axis), pitch (rotation about the transverse axis), and heave (vertical motion). Roll is probably the most troublesome DOF, since it produces the highest accelerations and, hence, is the principal villain in seasickness. Similarly, pitching and heaving feel uncomfortable to humans. When designing ship autopilots, yaw is the primary mode for

feedback control. Station-keeping of a marine vessel implies stabilization of the surge, sway and yaw modes.

3.1.3 Set-Point Regulation versus Trajectory Tracking Control

In GNC it is important to distinguish between the following two important control objectives:

Set-Point Regulation: The most basic guidance system is a constant input (set-point) provided by a human operator. The corresponding controller will then be a regulator. Examples of set-point regulation are constant depth, trim, heel and speed control, etc. It could also be regulation to zero which is commonly required in roll and pitch for instance.

Trajectory Tracking Control: The objective is for the position and velocity of the vessel to track given desired time-varying position and velocity reference signals. The corresponding feedback controller must then be a trajectory tracking controller. Tracking control can be used for course-changing maneuvers, speed changing, attitude control, etc. An advanced guidance system computes optimal time-varying trajectories from a dynamic model and a predefined control objective. If a constant set-point is used as input to a low-pass filter (reference model) the outputs of the filter will be smooth time-varying reference trajectories for position, velocity and acceleration (PVA).

3.1.4 Coordinate Frames

When analysing the motion of marine vehicles in 6 DOF it is convenient to define two coordinate frames. The moving coordinate frame $X_0Y_0Z_0$ is conveniently fixed to the vehicle and is called the body-fixed reference frame. The origin 0 of the body-fixed frame is usually chosen to coincide with the centre of gravity (CG) when CG is in the principal plane of symmetry or at any other convenient point if this is not the case.

For marine vehicles, the body axes X_0, Y_0 and Z_0 coincide with the principal axes of inertia, and are usually defined as:

- X_0 - longitudinal axis (directed from aft to fore);
- Y_0 - transverse axis (directed to starboard);
- Z_0 - normal axis (directed from top to bottom).

The motion of the body-fixed frame is described relative to an inertial reference frame. For marine vehicles it is usually assumed that the accelerations of a point on the surface of the Earth can be neglected. Indeed, this is a good approximation since the motion of the Earth hardly affects low speed marine vehicles. As a result of this, an earth-fixed reference frame XYZ can be considered to be inertial. This suggests that the position and orientation of the vehicle should be described relative to the inertial reference frame while the linear and angular velocities of the vehicle should be expressed in the body-fixed coordinate system. The different quantities are defined according to the SNAME notation (1950), as indicated in Table 3.1.

DOF	Motion/Rotation	Forces/ Moments	Linear/ Angular Velocities	Positions/ Euler Angles
1	In x-direction (surge)	X	u	x
2	In y-direction (sway)	Y	u	y
3	In z-direction (heave)	Z	w	z
4	About x-axis (roll)	K	p	ϕ
5	About y-axis (pitch)	M	q	θ
6	About z-axis (yaw)	N	r	ψ

Table 3.1: Notation Used for Marine Vehicles [17]

3.2 Guidance System

Guidance represents a basic methodology concerned with the transient motion behaviour associated with the achievement of motion control objectives [8]. In its simplest form, open-loop guidance systems for marine craft are used to generate a reference trajectory for time-varying trajectory tracking or, alternatively, a path for time-invariant path following. A motion control system will work in close interaction with the guidance system.

There are three type of Guidance Systems:

1. Setpoint regulation (point stabilization) is a special case where the desired position and attitude are chosen to be constant.

2. Trajectory tracking where the objective is to force the system output to track a desired output. The desired trajectory can be computed using reference models generated by low-pass filters, optimization methods. Feasible trajectories can be generated in the presence of both spatial and temporal constraints.

3. Path following where the objective is to follow a predefined path independent of time. No restrictions are placed on the temporal propagation along the path. Spatial constraints can, however, be added to represent obstacles and other positional constraints if they are known in advance.

Tracking control systems can also be designed for target tracking and path tracking. For instance, a target-tracking system tracks the motion of a target that is either stationary (analogous to point stabilization) or that moves such that only its instantaneous motion is known; that is no information about the future target motion is available [8].

As shown in Figure 3.2, the guidance system can use joystick or keyboard inputs, external inputs (weather data, for instance measured wind, wave and current speeds and directions), Earth topological information (digital chart, radar and sonar data), obstacle and collision avoidance data, and finally the state vector, which is available as output from the navigation and sensor systems. The required data are further processed to generate a feasible trajectory for motion control. This can be done using ad hoc techniques or sophisticated methods such as interpolation techniques, dynamic optimization or filtering techniques.A feasible trajectory

means one that is consistent with the craft dynamics.For a linear system, this implies that the eigenvalues of the desired states must be chosen such that the reference model is slower than the craft dynamics.

For a ship or an underwater vehicle, the guidance and control system usually consists of the following subsystems:

- Attitude control system
- Path-following system

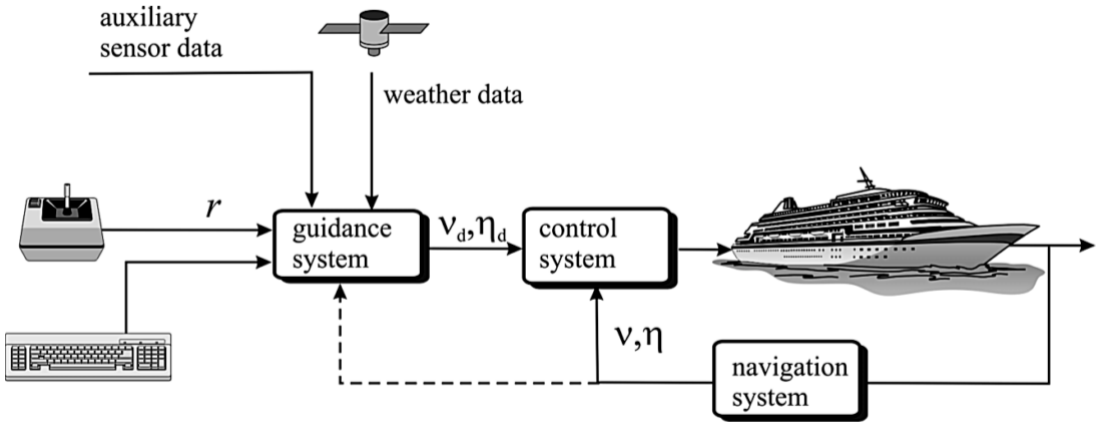

Figure 3.2: In closed-loop guidance (dotted line) the states are fed back to the guidance system while open-loop guidance only uses sensor and reference signal inputs [7].

In its simplest form,the attitude control system is a heading autopilot,while roll and pitch are regulated to zero or left uncontrolled. The main function of the attitude feedback control system is to maintain the craft in a desired attitude on the ordered path by controlling the craft in roll, pitch and yaw. The task of the path-following controller is to keep the craft on the pre described path with some pre defined dynamics, for instance a speed control system by generating orders to the attitude control system. For surface vessels it is common to use a heading controller in combination with a speed controller while aircraft and underwater vehicles also need a height/depth controller. The principles and definitions of guidance, navigation and control have been outlined in Section 3.1.

In the present case, the Path following guidance is chosen to be used. The vessel have to follow predefined path which is determined by a set of way points. Way-point guidance systems which consists of a way-point generator with human interface. The trajectory (path) for the ship can be generated using a batch of way-points which are stored in a suitable database. Commonly, these way-points can be produced in accordance with weather, routing, collision and obstacle avoidance, mission planning, etc.

3.2.1 Path Following

A trajectory describes the motion of a moving object through space as a function of time. The object might be a craft, projectile or a satellite, for example. A trajectory can be described mathematically either by the geometry of the path or as the position of the object over time. Path following is the task of following a pre defined path independent of time; that is there are no temporal constraints. This means that no restrictions are placed on the temporal propagation along the path. Spatial constraints, however, can be added to represent obstacles and other positional constraints. A frequently used method for path following is line-of-sight (LOS) guidance [9]. A LOS vector from the craft to the next waypoint or a point on the path between two waypoints can be used for both course and heading control. If the craft is equipped with a heading autopilot the angle between the LOS vector and the pre described path can be used as a setpoint for the heading autopilot. This will force the craft to track the path.

3.2.1.1 Path Generation based on Waypoints

Systems for waypoint guidance are used both for ships and underwater vehicles. These systems consist of a waypoint generator with a human interface. The selected waypoints are stored in a waypoint database and used for generation of a trajectory or a path for the moving craft to follow. Both trajectory and path-following control systems can be designed for this purpose. Sophisticated features such as weather routing,obstacle avoidance and mission planning can be incorporated in the design of waypoint guidance systems. Some of these features will be discussed in the forthcoming section.

Waypoint Representation

The route of a ship or an underwater vehicle is usually specified in terms of waypoints. Each waypoint is defined using Cartesian coordinates (x_k, y_k, z_k) for $k = 1, ..., n$. The waypoint database therefore consists of:

$$wpt.pos = (x_o, y_o, z_o), (x_1, y_1, z_1), ..., (x_n, y_n, z_n) \tag{3.1}$$

For surface craft, only two coordinates (x_k, y_k) are used. Additionally, other waypoint properties such as speed and heading can be defined, that is

$$wpt.speed = U_0, U_1, ..., U_n \tag{3.2}$$

$$wpt.heading = \psi_0, \psi_1, ..., \psi_n \tag{3.3}$$

For surface craft this means that the craft should pass through waypoint (x_i, y_i) at forward speed U_i with heading angle ψ_i. The three states (x_i, y_i, ψ_i) are also called the pose. The heading angle is usually unspecified during cross-tracking, whereas it is more important during a crab wise manoeuvre close to offshore installations (dynamic positioning).

The waypoint database can be generated using many criteria. These are usually based on:

- **Mission:** The craft should move from some starting point (x_0, y_0, z_0) to the terminal point (x_n, y_n, z_n) via the waypoints (x_i, y_i, z_i).

- **Environmental data:** Information about wind, waves and ocean currents can be used for energy optimal routing (or avoidance of bad weather for safety reasons).
- **Geographical data:** Information about shallow waters and islands should be included.
- **Obstacles:** Floating constructions and other obstacles must be avoided.
- **Collision avoidance:** Avoiding moving craft close to your own route by introducing safety margins.
- **Feasibility:** Each waypoint must be feasible, in that it must be possible to maneuver to the next waypoint without exceeding the maximum speed and turning rate.

Path Generation using Straight Lines and Circular Arcs

In practice it is common to represent the desired path using straight lines and circle arcs to connect the waypoints, as shown in Figure 3.3. This is related to the famous result Dubins (1957), which can be sumarized as:

The shortest path (minimum time) between two configurations (x, y, ψ) *of a craft moving at constant speed U is a path formed by straight lines and circular arc segments.*

Since a craft and not a point mass is considered, the start and end configurations of the craft are specified in terms of the positions (x,y),heading angle ψ and speed U. In addition,it is assumed that there are bounds on the turning rate r or the radius.

In this section, the discussion is limited to Dubins paths formed by straight lines and circles as shown in Figure 3.3, where the inscribed circle between two straight lines describes the desired turn. The radius of the inscribed circle is denoted $\tilde{R}_i (i = 1, ..., n)$. The drawback of this strategy, in comparison with a cubic interpolation strategy, for instance, is that a jump in the desired yaw rate r_d is experienced. This is due to the fact that the desired yaw rate along the straight line is $r_d = 0$ while it is $r_d = constant$ on the circle arc during steady turning. Hence, there will be a jump in the desired yaw rate during transition from the straight line to the circle arc. This produces a small offset during cross-tracking. If a smooth reference trajectory, for instance generated by interpolation, is used, these drawbacks are overcome. However, it is convenient to use straight lines and circle arcs due to their simplicity. Another consideration is that the human operator can specify a circle with radius R_i about each waypoint (see Figure 3.3).

These values are stored in the database as

$$wpt.radious = R_0, R_1, ..., R_n \tag{3.4}$$

The point where the circle arc intersects the straight line represents the turning point of the ship.Hence, the radius of the inscribed circle can be computed from R_i as

$$R_i = R_i tan(\alpha), (i = 1, ..., n) \tag{3.5}$$

where α_i is defined in Figure 3.4.

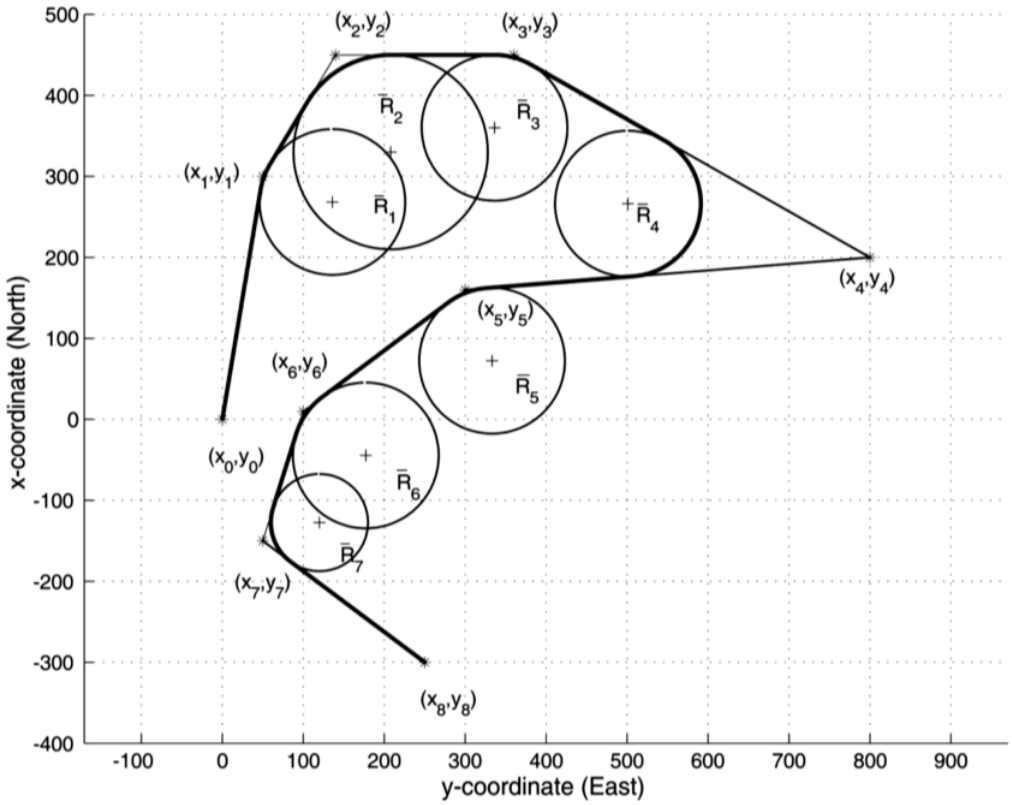

Figure 3.3: Straight lines and inscribed circles used for waypoint guidance [7].

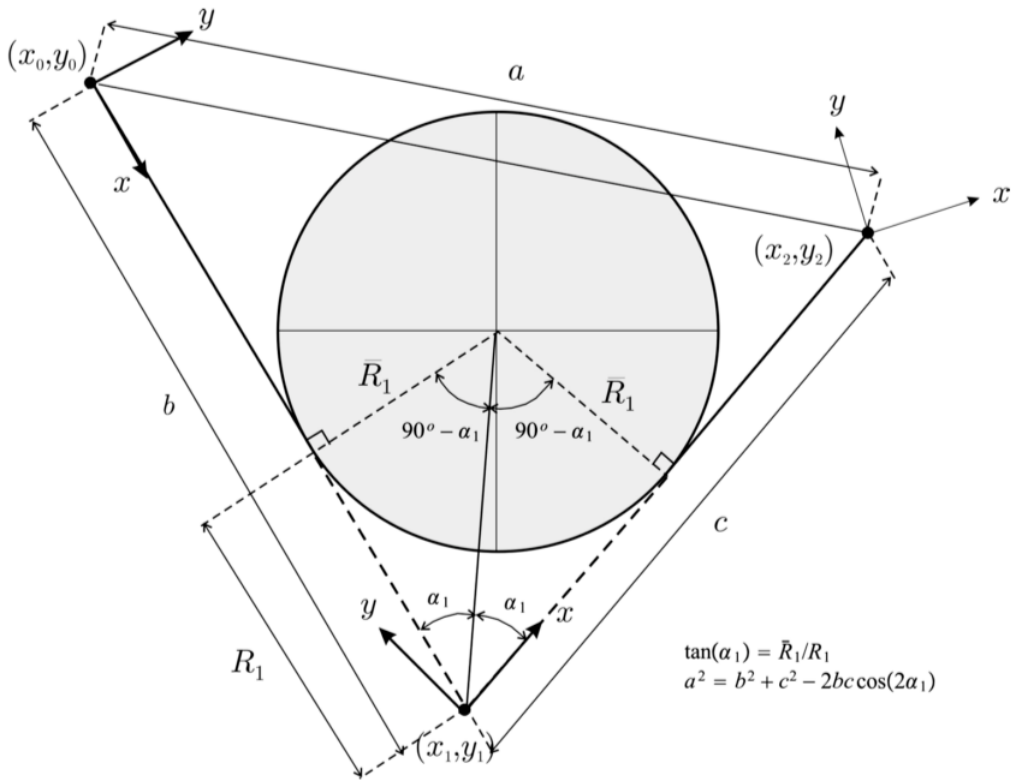

Figure 3.4: Circle with radius $\tilde{R}_i$ inscribed between the points (x_0, y_0), (x_1, y_1) and (x_2, y_2) [7].

LOS Steering Laws

According to Breivik and Fossen [8] for 2-D horizontal plane motions the craft speed is defined as

$$U(t) = u(t) = \sqrt{x(t)^2 + y(t)^2} >= 0 \tag{3.6}$$

while steering is related to the angle

$$\chi(t) = atan2(\dot{y}(t), \dot{x}(t)) \in S := [-\pi, \pi] \tag{3.7}$$

where $atan2(x, y)$ is the four-quadrant version of $arctan(y/x) \in [-\pi/2, \pi/2]$. Path following is ensured by proper assignments to $\chi(t)$ (steering control) as long as $U(t) > 0$ (positive speed) since the scenario only involves a spatial constraint.

Consider a straight-line path implicitly defined by two waypoints ${p_k}^n = [x_k, y_k]^T \in R^2$ and ${p_{k+1}}^n = [x_{k+1}, y_{k+1}]^T \in R^2$. respectively. Also, consider a path-fixed reference frame with origin in ${p_k}^n$ whose x axis has been rotated by a positive angle:

$$\alpha_k = atan2(y_{k+1} - y_k, x_{k+1} - x_k) \in S \tag{3.8}$$

relative to the x axis. Hence, the coordinates of the craft in the path-fixed reference frame can be computed by

$$\varepsilon(t) = R_p(\alpha_k)^T(p^n(t) - {p_k}^n) \tag{3.9}$$

where

$$R_p(\alpha_k) = \begin{bmatrix} cos(\alpha_k) & -sin(\alpha_k) \\ sin(\alpha_k) & cos(\alpha_k) \end{bmatrix} \in SO(2) \tag{3.10}$$

and $\varepsilon(t) = [s(t), e(t)]^T \in R^2$ with

For path-following purposes, only the cross-track error is relevant since $e(t) = 0$ means that the craft has converged to the straight line. Expanding (10.56), the along-track distance and cross-track error can be explicitly stated by

$$s(t) = [x(t) - x_k]cos(\alpha_k) + [y(t) - y_k]sin(\alpha_k) \tag{3.11}$$

$$e(t) = -[x(t) - x_k]sin(\alpha_k) + [y(t) - y_k]cos(\alpha_k) \tag{3.12}$$

and the associated control objective for straight-line path following becomes:

$$\lim_{x \to \infty} e(t) = 0 \tag{3.13}$$

Heading angle commands will be used in order to ensure that $e(t) \to 0$

Enclosure-based steering guidance principle will be used with the purpose of the vessel steer along the LOS vector and to simultaneously stabilize the $e(t)$ to the origin.

Enclosure-Based Steering

Consider a circle with radius $R > 0$ enclosing $p_n = [x, y]^T$. If the circle radius is chosen sufficiently large,the circle will intersect the straight line at two points.The enclosure-based strategy for driving e(t) to zero is then to direct the velocity toward the intersection point $pnlos = [xlos, ylos]^T$ that corresponds to the desired direction of travel, which is implicitly defined by the sequence in which the waypoints are ordered. Such a solution involves directly assigning χ_d as shown in Figure 3.5 [7]. Since

$$tan(\chi_d(t)) = \frac{\Delta y(t)}{\Delta x(t)} = \frac{y_{los} - y(t)}{x_{los} - x(t)} \tag{3.14}$$

the desired course angle can be computed as

$$\chi_d(t) = atan2(y_{los} - y(t), x_{los} - x(t)) \tag{3.15}$$

In order to calculate the two unknowns in p^n_{los}, the following two equations must be solved:

$$[x_{los} - x(t)]^2 + [y_{los} - y(t)]^2 = R^2 \tag{3.16}$$

$$tan(\alpha_k) = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{y_{los} - y_k}{x_{los} - x_k} = constant \tag{3.17}$$

where 3.16 represents the Pythagoras theorem, while 3.17 states that the slope of the line between the two waypoints is constant.

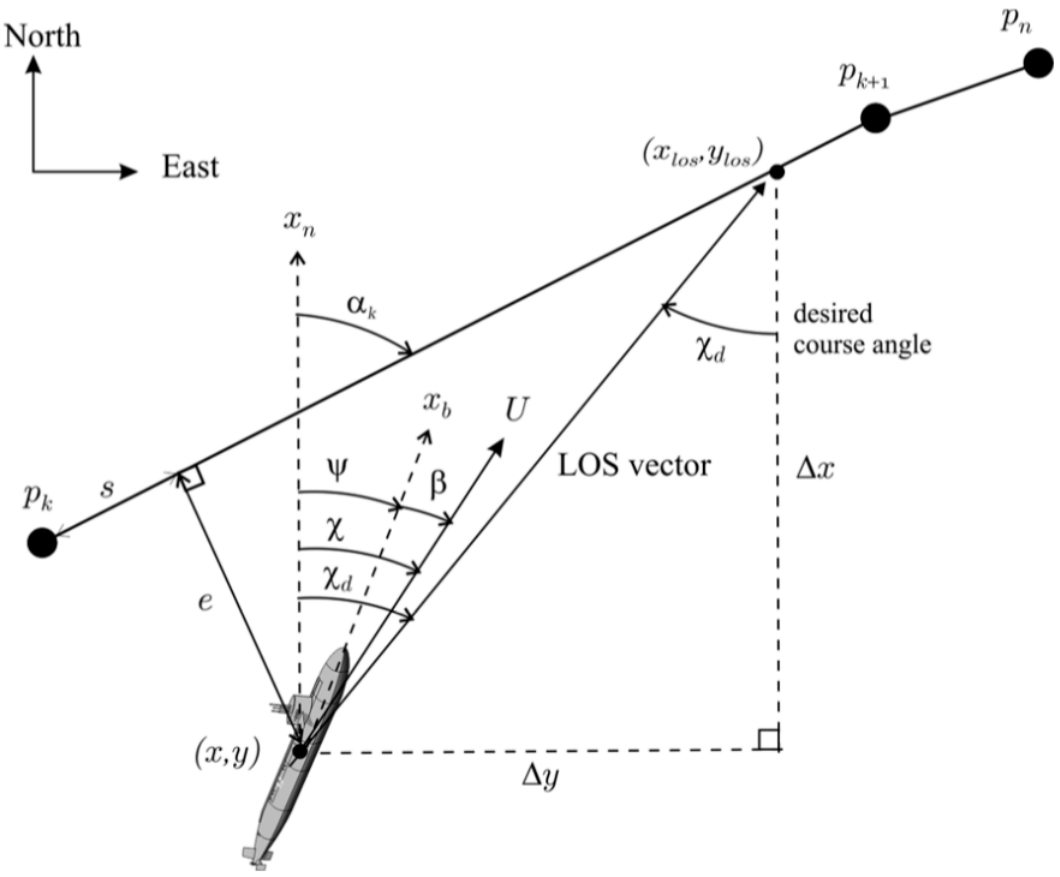

Figure 3.5: LOS guidance where the desired course angle χ_d (angle between x_n and the desired velocity vector) is chosen to point toward the LOS intersection point (x_{los}, y_{los}) [7].

3.3 Navigation System

The navigation system is responsible to estimate the position and in some cases the velocity and acceleration of the vessel. A satellite navigation system is commonly used to determine vessel's attributes (position, velocity, acceleration) in conjunction with motion sensors like accelerometers and gyrocompass. A satellite navigation system with global coverage may be termed a global navigation satellite system (GNSS) and it is a system that uses satellites to provide autonomous geo-spatial positioning. The system can be used for providing position, navigation or for tracking the position of something fitted with a receiver (satellite tracking) [10].

3.3.1 Accelerometer

An accelerometer is a device that measure proper acceleration. Proper acceleration, being the acceleration (or rate of change of velocity) of a body in its own instantaneous rest frame,is

not the same as coordinate acceleration, being the acceleration in a fixed coordinate system. For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity, straight upwards (by definition) of $g = 9.81 m/s^2$. By contrast, accelerometers in free fall (falling toward the center of the Earth at a rate of about $9.81 m/s^2$) will measure zero.

3.3.2 Gyrocompass

A gyrocompass is a type of non-magnetic compass which is based on a fast-spinning disc and the rotation of the Earth (or another planetary body if used elsewhere in the universe) to find geographical direction automatically. The use of a gyrocompass is one of the seven fundamental ways to determine the heading of a vehicle. Although one important component of a gyrocompass is a gyroscope, these are not the same devices; a gyrocompass is built to use the effect of gyroscopic precession, which is a distinctive aspect of the general gyroscopic effect. Gyrocompasses are widely used for navigation on ships, because they have two significant advantages over magnetic compasses:

1. they find true north as determined by the axis of the Earth's rotation, which is different from, and navigationally more useful than, magnetic north, and
2. they are unaffected by ferromagnetic materials, such as in a ship's steel hull, which distort the magnetic field.

3.3.3 GNSS

GNSS stands for Global Navigation Satellite System, and is the standard generic term for satellite navigation systems that provide autonomous geo-spatial positioning with global coverage. This term includes e.g. the GPS, GLONASS, Galileo, Beidou and other regional systems. GNSS is a term used worldwide The advantage to having access to multiple satellites is accuracy, redundancy and availability at all times. Though satellite systems don't often fail, if one fails GNSS receivers can pick up signals from other systems. Also if line of sight is obstructed, having access to multiple satellites is also a benefit. Common GNSS Systems are GPS, GLONASS, Galileo, Beidou and other regional systems.

3.3.4 GPS

The United States' Global Positioning System (GPS) consists of up to 32 medium Earth orbit satellites in six different orbital planes, with the exact number of satellites varying as older satellites are retired and replaced. Operational since 1978 and globally available since 1994, GPS is currently the world's most utilized satellite navigation system.

3.4 Control System

Control design for marine vessels have been an active field of research since the first autopilot was constructed by Elmer Sperry in 1911. Contemporary control systems use a variety of design techniques like PID control, linear quadratic optimal and stochastic control, fuzzy systems,

neural networks and non-linear control theory [7]. Motion control is the action of determining the necessary control forces and moments to be provided by the craft in order to satisfy a certain control objective. The desired control objective is usually seen in conjunction with the guidance system. Examples of control objectives are minimum energy, setpoint regulation, trajectory-tracking, path-following and maneuvering control. Constructing the control algorithm involves the design of feedback and feedforward control laws. The outputs from the navigation system, position, velocity and acceleration are used for feedback control while feedforward control is implemented using signals available in the guidance system and other external sensors.

In this thesis the models ,that they will be described in Chapter 6, are going to be controlled via a neural network autopilot. In this execution, the vessel has only one control surface, the rudder. The neural network controller is going to provide the appropriate command on the rudder to lead the vessel to the reference path. The main areas of application of neural networks for control of processes are identification, optimization, cancellation of non linearities and adaptive control of complex processes with variable and non-stationary parameters [4]. The advantage of artificial neural networks is the simple realization of complex logic functions and algorithms for control. For easy use of neural networks a language standard for configuration and training of neural networks and a specialized microchip for embedding in the configuration of programmable logic controllers is developed.

3.4.1 The Architecture

The idea behind the autopilot based on Neural Networks is to replace a simple controller like PID with a Neural Network controller (Figure 3.6).

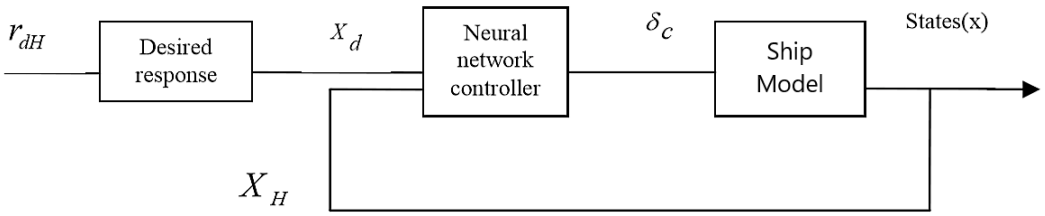

Figure 3.6: Autopilot based on Neural Networks [7]

A Neural Network controller is a batch of neurons which are trained to produce the desired result. Figure 3.7 shows a neural network with one hidden layer which produces the commanded rudder angle in accordance with a set of inputs [11].

3.4.2 The Concept

Neural networks have the ability to memorize a large data size, so they can be used to "learn" reality and they calculate the commanded rudder angle depending on each situation [11]. The controller have to produce the rudder angle which will be imported in actuator so the vessel follow the desired path. The desired path is predetermined and the controller continuously calculates the error between the actual and the desired course as illustrated in Figure 3.8. This error constitutes one of the inputs which are inserted in neural network (controller). The main

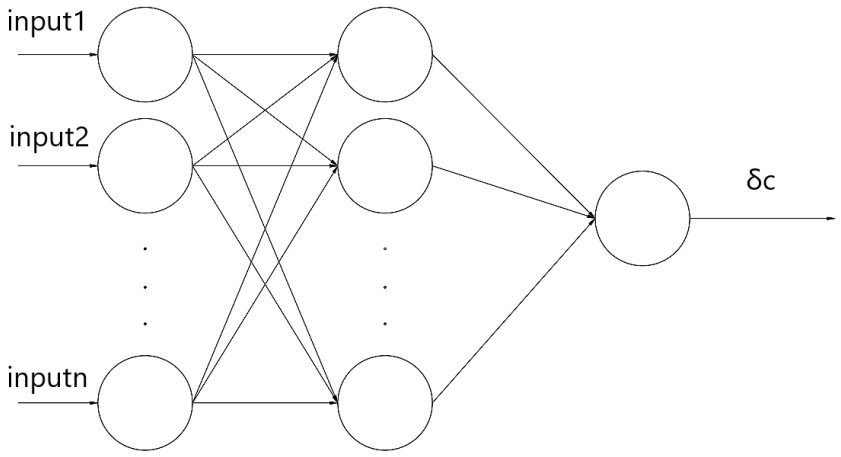

Figure 3.7: Neural Network Architecture

purpose is to minimize this error and if possible be zero. The desired course is calculated by the Guidance system in accordance with the predetermined path.

The biggest challenge is to find a way in which the neural network imitates the nature (sea, wind, etc.) in order to be able to produce the appropriate command and it lead the vessel in the desired route. This can be achieved by performing a batch of simulations for differences situations that cover the whole range of the vessel operation. Each situation consists of the error between the actual and desired heading and yaw velocity. This scheme is used when the mathematical model is Nomoto. When the model is the 4DOF, the shape used consist of the above states and also the . A set of simulations is performed and the produced data is exported for the neural network training.

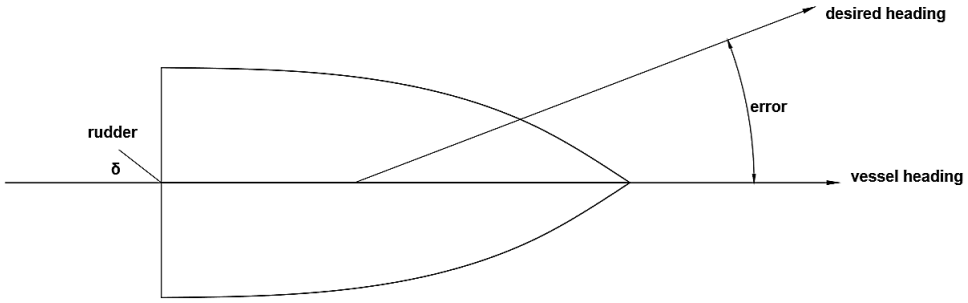

Figure 3.8: Vessel Heading

3.4.3 The Training

As described above, the data to be used for the neural network training is produced by conducting a set of simulations. Supervised learning was selected for the neural network training as the goal is to train the neural network to imitates the nature. The neural network will be trained to map input to a continuous output so the supervised learning type is called Regression [12].

3.4.3.1 Supervised Learning Training

In supervised training, both the inputs and the outputs (data) are provided from simulations as described. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked. The set of data which enables the training is called the "training set." During the training of a network the same set of data is processed many times as the connection weights are ever refined [13].

The current commercial network development packages provide tools to monitor how well an artificial neural network is converging on the ability to predict the right answer. These tools allow the training process to go on for days, stopping only when the system reaches some statistically desired point, or accuracy. However, some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also don't converge if there is not enough data to enable complete learning. Ideally, there should be enough data so that part of the data can be held back as a test. Many layered networks with multiple nodes are capable of memorizing data. To monitor the network to determine if the system is simply memorizing its data in some nonsignificant way, supervised training needs to hold back a set of data to be used to test the system after it has undergone its training. (Note: memorization is avoided by not having too many processing elements.)

If a network simply can't solve the problem, the designer then has to review the input and outputs, the number of layers, the number of elements per layer, the connections between the layers, the summation, transfer, and training functions, and even the initial weights themselves. Those changes required to create a successful network constitute a process wherein the "art" of neural networking occurs. Another part of the designer's creativity governs the rules of training. There are many laws (algorithms) used to implement the adaptive feedback required to adjust the weights during training. The most common technique is backward-error propagation, more commonly known as back-propagation. These various learning techniques are explored in greater depth later in this report.

Yet, training is not just a technique. It involves a "feel," and conscious analysis, to insure that the network is not overtrained. Initially, an artificial neural network configures itself with the general statistical trends of the data. Later, it continues to "learn" about other aspects of the data which may be spurious from a general viewpoint.

When finally the system has been correctly trained, and no further learning is needed, the weights can, if desired, be "frozen." In some systems this finalized network is then turned into hardware so that it can be fast. Other systems don't lock themselves in but continue to learn while in production use. In Figure 3.9 illustrated the Supervised training process.

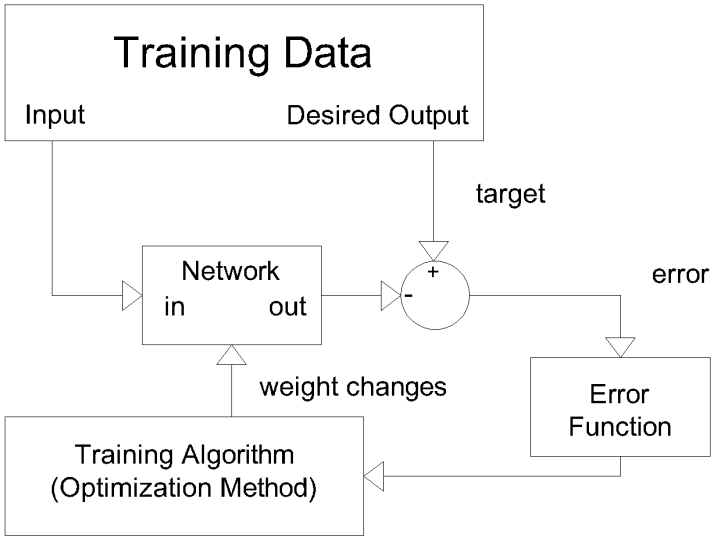

Figure 3.9: Supervised Training Process

Chapter 4

Programming Tools for Machine Learning

4.1 Python (Programming Language)

Python is a general-purpose programming language [14]. It has been around for quite a while: Guido van Rossum, Python's creator, started developing Python back in 1990. This stable and mature language is very high-level, dynamic, object-oriented, and cross-platform, all characteristics that are very attractive to developers. Python runs on all major hardware platforms and operating systems, so it doesn't constrain your platform choices (Figure 4.1). Python offers high productivity for all phases of the software life cycle: analysis, design, prototyping, coding, testing, debugging, tuning, documentation, deployment, and, of course, maintenance. Python's popularity has seen steady, unflagging growth over the years. Today, familiarity with Python is an advantage for every programmer, as Python has infiltrated every niche and has useful roles to play as a part of any software solution. Python provides a unique mix of elegance, simplicity, practicality, and power.

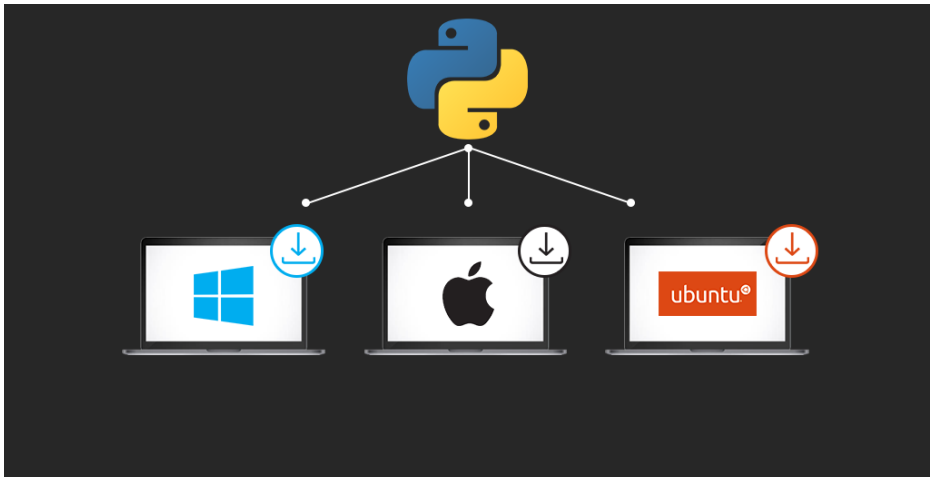

Figure 4.1: Python Platforms Compatibility [14].

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

There is more to Python programming than just the Python language [15]: the standard Python library and other extension modules are almost as important for effective Python use as the language itself. The Python standard library supplies many well-designed, solid, 100 percent pure Python modules for convenient reuse. Library is a collection of functions and methods that allows you to perform many actions without writing your code. It includes modules for such tasks as representing data, string and text processing, interacting with the operating system and filesystem, and web programming. Because these modules are written in Python, they work on all platforms supported by Python.

The most important advantages of Python language are enumerated in the following list:

1. Simple and easy-to-understand syntax.
2. Object Oriented Programming-driven.
3. Supports imperative and functional programming.
4. Extensive library.
5. Supports multiple platforms (Web and mobile computing).
6. Free and Open Source with large community support.

Python has undeniably become one of the most adapted programming languages in recent times. Simple and readable coding lines being the integral reason for the rise, Python has been widely used in every platform. From building a website, GUI Application, Smartphone apps to gaming, the influence of Python has been monumental and is rapidly strengthening its base. It has come a long way from 1989; and from the growth ratio witnessed, Python's chart shows no negative inclination.

4.2 Tensorflow (Python Library)

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks [1]. It is a symbolic math library, and is also used for large-scale machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions

can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications. The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

The architecture for TensorFlow looks like the Figure 4.2.

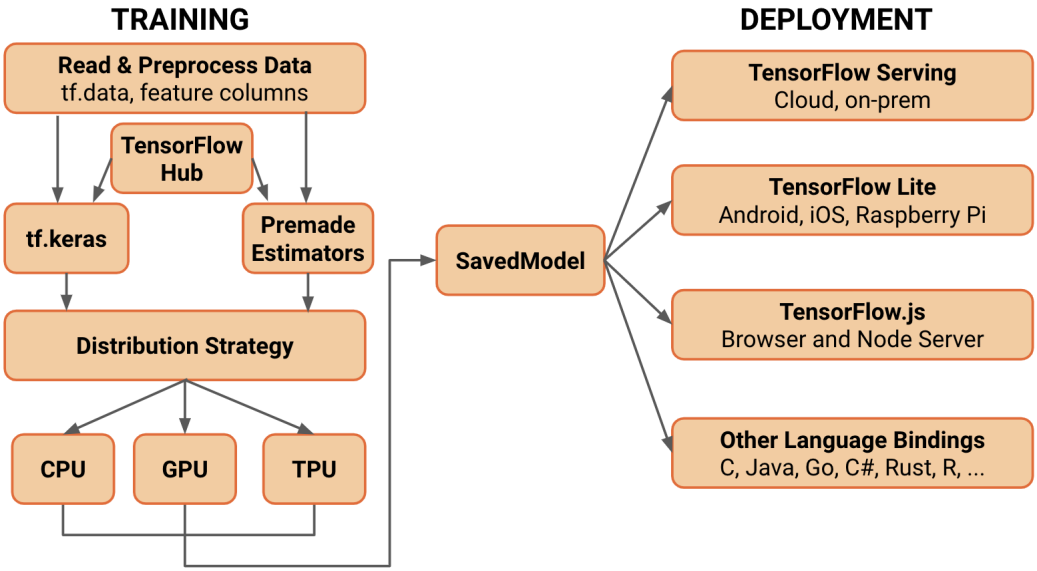

Figure 4.2: Tensorflow Architecture [1]

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

A Tensorflow example is illustrated in the following picture. This example shows how you can define variables (e.g. W and b) as well as variables that are the result of computation (y).

4.3 Keras (Python Library)

Keras is an open-source neural-network library written in Python [16]. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.[10] It also allows use of distributed training of deep-learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU).

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was "designed for human beings, not machines," and "follows best practices for reducing cognitive load."

Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

Chapter 5

Ship Models

A mathematical model for the ship is required because there is no physical model of the ship. With the mathematical model, the whole process of thesis can be executed via computer. This mathematical model has to corresponds to a physical-scale ship.

5.1 Nomoto Model

The Nomoto is one of the simplest model representation of a ship. Firstly we will use this simplest model to consider if the concept is feasible. Then we will use a more complicated model to represent the reality with greater accuracy. For our study we will use the First-order Nomoto model as it is simpler. This model are provided by the Second-order Nomoto model, so both will be described [7].

5.1.1 Second-Order Nomoto Model

A linear autopilot model for heading control can be derived from the below simplify maneuvering model:

$$M\dot{v} + N(u_o)v = b\delta \tag{5.1}$$

by choosing the yaw rate r as output:

$$r = c^T v, \quad c^T = [0, 1] \tag{5.2}$$

Hence, application of the Laplace transformation yields

$$\frac{r}{\delta}(s) = \frac{K(1 + T_3 s)}{(1 + T_1 s)(1 + T_2 s)} \tag{5.3}$$

A similar expression is obtained for the sway motion:

$$\frac{u}{\delta}(s) = \frac{K_u(1 + T_u s)}{(1 + T_1 s)(1 + T_2 s)} \tag{5.4}$$

where K_u and T_u differ from K and T_3 in the yaw equations. The equation (5.3) is referred to as Nomoto's second-order model.

The time-domain representation for Nomoto's second-order model becomes

$$T_1 T_2 \psi^{(3)} + (T_1 + T_2)\ddot{\psi} + \dot{\psi} = K(\delta + T_3\dot{\delta}) \tag{5.5}$$

5.1.2 First-Order Nomoto Model

The first-order Nomoto model is obtained by defining the equivalent time constant:

$$T := T_1 + T_2 - T_3 \tag{5.6}$$

such that

$$\frac{r}{\delta}(s) = \frac{K}{(1 + T_s)} \tag{5.7}$$

Finally, $\dot{\psi} = r$ yields

$$\frac{\psi}{\delta}(s) = \frac{K}{s(1 + T_s)} \tag{5.8}$$

which is the transfer function that is used in most commercial autopilot systems.
The time-domain representation for Nomoto's first-order model becomes

$$T\ddot{\psi} + \dot{\psi} = K\delta \tag{5.9}$$

5.1.3 First-Order Nomoto Model Implementation

The coefficients of the nomoto model will be obtained from remote controlled vessel (as illustrated on Figure 5.1) which has been produced by the university team OCEANOS in account for the annually competition Hydrocontest. The vessel's main particulars and first order nomoto coefficients are summarize in Table 5.1.

Quantity	Value	Unit
L	3.4	m
Vs	3.2	m/s
K	3.47	s^{-1}
T	3.94	s

Table 5.1: Vessel's Main Particulars and Coefficients

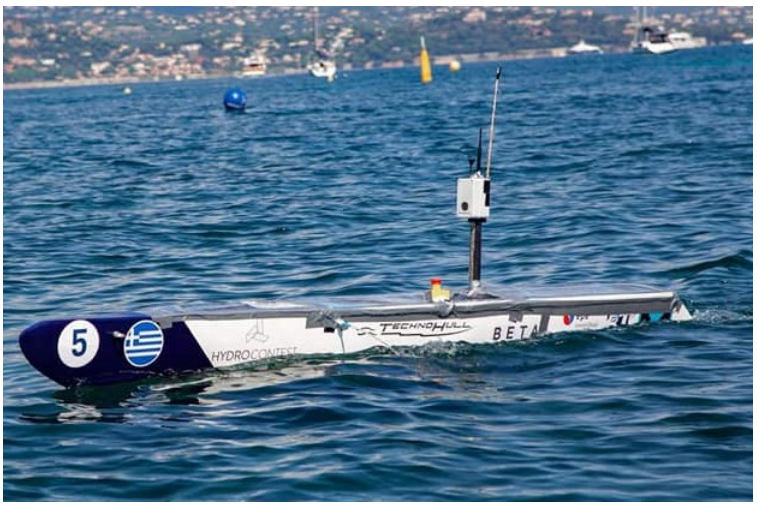

Figure 5.1: University team OCEANOS vessel.

5.2 4 DOF

5.2.1 6DOF Motion Equations

In accordance with the Newtonian mechanics, the rigid-body kinetics can be derived to the below nonlinear equations of 6DOF motion [17].

$$\begin{aligned}
m[\dot{u} - vr + wq - x_g(q^2 + r^2) + y_g(pq - \dot{r}) + z_g(pr + \dot{q})] &= X \\
m[\dot{v} - wp + ur - y_g(r^2 + p^2) + z_g(qr - \dot{p}) + x_g(qp + \dot{r})] &= Y \\
m[\dot{w} - uq + vp - z_g(p^2 + q^2) + x_g(rp - \dot{q}) + y_g(rq + \dot{p})] &= Z \\
I_x\dot{p} + (I_z - I_y)qr - (\dot{r} + pq)I_{xz} + (r^2 - q^2)I_{yz} + (pr - \dot{q})I_{xy} & \\
+m[y_g(\dot{w} - uq + vp) - z_g(\dot{v} - wp + ur)] &= K \\
I_y\dot{q} + (I_x - I_z)rp - (\dot{p} + qr)I_{xy} + (p^2 - r^2)I_{zx} + (qp - \dot{r})I_{yz} & \\
+m[z_g(\dot{u} - vr + wq) - x_g(\dot{w} - uq + vp)] &= M \\
I_z\dot{r} + (I_y - I_x)pq - (\dot{q} + rp)I_{yz} + (q^2 - p^2)I_{xy} + (rq - \dot{p})I_{zx} & \\
+m[x_g(\dot{u} - wp + ur) - y_g(\dot{u} - vr + wq)] &= N
\end{aligned} \tag{5.10}$$

The first three equations represent the translational motion, while the last three equations represent the rotational motion.

5.2.2 4 DOF Motion Equations

The study concerns the maneuvering control of the ship, so a mathematical model describing the motions of ship in 3 DOF (surge, sway and yaw) is needed. Sometimes roll is augmented to the horizontal plane model to describe more accurately the coupled lateral motions, that is sway–roll–yaw couplings while surge is left decoupled. In maneuvering theory, frequency-dependent added mass and potential damping are approximated by constant values and thus it is not necessary to compute the fluid-memory effects. The main results of these calculations are based on the assumptions that the hydrodynamic forces and moments can be approximated at one frequency of oscillation such that the fluid-memory effects can be neglected. The results is a nonlinear mass-damper-spring system with constant coefficients [18].

The Figure 5.2 illustrate the vessel motion on horizontal plane.

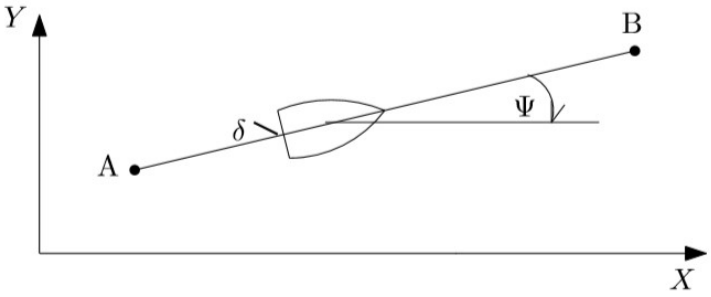

Figure 5.2: Vessel Motion on Horizontal Plane [17].

The equations of motion are formulated at the origin of the body-fixed frame (b-frame), which has coordinates

$$o_b = \begin{bmatrix} L_{pp}/2 \\ 0 \\ T \end{bmatrix} \tag{5.11}$$

with respect to the intersection of the aft per-pendicular (AP), the centerline, and the base line (BL), as illustrated on Figure 5.3.

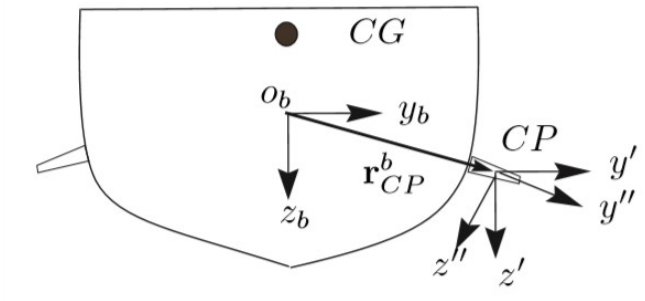

Figure 5.3: References Frames used to compute fin forces [17].

The rigid-body equations of motion in 4-DOF are provided if unnecessary motions are omitted from the 6DOF equations.

$$\begin{aligned} m[\dot{u} - {y_g}^b\dot{r} - vr - {x_g}^b r^2 + {z_g}^b pr] &= \tau_1 \\ m[\dot{u} - {z_g}^b\dot{p} + {x_g}^b\dot{r} + ur - {y_g}^b(r^2 + p^2)] &= \tau_2 \\ I_{xx}\dot{p} - m{z_g}^b\dot{v} + m[{y_g}^b vp - {z_g}^b ur] &= \tau_4 \\ I_{zz}\dot{r} + m{x_g}^b\dot{v} - m{y_g}^b\dot{u} + m[{x_g}^b ur + {y_g}^b vr] &= \tau_6 \end{aligned} \tag{5.12}$$

where all the coordinates are body-fixed coordinates, and the position of the centre of gravity (CG) with respect to the body-fixed frame is denoted by

$${r_g}^b = \begin{bmatrix} {x_g}^b \\ {y_g}^b \\ {x_g}^b \end{bmatrix} = \begin{bmatrix} LCG - L_{pp}/2 \\ 0 \\ T - VCG \end{bmatrix} \tag{5.13}$$

The lower script indicates that these are coordinates of the CG, with respect to the b-frame, and the upper script indicates that the vector is expressed in the b-frame.

The vector of forces on the right hand side of (1), can be separated into the following components:

$$\tau = \tau_{hyd} + \tau_c + \tau_p + \tau_w \tag{5.14}$$

where subscripts stand for hydrodynamic and hydrostatic forces, control device forces, propulsion forces, and wave excitation forces.

5.2.2.1 Hydrodynamic forces

The hydrodynamic forces considered in this section are those due to the motion of the vessel in calm water. These forces are modelled as series expansions of the velocities and the roll angle for the restoring terms. The following terms are considered:

Surge Terms

$$\tau_{1hyd} = X_{\dot{u}}\dot{u} + X_{vr}vr + X_{u|u|}u|u| \tag{5.15}$$

Sway Terms

$$\begin{aligned} \tau_{2hyd} = Y_{\dot{v}}\dot{v} + Y_{\dot{r}}\dot{r} + Y_{\dot{p}}\dot{p} + Y_{|u|v}|u|v + Y_{ur}ur + Y_{v|v|}v|v| + Y_{u|r|}u|r| + Y_{r|v|}r|v| + Y_{\phi|uv|}\phi|uv| + Y_{\phi|ur|}\phi|ur| \\ + Y_{\phi uu}\phi u^2 \end{aligned} \tag{5.16}$$

Roll Terms

$$\begin{aligned} \tau_{4hyd} = K_{\dot{v}}\dot{v} + K_{\dot{p}}\dot{p} + K_{|u|v}|u|v + K_{ur}ur + K_{|v|v}|v|v + K_{v|r|}v|r| + K_{r|v|}r|v| + K_{\phi|uv|}\phi|uv| + K_{\phi|ur|}\phi|ur| \\ + K_{\phi uu}\phi u^2 + K_{|u|p}|u|p \end{aligned} \tag{5.17}$$

Yaw Terms

$$\tau_{6hyd} = N_{\dot{v}}\dot{v} + N_{\dot{r}}\dot{r} + N_{|u|v}|u|v + +N_{|u|r}|u|r + N_{r|r|}r|r| + N_{r|v|}r|v| + N_{\phi|uv|}\phi|uv| + N_{\phi u|r|}\phi u|r| + N_p p + N_{|p|p}|p|p + N_{|u|p}|u|p + N_{\phi u|u|}\phi u|u| \quad (5.18)$$

The numerical values of the coefficients are included in Table A.

5.2.2.2 Propulsion forces

The dynamic of the propulsion system are not included in the model. Rather, it is assumed that the propellers produce a constant thrust T that compensates for the calm water resistance.

$$T = -X_u u_{nom} - X_{u|u|}u_{nom}^2 \quad (5.19)$$

here the service speed is $u_{nom} = 7.71m/s$ (15kt). Then the motion of the rudders and fins produce drag forces that slow down the vessel. Similarly, the action of the waves produce small deviations from the service speed.

5.2.2.3 Control Forces: Rudder and Fins

The vessel model is equipped with two rudders and one pair of stabilizer fins. The lift and drag of the hydrofoils, from [19], are calculated according to

$$L = \frac{1}{2}\rho {V_f}^2 A_f \overline{C}_L \alpha_e \quad (5.20)$$

$$D = \frac{1}{2}\rho {V_f}^2 A_f \left(C_{D0} + \frac{(\overline{C}_L \alpha_e)^2}{0.9\pi\alpha} \right) \quad (5.21)$$

where V_f is the local velocity at the foil, A_f is the area of the foil, α_e is the effective angle of attack in radians, and α is the effective aspect ratio. We use the linear approximation for the lift coefficient:

$$\overline{C}_L = \left. \frac{\partial C_L}{\partial \alpha_e} \right|_{a_e=0} \quad (5.22)$$

Once the stall angle of the hydrofoils is reached, the lift saturates in value. Table A shows the free-stream data for the rudder and fin profiles - this data was also adopted from [19].

To calculate the lift of the rudder, the effective angle of attack is approximated by the mechanical angle of the rudder: $\alpha_e = \alpha_r$, and the local flow velocity at the rudder is considered to be equal to the vessel's total speed, i.e., $V_f = \sqrt{u^2 + v^2}$. Then, a global correction for the lift and drag is used to account for the rudder-propeller interaction [20]

$$\Delta L = T(1 + \frac{1}{\sqrt{1 + C_{Th}}}) sin\alpha_e \quad (5.23)$$

$$\Delta D = T(1 + \frac{1}{\sqrt{1 + C_{Th}}}))(1 - cos\alpha_e) \quad (5.24)$$

where T is the propeller thrust, and C_{Th} is the propeller loading coefficient:

$$C_{Th} = \frac{2T}{\rho {V_f}^2 A_p} \tag{5.25}$$

in which A_p is the propeller disc area. The propeller diameter is 1.6 m.
The forces generated by the rudder in body-fixed frames are then approximated by:

$$\tau_{1rudder} \approx -D \tag{5.26}$$

$$\tau_{2rudder} \approx L \tag{5.27}$$

$$\tau_{4rudder} \approx z^b_{CP} L \tag{5.28}$$

$$\tau_{6rudder} \approx x^b_{CP} L \tag{5.29}$$

where x^b_{CP} and z^b_{CP} are the coordinates of the center of pressure of the rudder (CP) with respect to the b-frame. The center of pressure is assumed to be located at the rudder stock and half the rudder span. The rudder data is shown in Table.

For the stabilizer fins, the center of pressure is located halfway along the span of the fin. The coordinates of the center of pressure with respect to the b-frame are given by the vector r^b_{CP} see Figure 1.

To calculate the forces of the fins, the velocities in the b-frame are expressed in the frame x', y', z', which is located at the CP for the fin. These velocities are then rotated by the tilt angle of the fin, γ, expressing them in the frame x", y", z". This frame is used to calculate the angle of attack of the fin, and thus calculate the forces and moments generated. The mechanical angle of the fin is defined using the right hand screw rule along the y" axis: a positive angle means leading edge up: trailing edge down.

5.2.2.4 Wave excitation forces

The wave excitation forces are simulated as a multisine time series. This uses the force frequency response function (FRF) of the vessel in combination with the wave spectrum. The force-FRF were computed using ShipX VERES for the service speed and at intervals of 10 deg of encounter angle.

The sea surface elevation is considered as a realization of a random process characterized in terms of a directional sea spectrum $S_{\zeta\zeta}$. The dominant wave propagating direction is defined in the North-East frame with propagation angle positive clockwise; that is, if the dominant direction is 0 deg, the waves travel towards north, and if the dominant direction is deg, the waves travel towards the N-E.

The wave dominant direction and the vessel heading are used to find the encounter angle χ between the vessel and the waves. The following convention is adopted:

- $\chi = 0$ deg following seas
- $\chi = 90$ deg beam seas from port
- $\chi = 180$ deg head seas.

The calculation of the forces uses interpolation with a smooth switching of the encounter angle and the speed. The following formula are the basis to calculate the forces in the different DOF:

$$\tau_{wi}(t) = \sum_{n=1}^{N} \sum_{m=1}^{M} \overline{\tau}_{inm} cos[\omega_{enm} t + \phi_{inm}] \tag{5.30}$$

for i=1,2,4,6 with

$$\omega_{enm} = \omega_n - \frac{{\omega_n}^2 U}{g} cos(\chi_m) \tag{5.31}$$

$$\phi_{inm} = arg H_i(\omega_n^*, U, \chi_m^*) + \epsilon_n \tag{5.32}$$

$$\tau_{inm} = \sqrt{2|H_i(\omega_n^*, U, \chi_m^*)|^2 S_{\zeta\zeta}(\omega_n^*, \chi_m^*) \Delta\chi \Delta\omega} \tag{5.33}$$

where H_i are the force FRF of the vessel, and ω_n^* and χ_m^* are chosen randomly in the intervals

$$\left[\omega_n - \frac{\Delta\omega}{2}, \omega_n + \frac{\Delta\omega}{2}\right], \left[\chi_m - \frac{\Delta\chi}{2}, \chi_m + \frac{\Delta\chi}{2}\right] \tag{5.34}$$

5.2.3 4 DOF Model Implementation

The 4DOF mathematical model corresponds to a 50 metres long, fast monohull coastal patrol vessel, the vessel profile is illustrated on Figure 5.4 . This vessel is a Navy coastal patrol vessel, of a design by ADI-Limited Australia, but was never put into production.

The above mathematical model is based on the drawing lines and load conditions provided by ADI-Limited. The design evolved from a Danish vessel, for which a preliminary set of manoeuring coefficients was published by [2]. The vessel's main particulars and the loading condition are summarized in Table 5.2 [17].

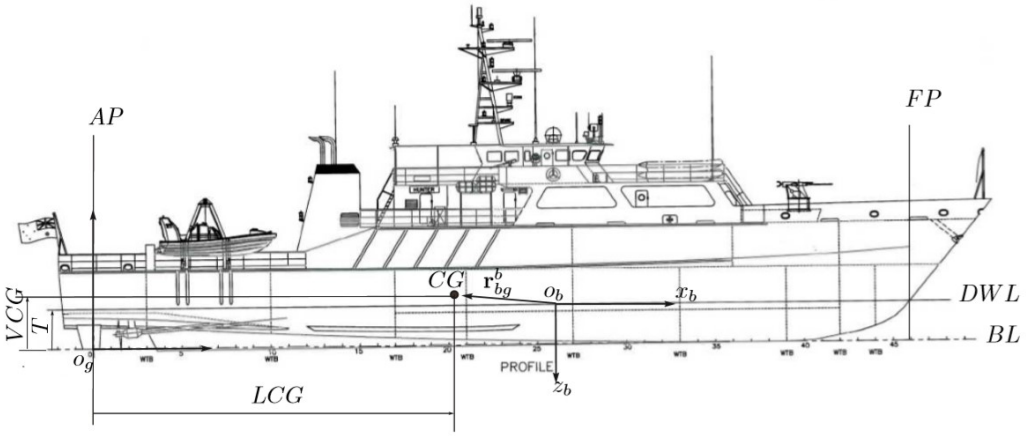

Figure 5.4: Vessel profile and reference frames [2].

Quantity	**Symbol**	**Full Load**	**Unit**
Length between perpendiculars	L_{pp}	51.5	m
Length over all	LOA	52.5	m
Beam over all	BOA	8.6	m
Nominal speed	U	15	kt
Draft at $Lpp/2$ at Design Waterline	T	2.29	m
Length Water Line	L_{WL}	47.702	m
Breadth Water Line	B_{WL}	7.726	m
Trim +ve aft	t	-0.06	m
Displacement mass	Δ	364.78 $\times 10^3$	kg
Displacement volume	∇	355.88 $\times 10^3$	m^3
Lateral Centre of Gravity from AP	LCG	19.82	m
Vertical Centre of Gravity from BL	VCG	3.36	m
Lateral Centre of Buoyancy from AP	LCB	19.82	m
Vertical Centre of Buoyancy from BL	VCB	1.549	m
Lateral Centre of Flotation from AP	LCF	18.27	m
Transverse Metacenter above keel	KMt	4.828	m
Longitudinal Metacenter above keel	KMl	124.15	m
Transverse Metacentric Height	GMt	1.0	m
Longitudinal Metacentric Height	GMl	113.99	m
Transverse Buoyancy to Metacentre	BMt	3.34	m
Longitudinal Buoyancy to Metacentre	BMl	114.49	m
origin of body-fixed frame	o_b	$[Lpp/2, 0, T]$	measured from AP and BL
Roll gyradius (about o_b)	k_4	3.053	m
yaw gyradius (about o_b)	k_6	9.591	m
Inertia Roll (about o_b)	I_{44}	3.4263×10^6	kg m^2
Inertia yaw (about o_b)	I_{66}	3.3818×10^7	kg m^2
Natural Roll Freq.	ω_ϕ	0.93	rad/s
Natural Roll Period	T_ϕ	6.76	s

Table 5.2: Main particulars and loading conditions [2].

X-Coefficients	N-Coefficients	K-Coefficients	Y-Coefficients
$X_{\dot{u}} = -17400$	$N_{\dot{v}}$=538000	$K_{\dot{u}}$=296000	$Y_{\dot{v}} = -1.9022 \times 10^6$
$X_{u\|u\|} = -1960$	$N_{\dot{r}} = -4.3958 \times 10^7$	$K_{\dot{r}}$= 0.0	$Y_{\dot{r}} = -1.4 \times 10^6$
X_{vr}= 0.33 $\times$ m	$N_{\dot{p}}$=0.0	$K_{\dot{p}} = -0.674 \times 10^6$	$Y_{\dot{p}} = -0.296 \times 10^6$
	$N_{\|u\|v} = -92000$	$K_{\|u\|v} = 9260$	$Y_{\|u\|v} = -11800$
	$N_{\|u\|r} = -4.71 \times 10^6$	$K_{ur} = -102000$	$Y_{ur} = 131000$
	$N_{v\|v\|} = 0.0$	$K_{v\|v\|} = 29300$	$Y_{v\|v\|} = -3700$
	$N_{r\|r\|} = -202 \times 10^6$	$K_{r\|r\|} = 0.0$	$Y_{r\|r\|} = 0.0$
	$N_{v\|r\|} = 0.0$	$K_{v\|r\|} = 0.621 \times 10^6$	$Y_{v\|r\|} = -0.794 \times 10^6$
	$N_{r\|v\|} = -15.6 \times 10^6$	$K_{r\|v\|} = 0.142 \times 10^6$	$Y_{r\|v\|} = -0.182 \times 10^6$
	$N_{\phi\|uv\|} = -0.214 \times 10^6$	$K_{\phi\|uv\|} = -8400$	$Y_{\phi\|uv\|} = 10800$
	$N_{\phi u\|r\|} = -4.98 \times 10^6$	$K_{\phi\|ur\|} = -0.196 \times 10^6$	$Y_{\phi\|ur\|} = 0.251 \times 10^6$
	$N_{\phi u\|u\|} = -8000$	$K_{\phi uu} = -1180$	$Y_{\phi uu} = -74$
	$N_{\|u\|p} = 0.0$	$K_{\|u\|p} = -15500$	$Y_{\|u\|p} = 0.0$
	$N_{p\|p\|} = 0.0$	$K_{p\|p\|} = -0.416 \times 10^6$	$Y_{p\|p\|} = 0.0$
	$N_p = 0.0$	$K_p = -0.5 \times 10^6$	$Y_p = 0.0$
	$N_\phi = 0.0$	$K_{\phi\phi\phi} = -0.325\rho g \nabla$	$Y_\phi = 0.0$
	$N_{\phi\phi\phi} = 0.0$		$Y_{\phi\phi\phi} = 0.0$
			$Y_{\delta uu} = 2 \times 3.5044 \times 10^3$

Table 5.3: Manoeuvring coefficients [2].

Chapter 6

Methods and Results

In this chapter, we will describe the way using for the data collection, the neural network training and the controller construction and we will illustrate the results of some representative simulations in order to confirm the efficiency and effectiveness of neural network controller.

6.1 Nomoto Model Implementation

As described on chapter 5.1 the First-Order Nomoto model is:

$$\frac{\psi}{\delta}(s) = \frac{K}{s(1+T_s)} \tag{6.1}$$

and the time-domain representation is:

$$T\ddot{\psi} + \dot{\psi} = K\delta \tag{6.2}$$

Where $\dot{\psi} = r$.

Thus, the Nomoto model equations have only one independent and two dependent variables as illustrated in Table 6.1.

Variables	Quantity
Independent	δ
Dependent	ψ, r

Table 6.1: Nomoto Model Independent and Dependent Variable

Based on the mentioned formulas (6.1, 6.2), we will create (as described in the next chapters) a ship controller in order to the vessel will follow a desired heading or a predefined path.

6.1.1 Data Collection

The first step in implementing the project is to produce the appropriate data to be used for the training of neural network. According to the Table 6.1, there are one independent and two dependent variables, so for different initial heading, yaw velocity (ψ_o, ρ_o) and rudder command (δ) will obtain the values for the next state (ψ, ρ) after $dt = 0.01s$. In Figure 6.1 depict the data producing flow.

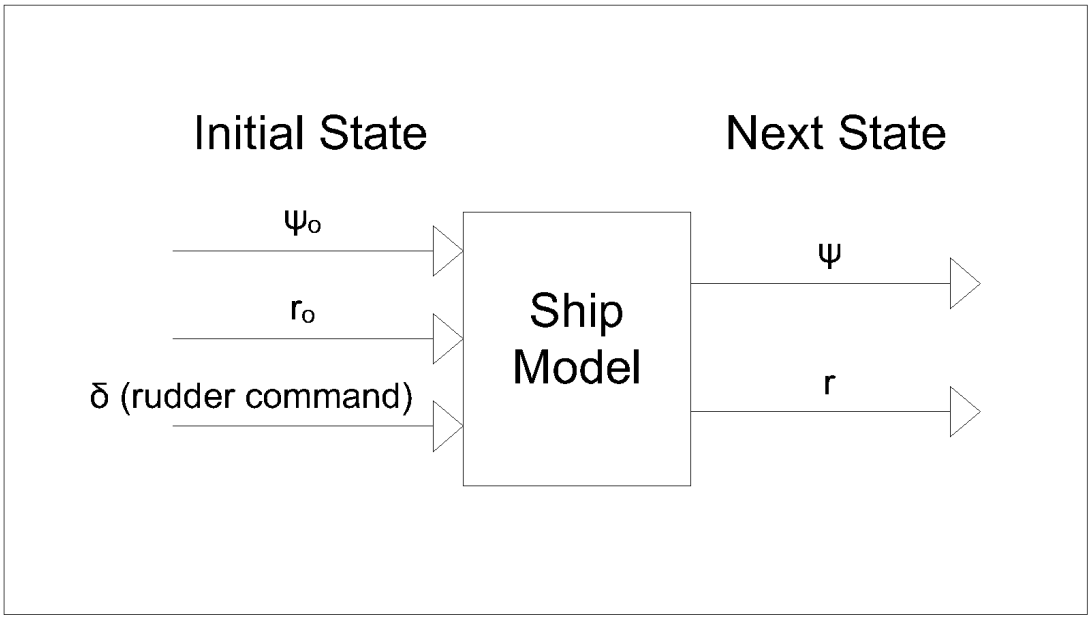

Figure 6.1: Data Producing Flow Diagram.

Accumulating the set of data by simulating the Nomoto model for different initial values, we can form a data table as illustrated on Table 6.2.

ψ_o	ρ_o	$\delta = -5^o$	.	$\delta = 5^o$
$\psi_o 1$	$\rho_o 1$	$\psi 11, \rho 11$	.	$\psi 1n, \rho 1n$
$\psi_o 2$	$\rho_o 2$	$\psi 21, \rho 21$	.	$\psi 2n, \rho 2n$
$\psi_o 3$	$\rho_o 3$	$\psi 31, \rho 31$	.	$\psi 3n, \rho 3n$
.	.	.	.	.
.	.	.	.	.

Table 6.2: Nomoto Simulation Data Format

The first three columns are the input data and the other columns are the desired output data (target). Based on these data, the neural network will be trained. We consider that the angle of rudder will not exceed 5 degrees at starboard or port side so we need data only for this rudder command range $(-5^o \quad to \quad -5^o)$.

6.1.2 Training

The neural network architecture that has been chosen is with one hidden layer which has 20 neurons. As we collect the data from simulations, we can begin the neural network training. The Figure 6.2 illustrate the workflow of the training. Having the Input and Desired Output data and using the predefined Keras algorithm we are able to train the neural network (set the appropriate weights) as illustrated on Figure 6.2 to map the input data to the output data. Thus, for random initial conditions (state) the neural network will calculate the next state for the predefined range of rudder command in accordance with the simulation data. The Keras algorithm will try to achieve mapping with a high accuracy of or about one hundred percent.

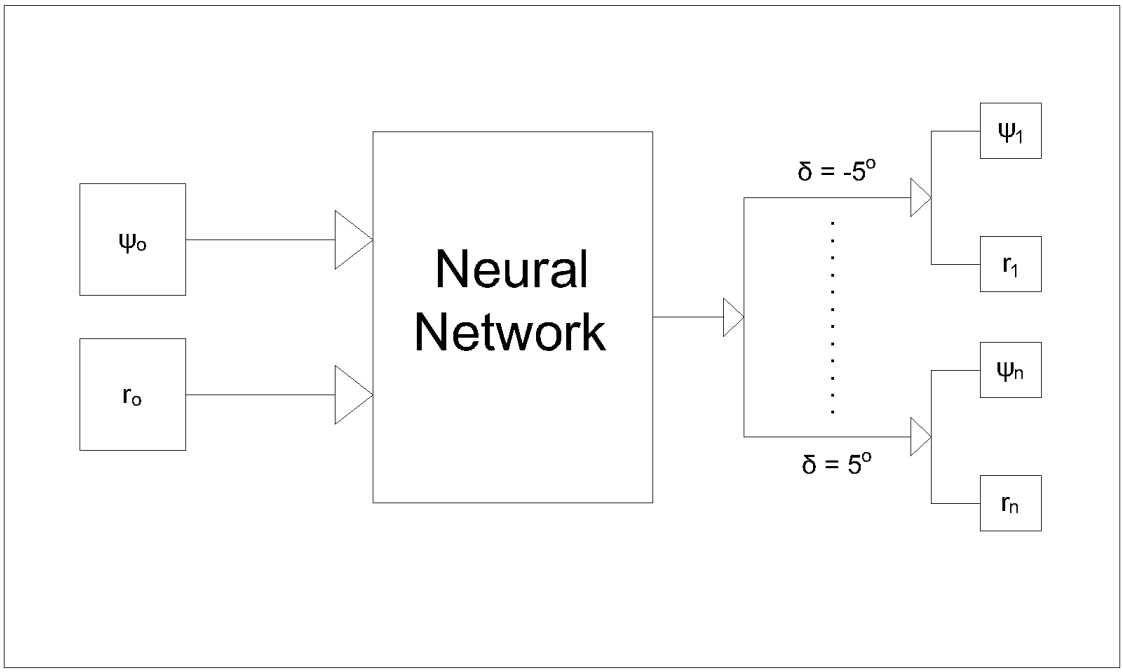

Figure 6.2: Training Flow Diagram.

The Figure 6.2 illustrate the mapping of neural network between the initial state and the next state for the predefined rudder command range ($-5^o \quad to \quad -5^o$). The more data we have accumulated the more effective imitation of the environment from the neural network we will achieve.

Neural Networks have been designed for exactly this purpose, so the training of a NN to map the input data to the output data is a very fast and easy procedure using the mentioned tools/libraries Tensorflow and Keras [21].

6.1.3 Controller

Having trained the neural network, the next step is to assemble the controller. The neural network is not able to calculate the optimum rudder command itself, so its need to add some extra modules in order to use a NN as controller. The controller has to calculate the optimum rudder command in accordance with the input state (heading and yaw velocity ψ_o, ρ_o) otherwise the vessel will not able to follow the desired path.

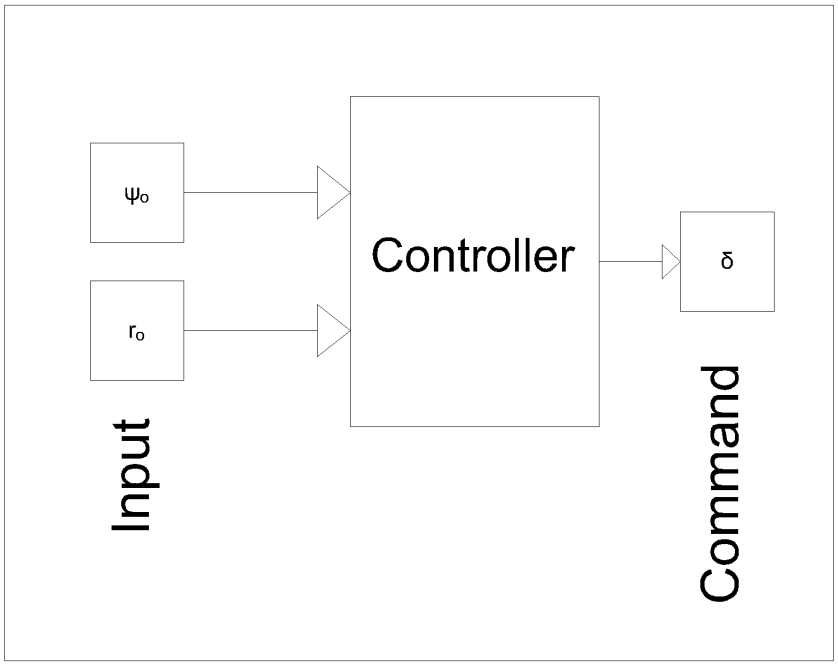

Figure 6.3: Nomoto Controller Format.

As presented in Figure 6.3, the neural network receives an initial state (ψ_o, ρ_o) and it calculate the next state (heading and yaw velocity ψ, ρ) for the predefined range of rudder command $(-5^o \quad to \quad -5^o)$. The next step is to calculate the error between the actual and the desired heading angle as shown below:

$$error = \psi_{Reference} - \psi_{actual} \tag{6.3}$$

The controller having calculated the batch of errors and of yaw velocities for the range of rudder command, now, it is able to find the rudder command $\delta_{min(error)}$ that minimize the error and the command that minimize the yaw velocity $\delta_{min(r)}$. Using the Formula 6.4, the controller calculate the optimum rudder command regarding the current conditions (state). The controller architecture is illustrated on Figure 6.4.

$$\delta = 0.2\delta_{min(error)} + 0.8\delta_{min(r)} \tag{6.4}$$

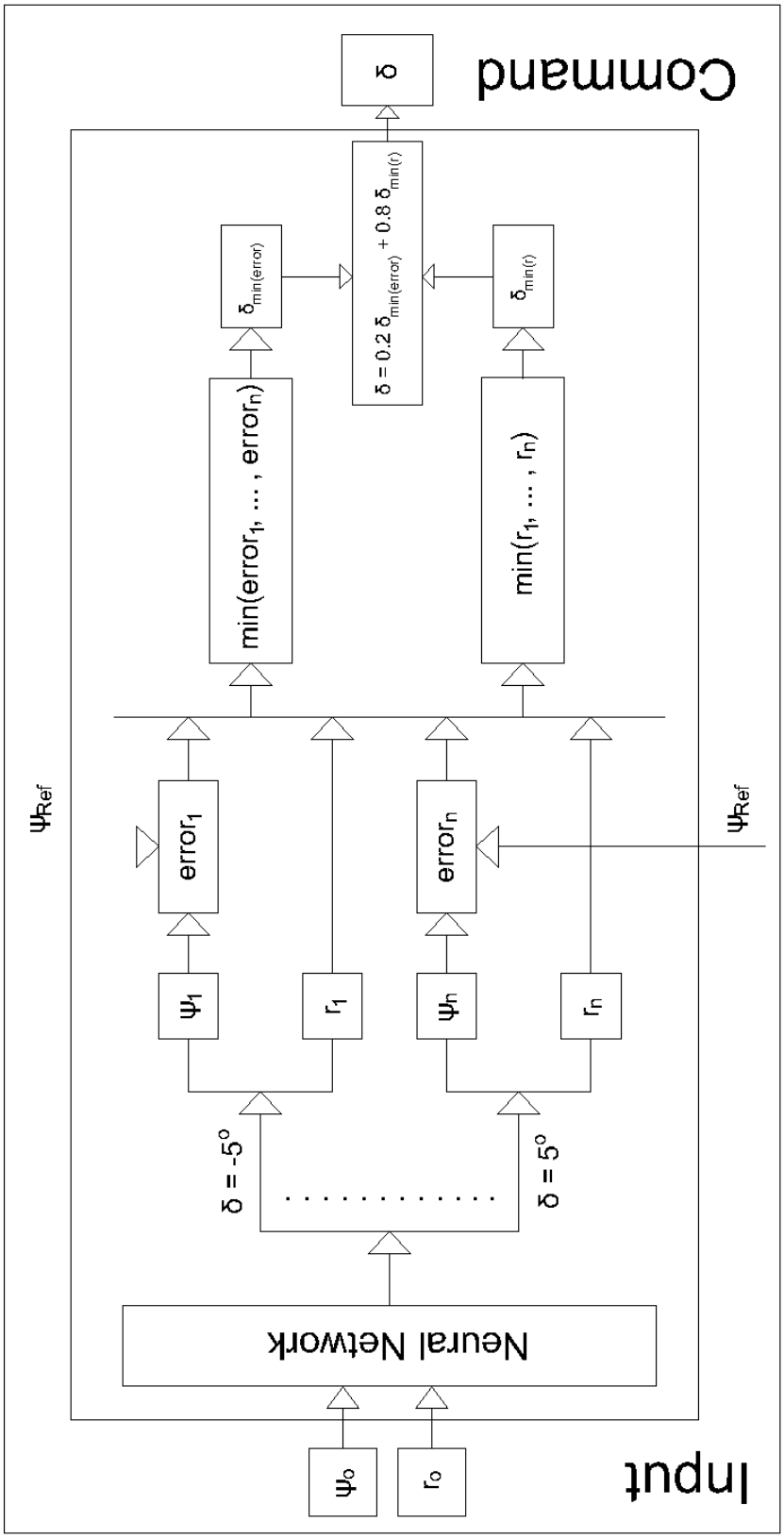

Figure 6.4: Controller Architecture.

6.1.4 Results

As we prepared the neural network controller based on Nomoto vessel model, we should test it on different conditions, thus, we implemented two different scenarios to check the controller performance. The first is that the vessel follows a **desired heading** and the second that the vessel follows a path (**path following**).

6.1.4.1 Desired Heading

The first simulation is about the following of desired heading. The system as illustrated on Figure 6.5 consists of the Neural Network controller, the Nomoto Ship model and a desired heading that we want to lead the vessel. As illustrated on Figure 6.6 the desired heading angle is 30^o and the initial heading angle is 0^o. So Autopilot has to properly turn the vessel in order to adapt her heading angle to 30^o. The duration of simulation is 200 seconds which is adequate in order to the vessel reach the goal.

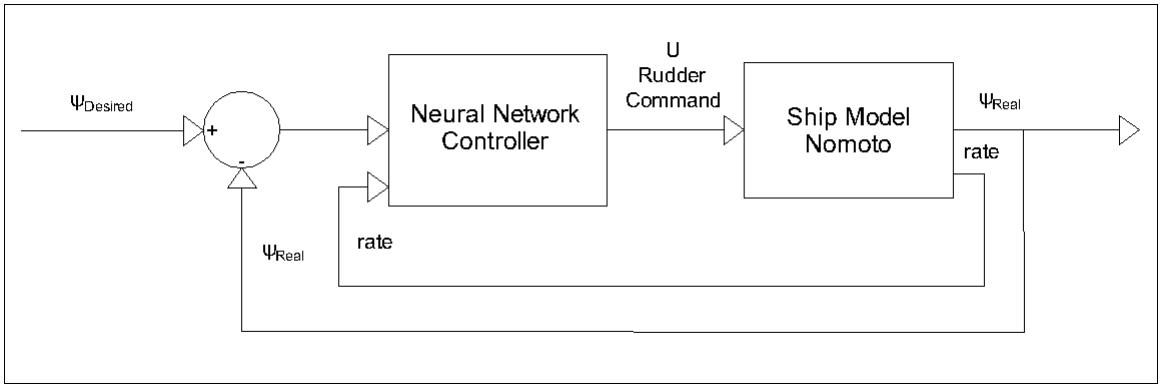

Figure 6.5: Desired Heading Following System.

The simulation was implemented on Python and on Figure 6.6 illustrated its results, which are the heading angle, the heading rate and the controller command that is the rudder angle.

Comments

The desired and actual heading angle of the ship versus time is shown in Figure 6.6. Figure clearly shows that ship is turing at 30^o as desired. Desired heading is achieved at about 50 seconds. For comparison purposes, on Figure 6.7 is presented the relevant results of a simulation using PID controller. As identified, the time needed for the vessel to reach the desired heading is about 50 seconds as well. Figure 6.6 illustrates also the yaw velocity / heading rate and the rudder angle versus time.

This simulation is very simple but at the same time very important because in this theory will be based on the next path following simulation. If the vessel cannot follow a desired heading then she will not able to follow the predefined waypoints.

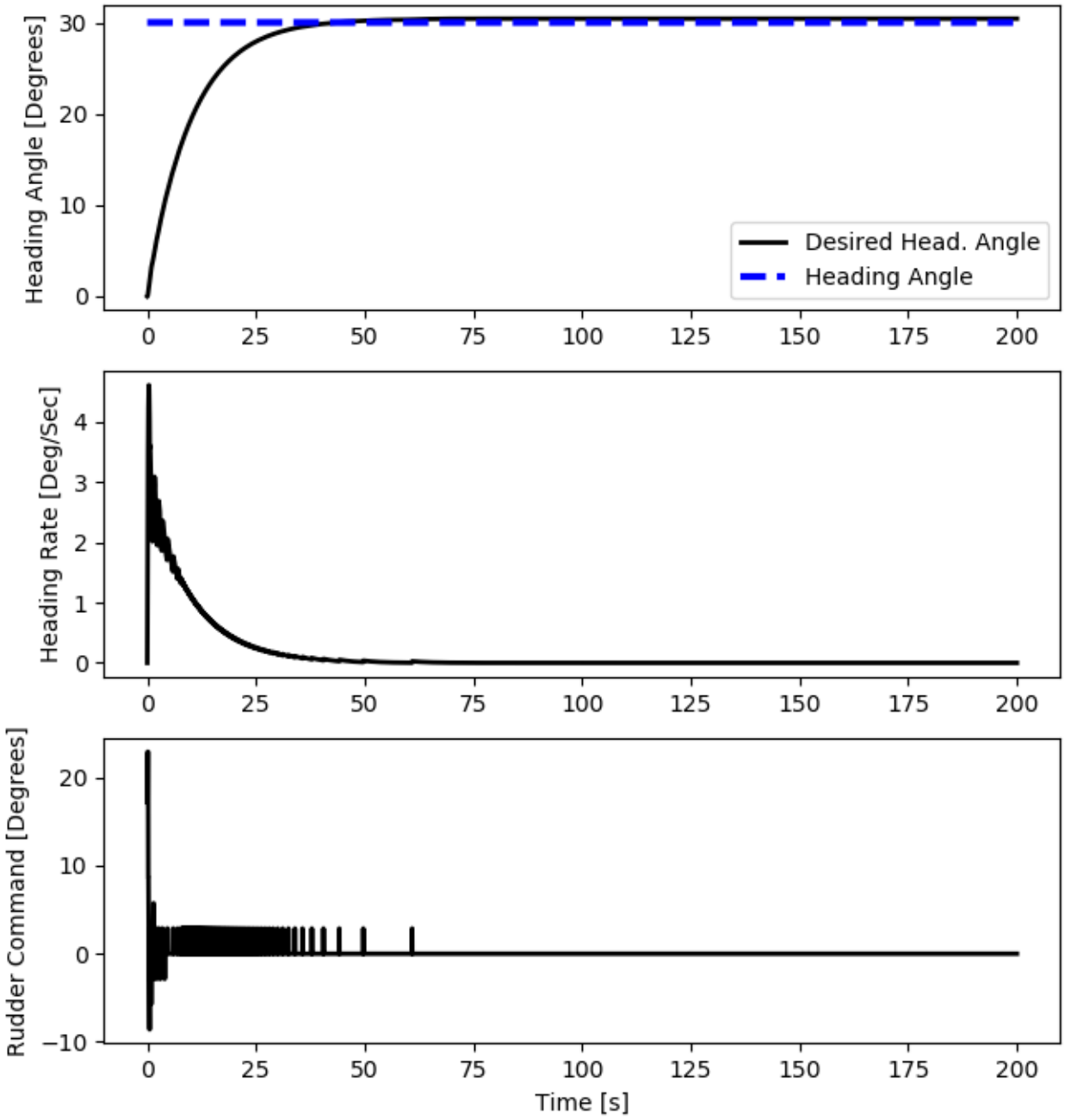

Figure 6.6: Nomoto Desired Heading Simulation using ANN Controller.

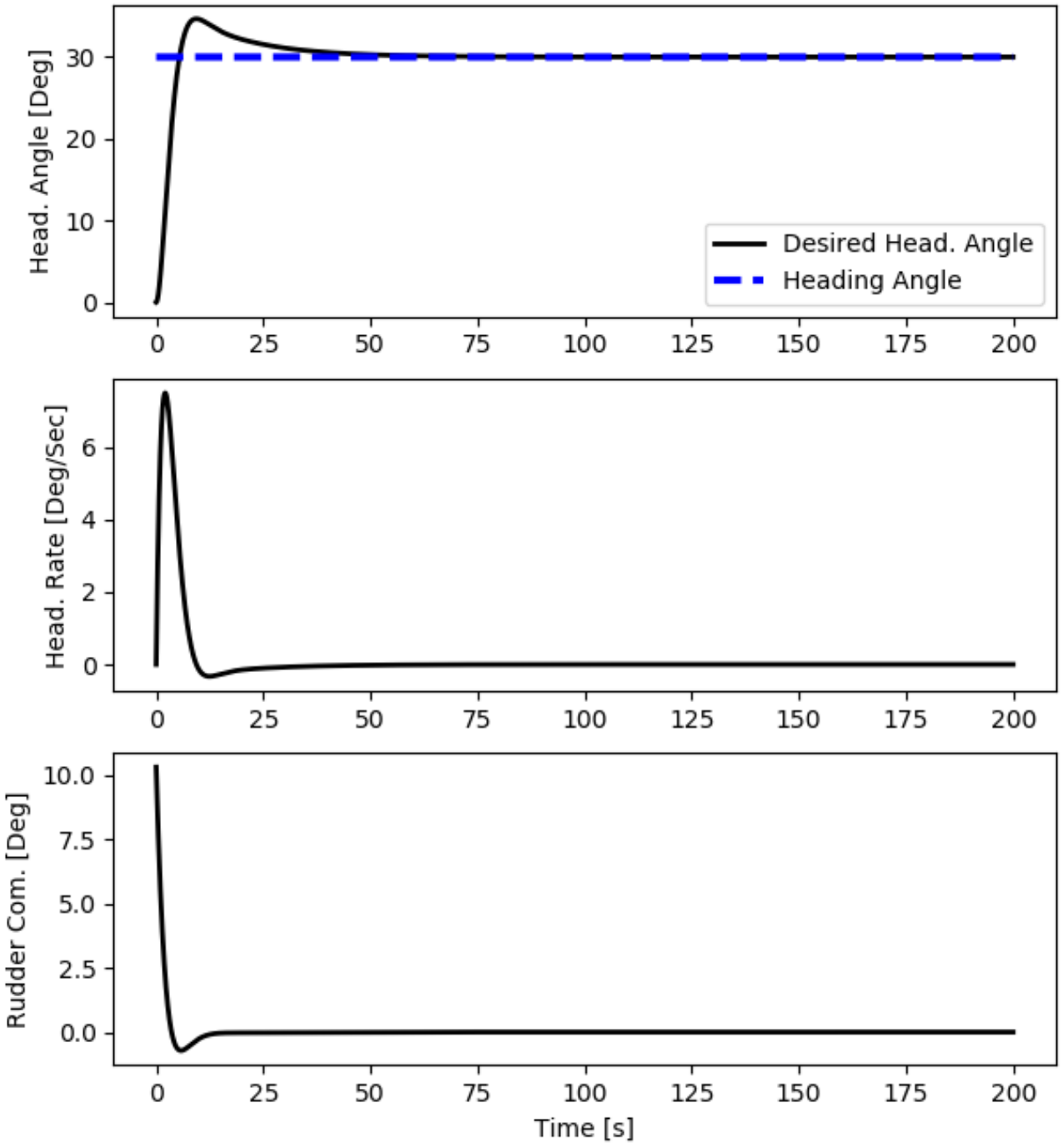

Figure 6.7: Nomoto Desired Heading Simulation using PID Controller.

6.1.4.2 Path Following

In the second simulation the vessel has to follow a predefined path on coordinate axis. A LOS guidance system was created so the vessel is able to get through the buoys, with circle of acceptance $\rho = L$ as illustrated on Table 6.3. If the vessel pass through the buoy's circle of acceptance then she will go on the next buoy. Two different implementation will be executed, the first one without the constraints and the second with them. The buoys coordinates and the constraints have been chosen according to the vessel size. The Nomoto model corresponds to a ship with the particulars of Table 5.1 so the table 6.3 was filled out accordingly.

No.	Coordinates		Constraints	
A/A	$x_{Bi}[m]$	$y_{Bi}[m]$	λ_{Bi}	$R_{Bi}[m]$
1	50	30	clockwise	10
2	100	20	counter-clockwise	10
3	50	80	clockwise	10
4	80	120	counter-clockwise	10
5	20	120	counter-clockwise	10
6	0	0	counter-clockwise	10

Table 6.3: Table of Buoys Coordinates and Constraints

The diagrammatic plan of the Guidance system is illustrated on Figure 6.8. The Guidance system receives as input the coordinates of the buoys, and check which buoy is next. In accordance with the vessel coordinates and the next buoy coordinates, the Guidance system calculate which is the appropriate yaw angle in order to the vessel get through the next buoy. The Neural Network as previous, receives as input the error between the real yaw angle and the desired yaw angle and calculate the appropriate rudder command.

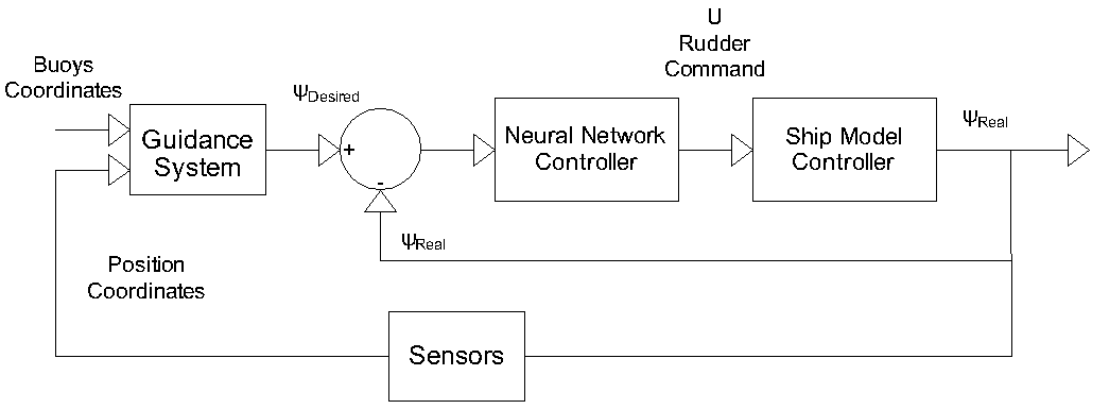

Figure 6.8: Diagrammatic plan of the Guidance system.

Following Simulation with no constraints

The result of simulation with no constraints is illustrated on Figure 6.9. This figure represent a two dimension cartesian space x-y in which the vessel is moving and trying to pass through of the mentioned buoys. When the vessel go into the circle of acceptance (dashed line circle) of each buoy then the guidance system feed to the neural network controller the next buoy as input. The buoys coordinates are depicted in Table 6.3 and the radius of circle of acceptance is $\rho = L$. In this simulation is decided to not apply any disturbance (e.g. sea current, wave or wind) as we want to check how controller is able to correspond in calm water and the feasibility of this project. The velocity of ship is 3.2 m/s and the time is needed for ship to complete the course is 130 seconds. For a vessel with this size, the mentioned time is satisfactory.

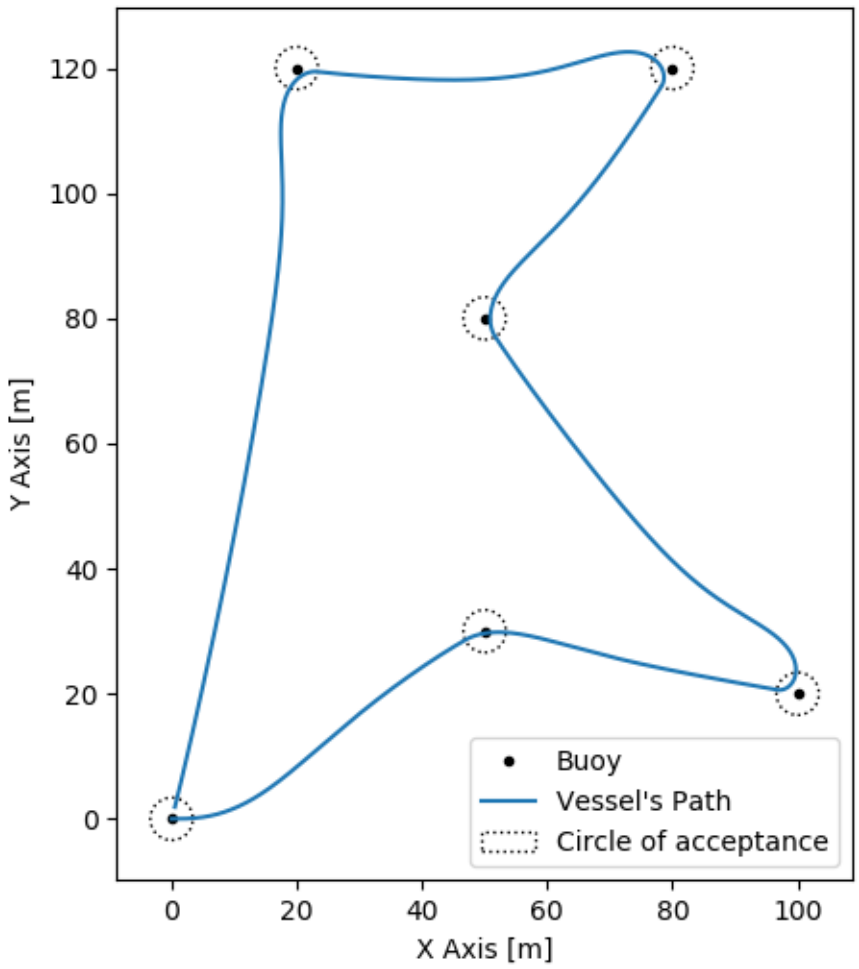

Figure 6.9: Results of Path Following Simulation without constraints.

Following Simulation with constraints

In this implementation the difficulty have been increased. The vessel has to pass around the buoy and not through of them and tangent to the circle with radius 10 meters in accordance with the Table 6.3. In reality the buoys are physical with result a ship cannot pass through of them as considered during first simulation. The buoys coordinates are the same with the previous simulation as shown in Table 6.3. In this simulation, no disturbance has been applied as well as with the first simulation.

According to Figure 6.3, the neural network controller successfully conduct the vessel to meet the mentioned requirements and follow the predefined path, as shown in Figure 6.10. The velocity of ship is 3.2 m/s and the time is needed for ship to complete the course is 160 seconds. The ship is able to move very smoothly and efficiently and as shown in Figure 6.10 the vessel don't have any deviation of her course.

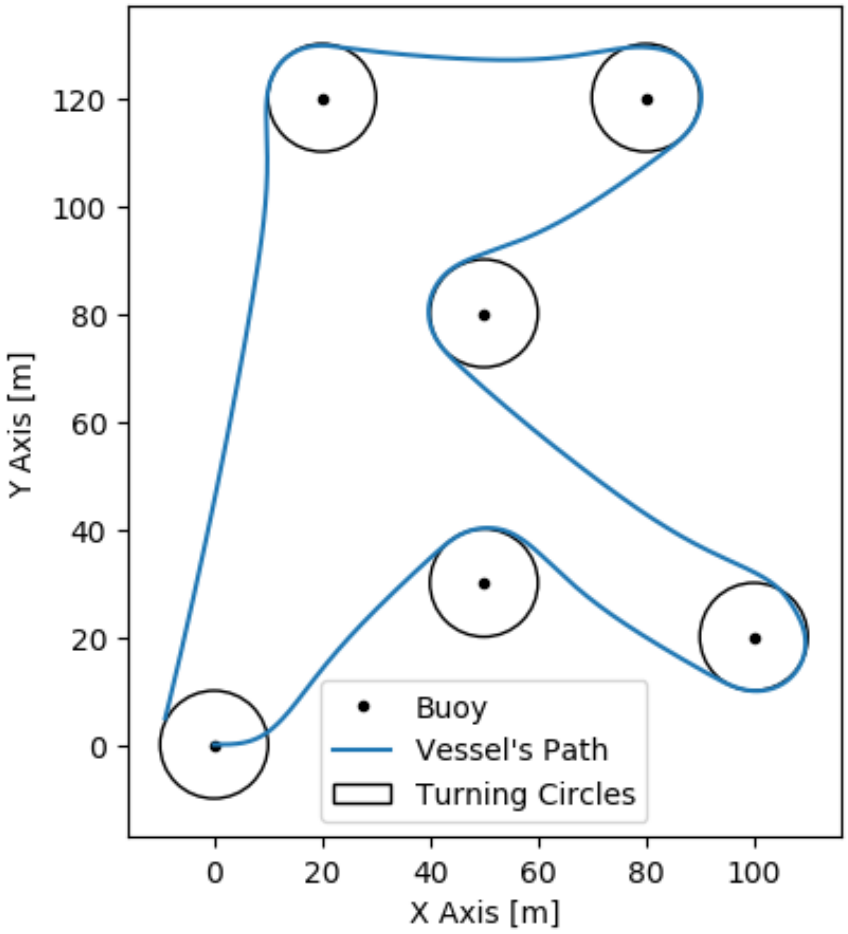

Figure 6.10: Results of Path Following Simulation with constraints.

6.2 4DOF Model Implementation

The 4DOF is a more analytical vessel model in contrast with the First-Order Nomoto Model. As described on chapter 5.2, the model consist of the following rigid-body equations of motion (equations 5.12):

$$\begin{aligned} m[\dot{u} - {y_g}^b\dot{r} - vr - {x_g}^b r^2 + {z_g}^b pr] &= \tau_1 \\ m[\dot{u} - {z_g}^b\dot{p} + {x_g}^b\dot{r} + ur - {y_g}^b(r^2 + p^2)] &= \tau_2 \\ I_{xx}\dot{p} - m{z_g}^b\dot{v} + m[{y_g}^b vp - {z_g}^b ur] &= \tau_4 \\ I_{zz}\dot{r} + m{x_g}^b\dot{v} - m{y_g}^b\dot{u} + m[{x_g}^b ur + {y_g}^b vr] &= \tau_6 \end{aligned}$$

The vector of forces are separated into the following components (equation 5.14):

$$\tau = \tau_{hyd} + \tau_c + \tau_p + \tau_w$$

Thus, the 4DOF model equations have only one independent and six dependent variables as illustrated in Table 6.4.

Variables	Quantity
Independent	δ
Dependent	u, v, r, p, ψ, ϕ

Table 6.4: 4DOF Model Independent and Dependent Variable

We will follow the same path with the chapter 6.1 to create a new controller for the ship in order to the 4DOF model vessel to be able to successfully execute the same simulations as described in the previous sub chapter. Except of the different more analytical model, there is an extra different between two models as well. The 4DOF model vessel is bigger than the Nomoto model and the main particulars of them can be compared on Table 6.5.

Quantity	Nomoto Model	4DOF Model	Unit
L	3.4	51.5	m
Vs	3.2	7.7	m/s

Table 6.5: Vessel's Main Particulars and Coefficients

6.2.1 Data Collection

As implemented in Chapter 6.1 (Nomoto Model), the first step is to produce the appropriate data to be used for the training of neural network. According to the Table 6.4, there are one independent and six dependent variables, so for different initial heading and yaw velocity (u_o, v_o, r_o, p_o, ψ_o, ϕ_o) and rudder command (δ) will be obtain the values for the next state (u, v, r, p, ψ, ϕ) after $dt = 0.01s$.

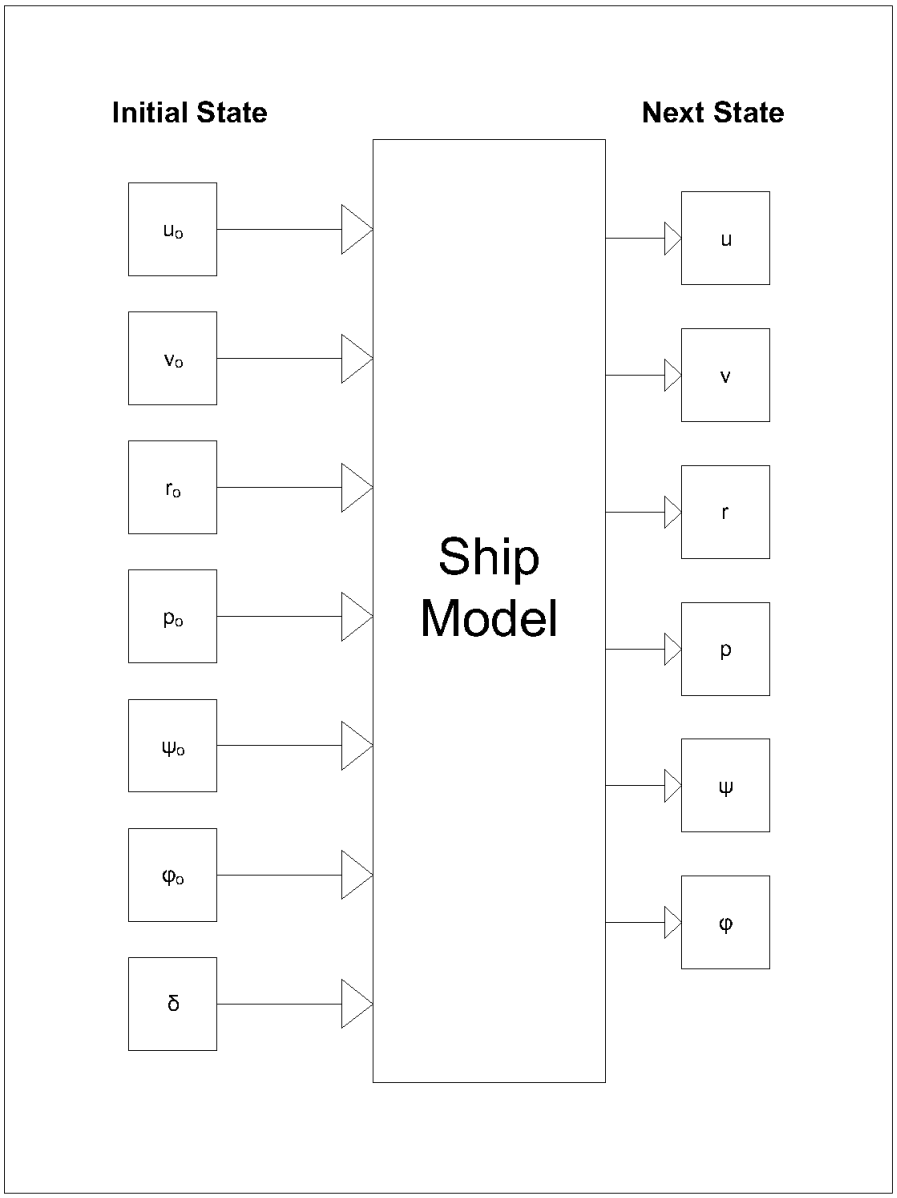

Figure 6.11: Data Producing Flow Diagram.

Accumulating the set of data by simulating the 4DOF model for different initial values, we can form a data table as illustrated on Table 6.6 and Table 6.7.

u_o	v_o	r_o	p_o	ψ_o	ϕ_o
u_o1	v_o1	r_o1	p_o1	ψ_o1	ϕ_o1
u_o2	v_o2	r_o2	p_o2	ψ_o2	ϕ_o2
u_o3	v_o3	r_o3	p_o3	ψ_o3	ϕ_o3
.	.	.	.	.	.
.	.	.	.	.	.

Table 6.6: 4DOF Simulation Input Data Format

$\delta = -5^o$	.	$\delta = 5^o$
$u11, v11, r11, p11, \psi11, \phi11$	.	$u1n, v1n, r1n, p1n, \psi1n, \phi1n$
$u21, v21, r21, p21, \psi21, \phi21$	.	$u2n, v2n, r2n, p2n, \psi2n, \phi2n$
$u31, v31, r31, p31, \psi31, \phi31$	.	$u3n, v3n, r3n, p3n, \psi3n, \phi3n$
.	.	.
.	.	.

Table 6.7: 4DOF Simulation Output Data Format

The first Table 6.6 is about the input data trying to simulate all the possible situations. The second Table 6.7 contains the next state in accordance with the rudder angle. All of these data will be used for the neural network training describing in the next chapter. We consider as Nomoto model training that the angle of rudder will not exceed 5 degrees at starboard or port side so we need data only for this rudder command range ($-5^o \quad to \quad -5^o$).

6.2.2 Training

For the 4DOF model, the neural network architecture that has been chosen is with one hidden layer which has 30 neurons. The training of neural network based on the collected data of Chapter 6.2.1. The workflow of training is illustrated on the Figure 6.12. Using Keras we will calculate the Neural Network weights so it able to map the input data on the output data. Thus, for random initial conditions (states) the neural network can predict the next state of the model for the predefined range of rudder command in accordance with the simulation data. The Keras algorithm will try to achieve mapping with a high accuracy of or about one hundred percent.

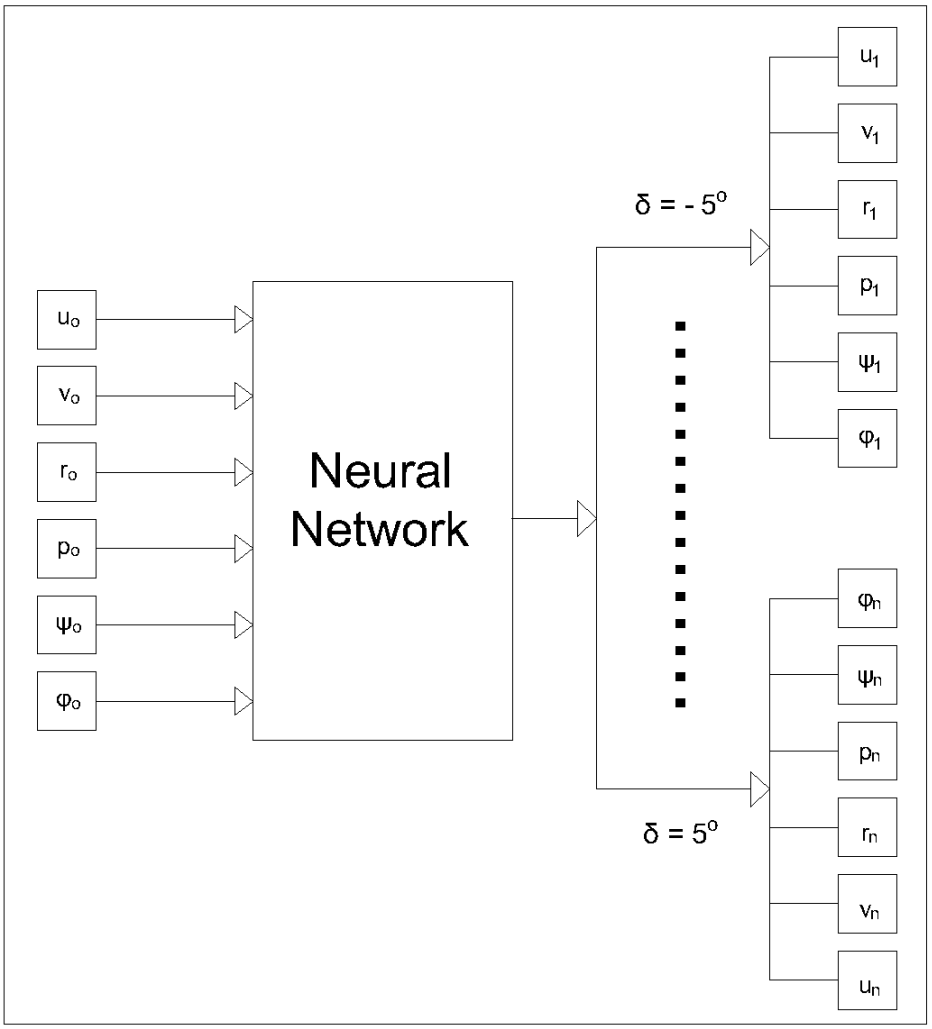

Figure 6.12: Training Flow Diagram.

The Figure 6.12 illustrate the mapping of neural network between the initial state and the next state for the predefined rudder command range ($-5^o \quad to \quad -5^o$). The training procedure is exactly the same with the neural training which was implemented for Nomoto model.

6.2.3 Controller

As discussed in Chapter 6.1.3 for Nomoto model, the neural network is not able to calculate the optimum rudder command itself, it can only imitate the environment of the vessel's model. So we will add some extra modules in order to the controller can calculate the optimum rudder command in accordance with the input state. Using these modules the vessel can follow the desired heading angle or path accordingly.

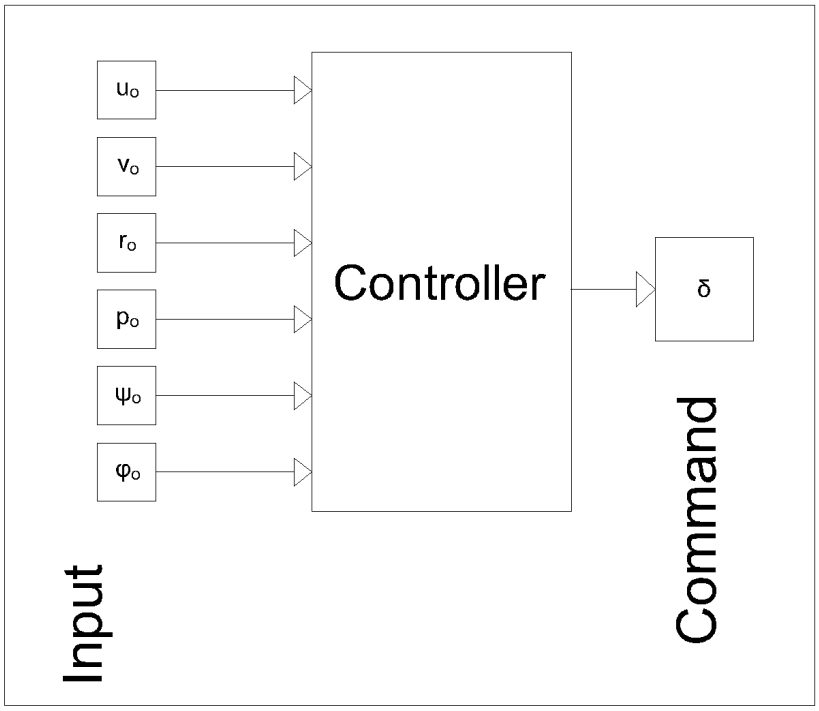

Figure 6.13: 4DOF Controller Format.

The neural network receives an initial state $(u_o, v_o, r_o, p_o, \psi_o, \phi_o)$ and it calculate the next state (u, v, r, p, ψ, ϕ) for the predefined range of rudder command $(-5^o \quad to \quad -5^o)$ as shown in Figure 6.13. The next step is to calculate the error between the actual and the desired heading angle as shown below:

$$error = \psi_{Reference} - \psi_{actual}$$

The controller having calculated the batch of errors and of heading rates for the range of rudder command, now, it is able to find the rudder command $\delta_{min(error)}$ that minimize the error and the command that minimize the yaw velocity $\delta_{min(r)}$. Using the below formula, the controller calculate the optimum rudder command regarding the current conditions (state). The Controller design is described on Figure 6.14.

$$\delta = 0.2\delta_{min(error)} + 0.8\delta_{min(r)}$$

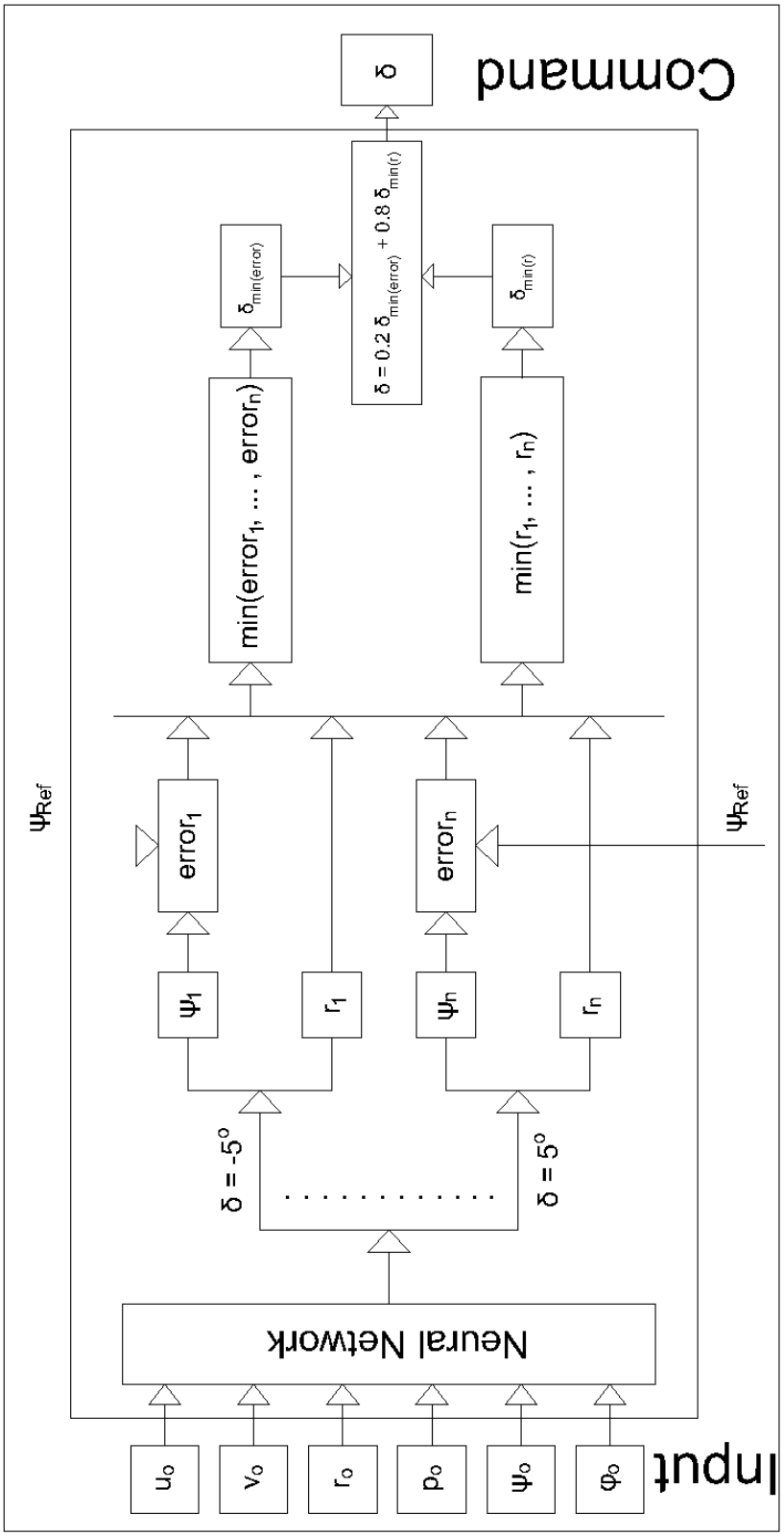

Figure 6.14: Controller Architecture.

6.2.4 Results

As we prepared the neural network controller based on 4DOF vessel model, we should test it on different conditions as implemented with Nomoto model as well. We will carry out simulations with two different scenarios to evaluate the neural network controller efficiency and effectiveness. The first is that the vessel follows a **desired heading** and the second that the vessel follows a predefined path (**path following**).

6.2.4.1 Desired Heading

The first simulation is about the following of desired heading. The system as illustrated on Figure 6.15 consists of the Neural Network controller, the 4DOF Ship model and a desired heading that we want to lead the vessel. As illustrated on Figure 6.16 the desired heading angle is 30^o and the initial heading angle is 0^o. The duration of simulation is 200 seconds which is adequate in order to the vessel reach the goal.

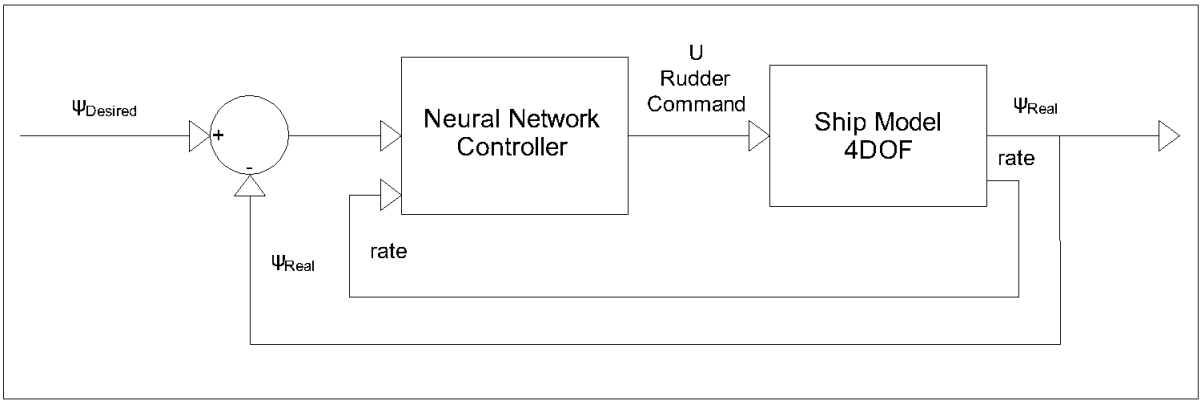

Figure 6.15: Desired Heading Following System.

The simulation was implemented on Python and on Figure 6.16 and Figure 6.17 illustrated its results, which are the Roll angle ϕ, the Yaw angle ψ (or Heading angle), the Rudder angle δ, and the velocities u, v, p, r.

Comments

Figure 6.16 and Figure 6.17 are shows the results of the path following simulation. The Figure 6.16 clearly shows that ship is turning at 30^o as desired. Desired heading is achieved at about 120 seconds. For comparison purposes, on Figure 6.18 and Figure 6.19 are presented the relevant results of a simulation using PID controller. As identified, the time needed for the vessel to reach the desired heading is about 120 seconds as well.

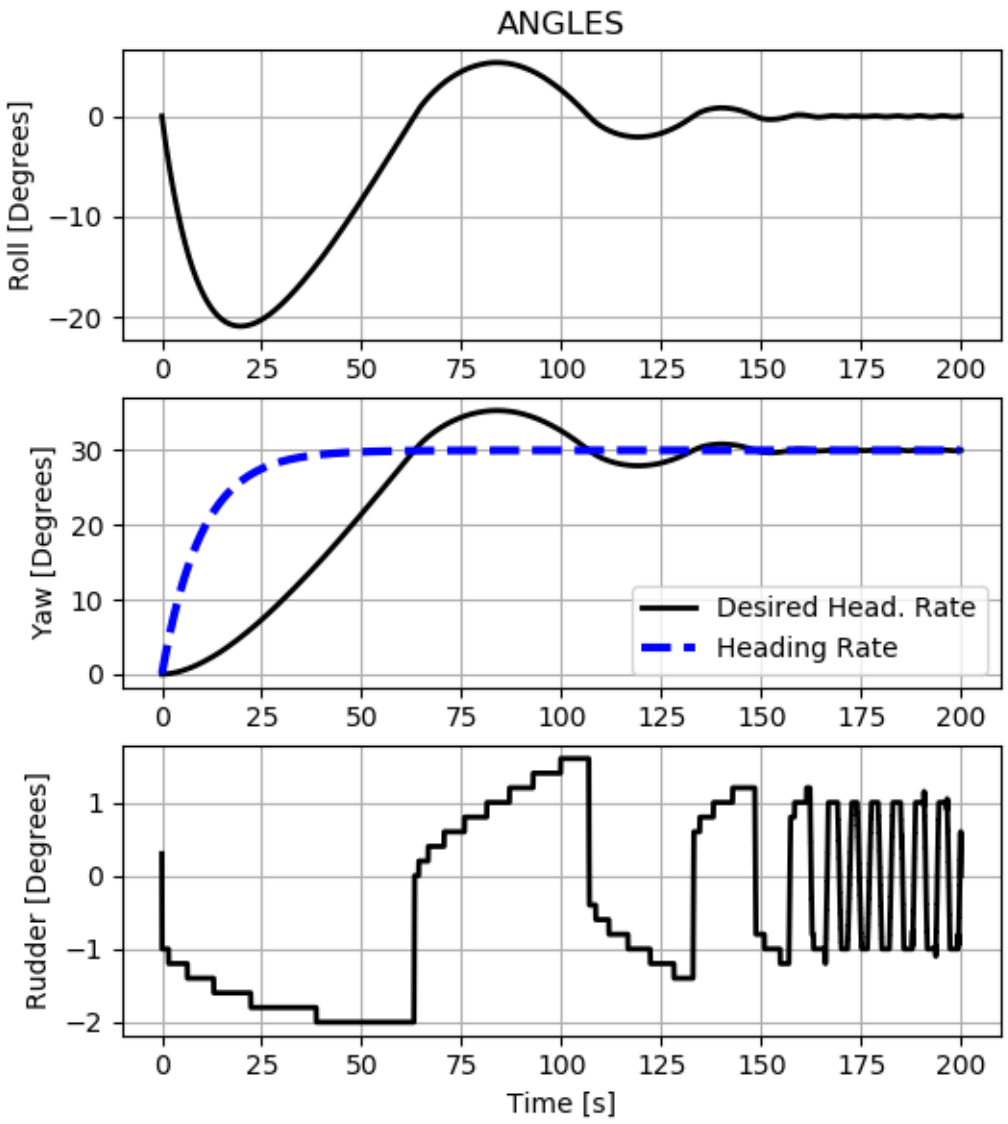

Figure 6.16: 4DOF Desired Heading Simulation using ANN - Angles.

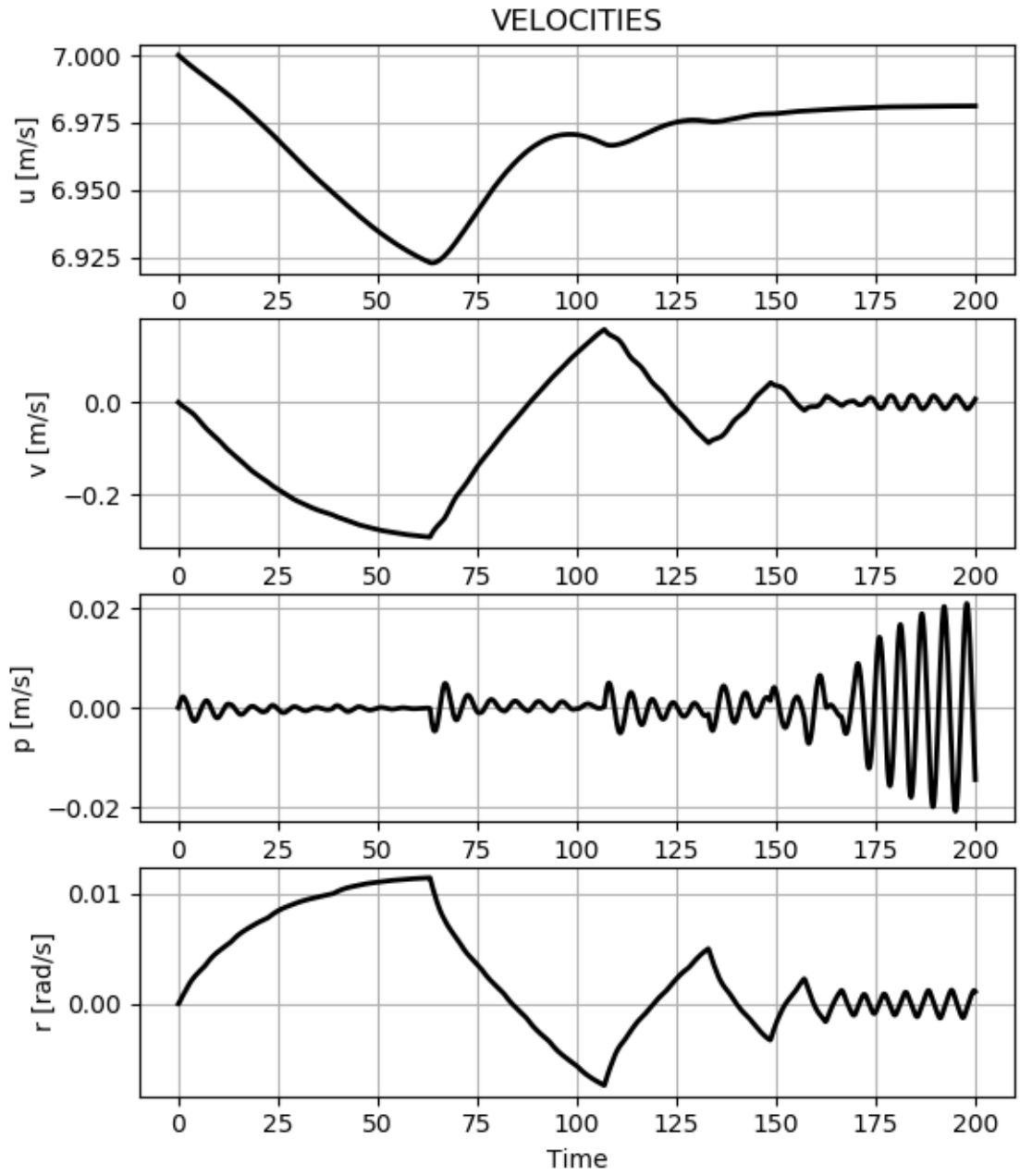

Figure 6.17: 4DOF Desired Heading Simulation using ANN - Velocities

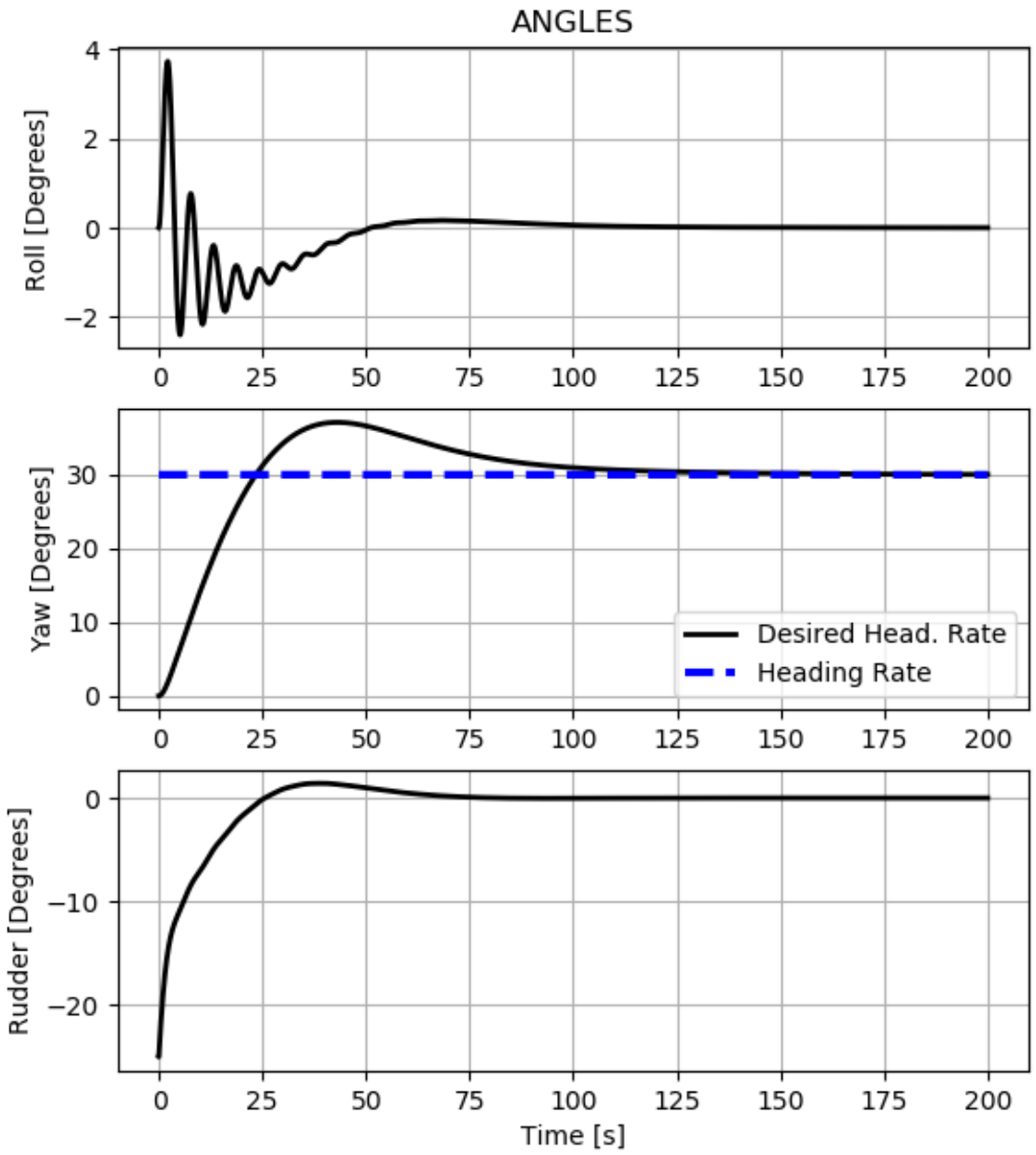

Figure 6.18: 4DOF Desired Heading Simulation using PID - Angles.

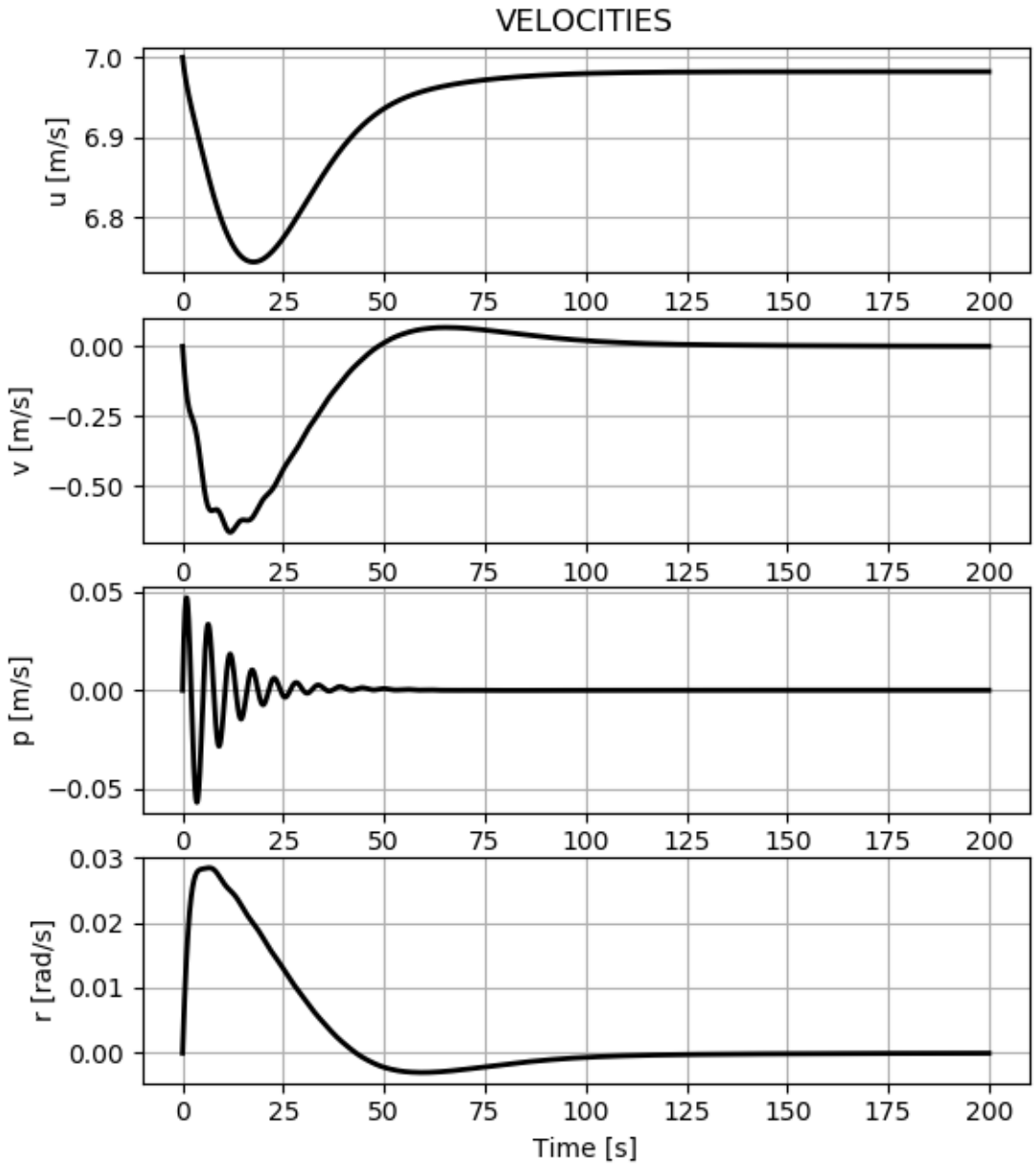

Figure 6.19: 4DOF Desired Heading Simulation using PID - Velocities

6.2.4.2 Path Following

In the second simulation the vessel should follow a predefined path on coordinate axis. A LOS guidance system was created so the vessel is able to get through the buoys, with circle of acceptance $\rho = L$ as illustrated on Table 6.8. Two different implementation will be executed, the first one without the constraints and the second with them. The buoys coordinates and the constraints have been chosen according to the vessel size. The 4DOF model corresponds to a ship with the particulars of Table 5.2 so the Table 6.8 was filled out accordingly.

No.	Coordinates		Constraints	
A/A	$x_{Bi}[m]$	$y_{Bi}[m]$	λ_{Bi}	$R_{Bi}[m]$
1	500	300	clockwise	100
2	1000	200	counter-clockwise	100
3	500	800	clockwise	100
4	800	1200	counter-clockwise	100
5	200	1200	counter-clockwise	100
6	0	0	counter-clockwise	100

Table 6.8: Table of Buoys Coordinates and Constraints

The simulation is the same with Nomoto model in Chapter 6.1.4. The diagrammatic plan of the Guidance system is shown on Figure 6.20. The Guidance system receives as input the coordinates of the buoys, and check which buoy is next. In accordance with the vessel coordinates and the next buoy coordinates, the Guidance system calculate which is the appropriate yaw angle in order to the vessel get through the next buoy. The Neural Network as previous, receives as input the error between the real yaw angle and the desired yaw angle and calculate the appropriate rudder command.

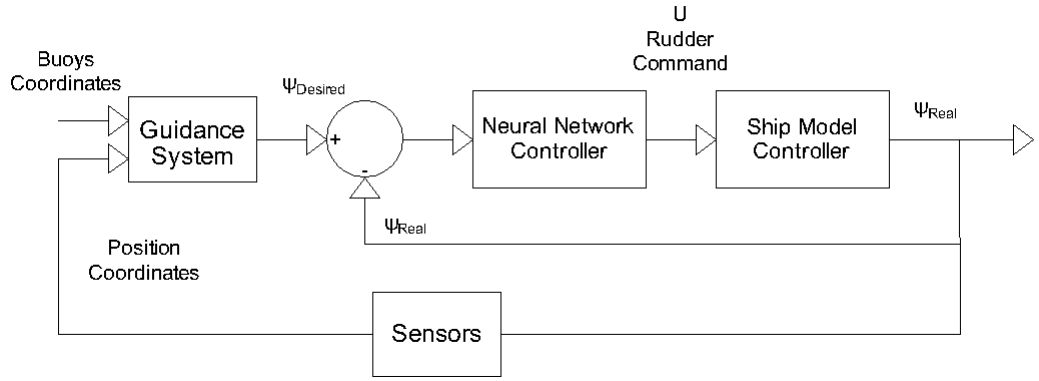

Figure 6.20: Diagrammatic plan of the Guidance system.

Following Simulation with no constraints

The result of simulation with no constraints is illustrated on Figure 6.21. This figure represent a two dimension cartesian space x-y in which the vessel is moving and trying to pass through of the mentioned buoys. When the vessel go into the circle of acceptance (dashed line circle) of each buoy then the guidance system feed to the neural network controller the next buoy as input. The buoys coordinates are depicted in Table 6.8 and the radius of circle of acceptance is $\rho = L$. The velocity of ship is $7m/s$ and the time is needed for ship to complete the course is 700 seconds. For a vessel with this size, the mentioned time is satisfactory.

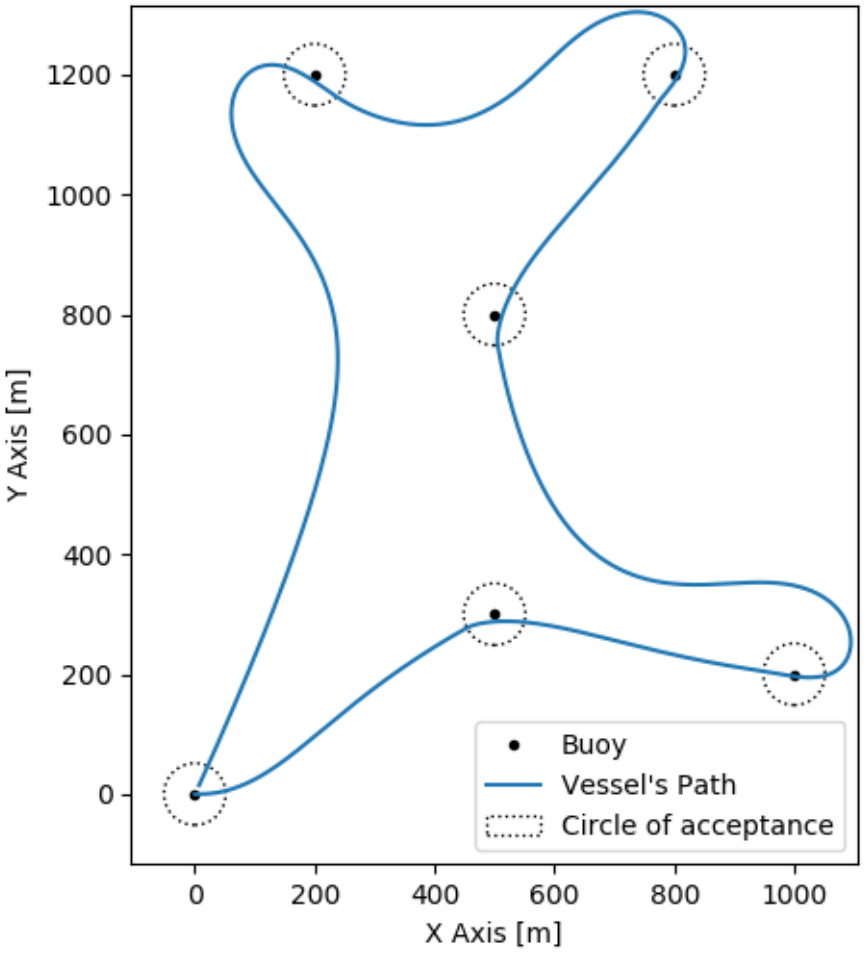

Figure 6.21: Results of Path Following Simulation without constraints.

Following Simulation with constraints

In this implementation the difficulty have been increased. The vessel has to pass around the buoy and not through of them and tangent to the circle with radius 10 meters in accordance with the Table 6.8. The buoys coordinates are the same with the previous simulation as shown in Table 6.3. In this simulation, no disturbance has been applied as well as with the above simulations.

According to Figure 6.8, the neural network controller successfully conduct the vessel to meet the mentioned requirements and follow the predefined path, as shown in Figure 6.22. The velocity of ship is 7 m/s and the time is needed for ship to complete the course is 720 seconds. The ship is able to move very smoothly and efficiently and as shown in Figure 6.22 the vessel don't have any deviation of her course.

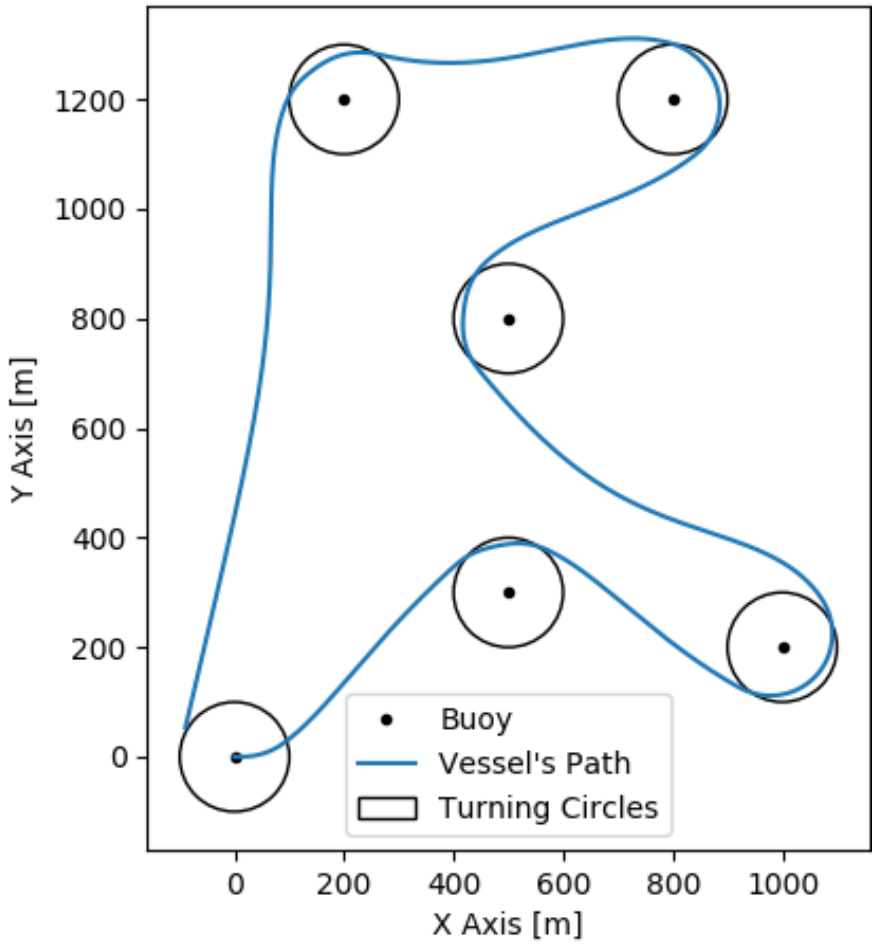

Figure 6.22: Results of Path Following Simulation with constraints.

Chapter 7

Conclusion and Future Work

In this thesis, we addressed the problem of designing and constructing a vessel's autopilot using neural networks and deep learning. One of the main contribution of this work is to develop new and more complicated technologies, which can be used for the vessel's navigation. The challenge is to include more parameters and variables that could be taken into account during the autopilot's decision. In this thesis the benefit, in comparison with the old type controllers, is that we have built a controller which can get as inputs 4 different vessel's velocities which are responsible for surface manoeuvring. These data can conduct a better comprehension of each situation/state leading to a more effective decision.

A Neural Network controller can learn from the environment to become more efficient and effective in contrast with other types of controllers which cannot adapt accordingly. The Artificial Neural Networks is a computing system designed to replicate the way humans analyze and repeat work. Neural Networks have the ability to learn by themselves and produce the output that is not limited to the input provided to them. The input is stored in its own networks instead of a database, hence the loss of data does not affect its working. Even if a neuron is not responding or a piece of information is missing, the network can detect the fault and still produce the output. Regarding the mentioned ANN advantages, ANN are a very useful tool that can become more preferable in the future.

Many different parameters and variable can be used as controller inputs for better effectiveness. Sea waves, weather, wind, and fuel optimization are some parameters that could be processed by a neural network controller and could be taken into account in a future work. The most important is using Artificial intelligence and autonomy to improve safety, increase efficiency, and relieve human operators from unsafe and repetitive tasks.

Bibliography

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[2] Mogens Blanke and Anders C Christensen. *Rudder-roll damping autopilot robustness to sway-yaw-roll couplings*. Aalborg Universitetscenter., 1993.

[3] Tom M Mitchell et al. *Machine learning. 1997*. WCB/McGraw-Hill, 1997.

[4] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[5] Geoffrey E Hinton, Terrence Joseph Sejnowski, Tomaso A Poggio, et al. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[6] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[7] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[8] Morten Breivik and Thor I Fossen. Guidance laws for autonomous underwater vehicles. *Underwater vehicles*, pages 51–76, 2009.

[9] Thor I Fossen, Morten Breivik, and Roger Skjetne. Line-of-sight path following of underactuated marine craft. *Proceedings of the 6th IFAC MCMC, Girona, Spain*, 244249, 2003.

[10] Esmat Bekir. *Introduction to modern navigation systems*. World Scientific, 2007.

[11] W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.

[12] R Burns and R Richter. A neural-network approach to the control of surface ships. *Control Engineering Practice*, 4(3):411–416, 1996.

[13] Arbab Nighat Khizer and MA Dai Yaping. Unar," design of heading controller for cargo ship using feed forward artificial neural network. *International Journal of Advancements in Computing Technology (IJACT)*, 5(9):556–566, 2013.

[14] J Wesley. *Core python programming*. Prentice-Hall, 2006.

[15] Alex Martelli. *Python in a Nutshell*. " O'Reilly Media, Inc.", 2006.

[16] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[17] Tristan Perez, Andrew Ross, and Thor I Fossen. A 4-dof simulink model of a coastal patrol vessel for manoeuvring in waves. 2006.

[18] Thor I Fossen et al. *Guidance and control of ocean vehicles*, volume 199. Wiley New York, 1994.

[19] Edward V Lewis. *Principles of Naval Architecture: Vol. 3: Motions in Waves and Controllability*. Society of Naval Architects & Marine Engineers The, 1988.

[20] Volker Bertram. *Practical ship hydrodynamics*. Elsevier, 2011.

[21] Sebastian Raschka. *Python machine learning*. Packt Publishing Ltd, 2015.

Appendix A

Code Blocks

Listing A.1: Training Part of Code

```
import random
import numpy as np
import math
import os
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras import backend as K
import scipy.misc
from scipy.integrate import odeint

class QNetwork():
    def __init__(self):
        self.model = Sequential()
        self.model.add(Dense(20, input_dim= 2,
        kernel_initializer='normal', activation='linear'))
        self.model.add(Dense(41, kernel_initializer='normal',
         activation='linear'))
        self.model.compile(loss='mean_squared_error',
        optimizer=Adam(lr=0.01), metrics=['accuracy'])

    def predict(self, state):
        self.Qout = self.model.predict(state)
        self.action = np.argmin(abs(self.Qout[0]))
        return [self.Qout, self.action]

    def fit(self, state, label):
        self.model.fit(state, label, batch_size = 5000,
        epochs=100, verbose=1)

    def evaluate(self, X, Y):
```

```
        score = self.model.evaluate(X, Y, verbose=0)

    def save(self):
        self.model.save('Models/nomoto_r.h5')


if __name__ == "__main__":

    agent = QNetwork()

    data = np.load('Data/data2.npy')
    nn_input = np.vstack(data[:, 0])
    nn_label = np.vstack(data[:, 1])

    agent.fit(nn_input, nn_label)
    agent.save()

    #agent.evaluate(nn_input, nn_label)
```

Listing A.2: Simulation Part of Code

```
import numpy as np
import math
import random
from keras.models import load_model
import scipy.misc
from scipy.integrate import odeint
import matplotlib.pyplot as plt

class Model():
    def __init__(self, dt):
        self.K = 3.47
        self.T = 3.94

        self.delta_t = dt

    def model(self, r, t, u):
        drdt = (-r + self.K * u) / self.T
        return drdt

    def State(self, u, state, psid):
        psio = state[0, 2]
        ro = state[0, 1]
        y = odeint(self.model, ro, [0, self.delta_t], args=(u,))
        self.r = y[-1]
        self.psi = (psio + self.r * self.delta_t) #
```

```
            + random.uniform(-0.0001, 0.0001)
            self.e = psid - self.psi  # error desired head. angle - head. angle
            state = np.array([[float(self.e), float(self.r), float(self.psi)]])
            return state


class QNetwork():
    def __init__(self):
        self.model = load_model('Models/nomoto_error.h5')

    def predict(self, state):
        self.Qout = self.model.predict(state)
        self.action = np.argmin(abs(self.Qout[0]))
        return [self.Qout, self.action]

class QNetwork2():
    def __init__(self):
        self.model = load_model('Models/nomoto_r.h5')

    def predict(self, state):
        self.Qout = self.model.predict(state)
        self.action = np.argmin(abs(self.Qout[0]))
        return [self.Qout, self.action]



if __name__ == "__main__":
    time = 200
    dt = 0.01
    t = np.arange(0, time + dt, dt)
    len_t = len(t)

    # Storage for recording values
    u = np.zeros(len_t)      # controller output
    psi = np.zeros(len_t)    # heading angle
    r = np.zeros(len_t)      # heading rate
    e = np.zeros(len_t)      # error
    ie = np.zeros(len_t)     # integral of the error
    de = np.zeros(len_t)     # derivative of the error
    psid = np.zeros(len_t)   # desired heading angle
    psid_ = np.zeros(len_t)
    psid_[:] = 3

    for i in range (0, len_t):
        psid[i]=(1-math.exp(-t[i]/10))*math.radians(3)
```

```
    #Initial Condition
    r[0] = 0
    psi[0] = 0
    e[0] = psid[0] -psi[0]

    ship = Model(dt)
    agent = QNetwork()
    agent2 = QNetwork2()
    state = np.array([[e[0], r[0], psi[0]]])     # #0: error  #1: r
#2: psi

    actions = np.arange(-5, 5.25, 0.25)
    len_actions = len(actions)
    # Convert action to radians
    for i in range (0, len_actions):
        actions[i] = math.radians(actions[i])


    for i in range (0, len_t - 1):

        ind_action1 = agent.predict(np.reshape(np.array
        (state[0, :2]),[1,2]))[1]
        ind_action2 = agent2.predict(np.reshape(np.array
        (state[0, :2]),[1,2]))[1]
        action = 0.2*actions[ind_action1] + 0.8*actions[ind_action2]

        new_state = ship.State(action, state, psid[i])

        u[i] = math.degrees(action)
        psi[i + 1] = math.degrees(new_state[0, 2])
        r[i + 1] = math.degrees(new_state[0, 1])
        e[i + 1] = math.degrees(new_state[0, 0])

        state = new_state
```