Hybrid Diesel-Electric Marine Propulsion Plant Control with Neural Networks

Stergios Evangelos Bachoumas

Diploma Thesis

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

Supervisor: Assistant Prof. George Papalambrou

Committee Member : Prof. N. Kyrtatos
Committee Member : Associate Prof. Ch. Papadopoulos

2019 October

Acknowledgements

Ever since I was a little kid, my parents wanted the best for my education and made truly such great sacrifices to get me to the point that I am today. So from the bottom of my heart I want to thank them for their continuous support and for pushing me to the right directions at the right times.

The present work is the final requirement for the fulfilment of my studied and it was completed at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens (NTUA), under the supervision and instrumental help of Assistant Professor Dr. George Papalambrou.

I want to emphasize how grateful I am to my instructor, for giving me the opportunity and the motivation to work on this subject that deeply interested me even before the start of this assignment and also for devoting his time into helping me achieve a very good outcome of which I am proud.

Furthermore I want to thank Professor Nikolaos Kyrtatos and Associate Professor Christos Papadopoulos for evaluating this assignment and for being a member of the audit committee at the day of the presentation.

At this point I want to sincerely express my heartfelt gratitude to PhD Candidate Nikolaos Planakis for his crucial assistance and wish him the best for his remaining and all his future endeavours.

With this chance, I wish to express my gratitude to friends for their support during these years of my studies, and to my fellow students with whom we had great cooperation and they were the driving force I needed to achieve better results.

Last but not least, I want to make a note of gratitude for everyone, that directly or indirectly, lent a hand in this venture. This accomplishment would not have been possible without them. Thank you.

Abstract

The purpose of this assignment is to develop a machine learning framework (LME-NNPower) based on neural networks that can be used for the online quasi-optimal control of HIPPO-2, the Hybrid Diesel Electric Marine Propulsion Plant of the Laboratory of Marine Engineering (LME).

Initially we discuss the inner workings of the predictive stage of the framework. To be more precise in this stage of the assignment a neural network was created for the classification of the near future Cruising Trend (NN-CRT) into six possible classes, by using features from the Pitch Angle Operation Profile. Then another neural network was employed for the prediction of the mid term Pitch Angle Pattern (NN-PAC) into five possible classes (patterns). Yet again this is done by extracted different features from the Pitch Angle Operation Profile.

Afterwards we examine the second stage of the framework, the control stage. Here for every pitch pattern we create a set of regression neural networks. One network (NN-ENG) is responsible for the torque command to the diesel engine and the other (NN-MOT) to the electric motor's command. With this approach we are able to emulate optimal control that is based on a Nonlinear Model Predictive Control scheme that was studied in previous work.

Finally a simulation scheme of the developed framework was established in the environment MATLAB Simulink. The simulations cover some crucial operation profiles and the framework was tested for it's ability to emulate the original NMPC controller and therefore conduct an effective power split between the two main drivers of a diesel-electric marine power plant.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The shipping industry fills a very important role in the modern world. It is regarded as one of the fundamentals of World Trade and Globalisation, as it is the engine that transports 90% of the global traded goods. Nevertheless the maritime industry is not invulnerable to the technological changes and it is projected that in the next decade the new advancements in technology will completely transform shipping.

Over the past two decades constructive effort has been rising exceptionally for higher efficiency and optimum performance in the maritime industry. While this endeavour is done with the purpose of keeping this leading position in the trading market, it has transformed modern ships into floating factories. A ship nowadays contains numerous machinery equipment such as engines, electronic networks and complex piping systems. All this machinery needs to be optimised and monitored to avoid hazardous outcomes. A lot of sources address this as a strategic business decision to maximise profits by reducing the energy consumption and loses. On the other hand there is a shift of humanity towards "greener" solutions in order to reduce the environmental impact of human industrial activities as a whole. In the context of this situation, the maritime industry must comply to increasingly stricter regulations, imposed by the legislative authorities around the industry. This desire has brought the operation of ships under examination with the aim to reduce the exhaust gases emissions primarily carbon dioxide (CO_2), nitrogen oxides (NO_x) and sulphur oxides (SO_x).

Since the November of 1973, the International Maritime Organization (IMO) has adopted a convention for the prevention of pollution from ships called MARPOL. But the specific Annex VI for the Prevention of Air Pollution was only added in 2005. These regulations are enforcing operational and design limitations to the maritime sector. An example of these, is the limit of NO_x emissions from engines that output at least $130kW$ and are built after 2011. To make matters harder, even lower limits are applied in specific sea areas of increased environmental "sensitivity". Furthermore Chapter 4, "Regulations on energy efficiency for ships" of Annex VI is imposing the mandatory Energy Efficiency Design Index (EEDI) for new ships and the Ship Energy Efficiency Plan (SEEMP) for all ships. This efficiency index depends on the total CO_2 related emissions that the ship would emit in order to complete the required transportation work. The ultimate goal is for ships constructed in 2030 to have a 30% reduced EEDI compared to 2013. Despite the aim of this regulation to increase the hull efficiency, the general trend is to satisfy it by increasing the efficiency of the propulsion plant. The vast majority of ships nowadays use diesel engines as their main propulsion drive power. Admittedly diesel engines played an integral role in the technological growth of humanity, but they are heavily pollutant machines that have already reached their peak thermal capabilities. Thus in order to cope with the increasing

aforementioned reductions more sophisticated technological advancements are needed for the future.

According to [1], a plethora of techniques exists regarding the task of reducing emissions from the operation of marine engines. This is done either by aiming towards increasing the efficiency of the hull and propeller or that of the installed power plant. The band of solutions that target the second proposal promise to reduce both emissions as well as fuel consumption by affecting directly or indirectly the existing diesel engines. Some direct techniques are the Exhaust Gas Recirculation System (EGR), the 'mini-sac' and the slide-type fuel valves introduced by MAN B&W Diesel and systems like intake air humidification. The most prevalent indirect technique is the Selective Catalytic Reduction System (SCR) like the SINOx by Siemens. Furthermore alternative fuels (e.g. LNG and biofuels) and renewable sources of power have also been proposed. Moreover a lot of progress has been shown in the field of batteries making the full battery depended ship, a viable solution for the future. A very interesting solution is that of a Hybrid Propulsion and Energy Conversion system.

1.1 The Hybrid Propulsion System

The Hybrid Propulsion system is developed to take advantage of the best from both worlds of propulsion, conventional thermal, like diesel, and electric to make a system better suited also for these types of vessels that have a flexible power demand. This is done by connecting an electrical motor and a diesel engine to the same gearbox giving the system a high degree of redundancy and flexibility. Optimised fuel economy in all operational modes and reduced investment cost compared to a full diesel electrical system is also achieved.

The electric motor and the directly connected diesel engine can run the propeller separately or in parallel. Typical operation for pure diesel mechanical setup is steaming. Typical operation for electrical set-up is transit at lower speed and dynamic positioning(DP) operation. Parallel boosting operation between mechanical and electrical is typically heavy towing, fast steaming and anchor handling operations.

A hybrid propulsion system can be realised with both azimuths and conventional propellers, but for many reasons the best layout is with conventional propellers and controllable pitch control. By the use of the conventional and well proven propeller system, it is possible to combine electrical and mechanical operation of the propeller. And, full advantage of both variable speed control and variable pitch can be taken.

Summarising the advantages of the Hybrid propulsion system's properties compared to conventional systems as follows:

- Improved redundancy: engines/gen-sets can be decoupled without major consequences to the operation of the vessel.
- Shaft generator/main engine can be operated independent of main gear.
- Partly improved utilisation of cargo space and more flexible overall ship design compared to conventional design. Shaft line is still there but the physical size of the main engine will be reduced and the auxiliary engine room can be located freely.
- Optimised fuel economy in all modes of operation, also transient, especially for vessels with fluctuating load demands. This is due to the optimised utilisation of the diesel engines and elimination of zero-pitch losses of propellers.
- Emissions are reduced due to the reduced fuel consumption
- Investment cost is mostly equivalent to a conventional system, but of course depending on layout and ratings.

The key factor in order to achieve respectable higher efficiency is the control strategy. For example, studies have shown that a $10-35\%$ fuel and emission reduction is possible in battery deployment and intelligent use of DC configurations by implementing appropriate control strategies [2]

1.2 The Need for Intelligent Engines

Smart shipping is the inclusive term coined by the industry to describe the digital technologies available for determining and optimising operational efficiency. Tightening margins and the affordability and availability of computing power have met to form opportune conditions for the adoption of smart shipping: weather-routing, voyage planning, fuel consumption, emissions control and predictive maintenance are popular options for improved commercial efficiency as well as meeting new regulatory standards.

The first intelligent engine in the maritime world was delivered in October-1998. Since then a lot of things have changed so it's only logical that there is again a spark of interest for the development of even smarter engines that use state-of-the-art technology.

Today the world is in a need of engines that can cope up with the stringent emission norms and also the higher demands for robust, reliable, smart and low cost operations. To achieve the above requirements, a whole new generation of engines is being developed with a comprehensive use of electronics, hardware and software in both two and four stroke engines known as "Intelligent Engines".

One of the greatest benefit of an intelligent engine is that it contains a central advanced control system that we can describe as the brain of the engine. This system is responsible for constantly monitoring and evaluating the working performance of the engine so that it maintains the operating parameters at an optimal level. These actions lead to unmatched performance levels at every working condition. The ultimate goal of the system is to achieve lower operational costs and it does that emphatically, so even if the initial cost of such an engine is higher the investment will eventually pay off.

Another great feature of an intelligent engine is the incorporation of reliable smart diagnostics like predictive alarms for all types of malfunctions. These diagnostics assist the on-board crew into acting predictively and not reactively, a very important aspect of hazard identification and prevention. Furthermore, in the long run, this feature helps a lot in reducing the working overhead and stress not only of the crew but also of the offshore office people because the engine's performance is "as new" for its lifetime.

1.3 Literature Overview

During recent years there is an expansion in the number of hybrid marine applications. These systems come with increased complexity due to the extra degrees of freedom that they have. Nonetheless, control strategies for marine applications are not very advanced and the control systems that are used are mostly based on classic control logic, e.g. rule based control according to [2] or heuristic rules/fuzzy logic for control algorithm development according to [3]. However, research has shown that conservative control strategies fail to provide a significant fuel and emission reduction when they are employed to control advanced system arrangements.

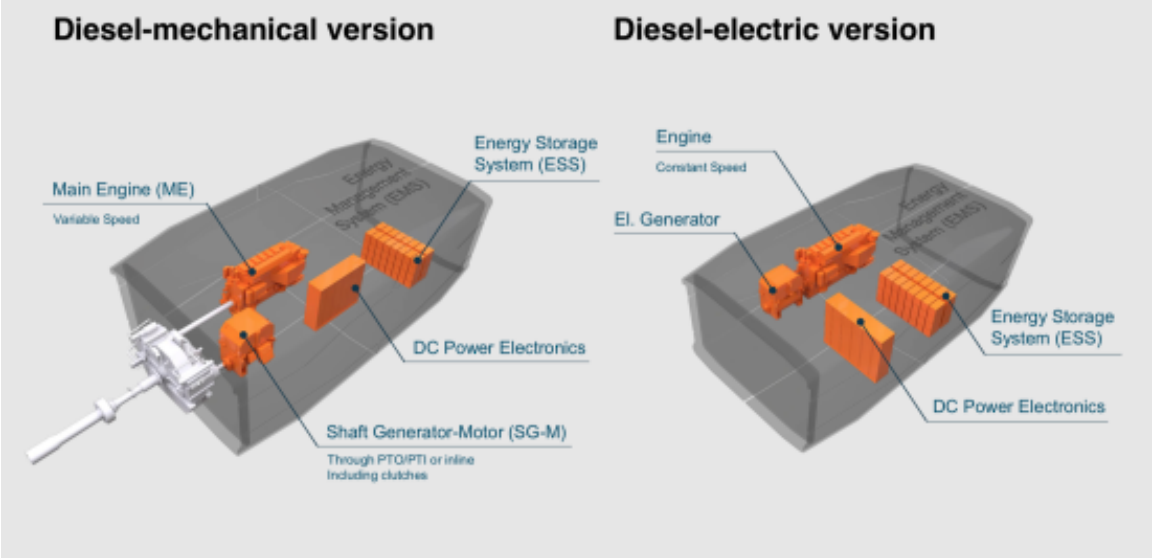


Figure 1.1: Wärtsilä HY Hybrid Modules

Hybrid propulsion has many configurations, two of them that can be seen in Fig. 1.1. The most prevalent configuration has a conventional main propulsion unit, such as a diesel engine and the electric motor is directly connected to the main shaft line. The theory behind this arrangement is for the diesel engine to supply the needed propulsive power in higher speeds and for the electric motor to assist in lower speeds where the diesel efficiency is critically low.

Another interesting application which refers to parallel operation of the electric motor and diesel engine is presented in [**?**], which the motor assists the engine so as to maintain a specific air-fuel ratio λ reference. In that way, during transient operations, the thermal loading of the engine is decreased and consequently the NOx emissions drop. In [4], the above is implemented via Model Predictive Control.

A different implementation besides diesel engines and diesel generators, is the usage of batteries as secondary power source. In this way, partial and low load are handled by the electric part with high efficiency while the diesel part manages the higher loads. Two approaches are conceived for this scheme. The first is referred as heuristic control strategy, in which the battery charge is provided offshore. In [5], a rule based strategy which greatly reduce the fuel consumption for hybrid harbour tugs is suggested. The other approach suggests that the battery should be recharged during the operation by the primary mover. According to [5], an equivalent fuel consumption strategy is proposed which aims to drop the fuel consumption, by applying linear programming. In [6] a NMPC control scheme of the aforementioned approach is implemented thoroughly.

1.4 Motivation and Structure of the study

In the present study, we investigate the implementation of a control strategy for a *hybrid diesel-electric ship propulsion* plant with the use of Artificial Neural Networks (ANNs) that we based on a Nonlinear Model Predictive Control Scheme. The control strategy is examined with respect to minimising the produced emissions during transient loads.

More specifically, the developed framework is employed for the closed-loop control of a simulation model that is based on the Hybrid powertrain HIPPO-2 housed in the Laboratory of Marine Engineering (LME) of the National Technical University of Athens (NTUA).

The framework comprises of two stages. The prediction stage and the control stage. The prediction stage contains two neural networks, one that predicts the Current Pitch Angle Pattern and another one that predicts the Cruising Trend of the Engine. Base on the pitch angle pattern prediction the control stage kicks in with a set of neural networks that are employed to emulate Nonlinear Model Predictive Control (NMPC) a technique that has been used in the past to great success. Five pitch angle patterns exist, therefore five sets of neural networks are used.

The motivation of this study comes from the fact that MPC solves a constrained quadratic-programming (QP) optimisation problem in real time based on the current state of the plant. Since MPC solves its optimisation problem in an open-loop fashion, there is the potential to replace the controller with a trained deep neural network. Doing so is an appealing option, since evaluating a deep neural network can be more computationally efficient than solving a QP problem in real-time.

The structure of the thesis is as follows: Firstly a brief description of the test bed HIPPO-2 and the previously developed simulation model are introduced in Chapter 2. Then a theoretic background regarding neural networks is presented in Chapter 3. Next, in Chapter 4 we cover the used data for our experiments as well as the data pre-processing techniques that we used to improve the overall performance of the proposed method. Subsequently in Chapter 5 we the developed framework of neural networks as well as the training results from all the explored different models. In Chapter 6 the results from the simulated control scheme are presented and finally in Chapter 7 we provide our conclusions and propose future work in order to improve the framework as well as different possibilities that have not been studied here.

Chapter 2

The Experimental Facility

In this chapter we present the experimental hybrid powertrain facility HIPPO-2, based on which we created the simulation environment where the final tests of the presented control method took place.

The powering units of the facility are the *Internal Combustion Engine (ICE)* and the *Electric Motor/Generator (EMOT), while the load torque is applied by the Electric Motor (EB)*. For this study we incorporated a *Virtual Battery (B)* which was simulated in parallel during the experimental test via the control platform. The Neural Network approach of this thesis is based on a Model Predictive Control (MPC) scheme that requires models for the aforementioned components so as to simulate and solve the optimization control problem. These models can be found in Karystinos [6].

2.1 HIPPO-2

The HIPPO2 Fig. 2.1 contains:

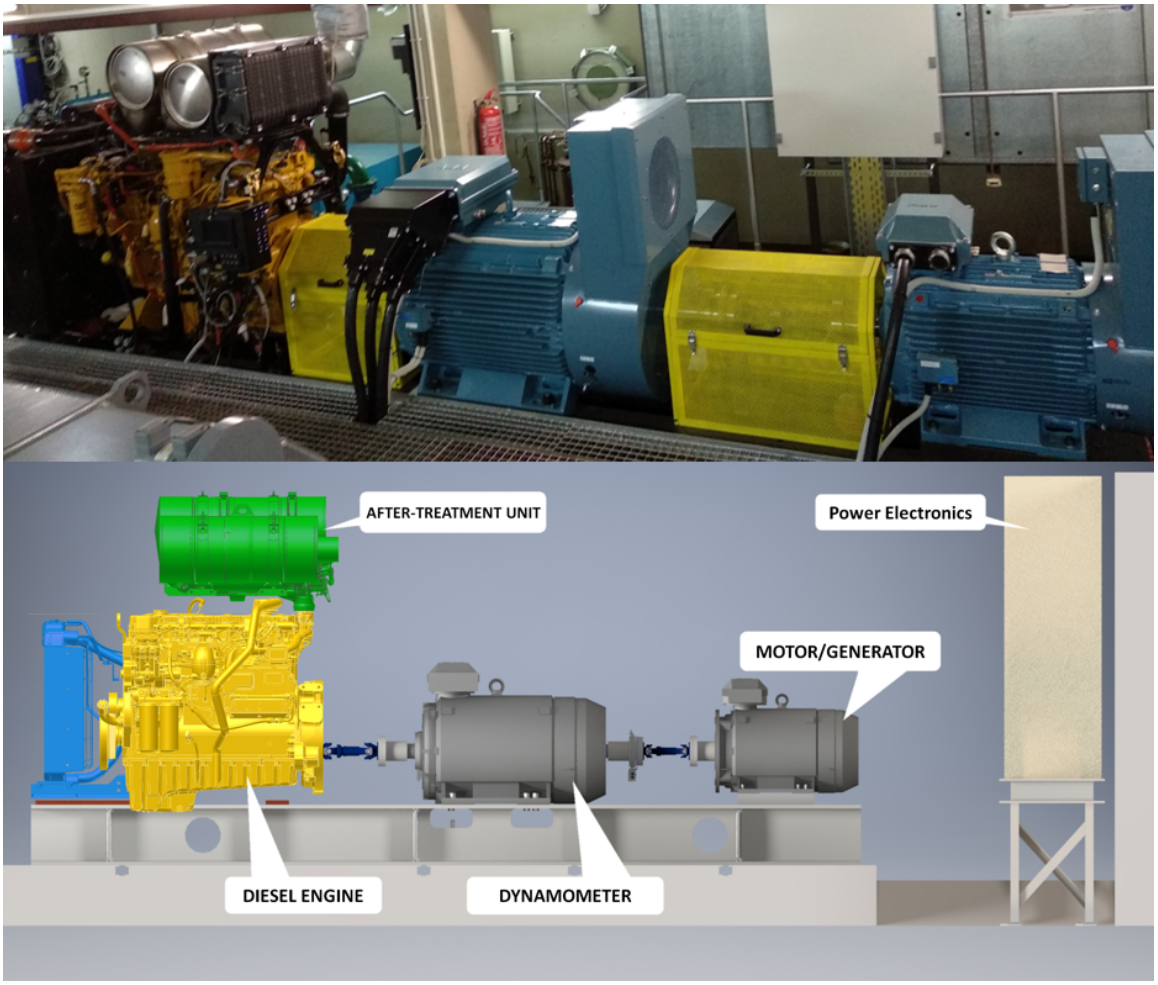


Figure 2.1: HIPPO-2

- an internal combustion engine with the following characteristics:

Table 2.1: The characteristics of the internal combustion diesel engine

Caterpillar	C9.3 Tier IV
Power(MCR)	261 kW at 1800-2200 rpm
Peak Torque	1596 Nm at 1400 rpm
Mass Inertia	12.047 $Kg \cdot m^2$

The load diagram of the engine can be seen in 2.2, emphasizing at Rating C where the engine is operating. According to the needed rotational speed and it's deviation from the real time speed that is measured from the appropriate sensor, the Electronic Control Unit (ECU) of the engine controls the amount of the injected fuel into the cylinders with a closed loop system, using a controller based on engine maps.

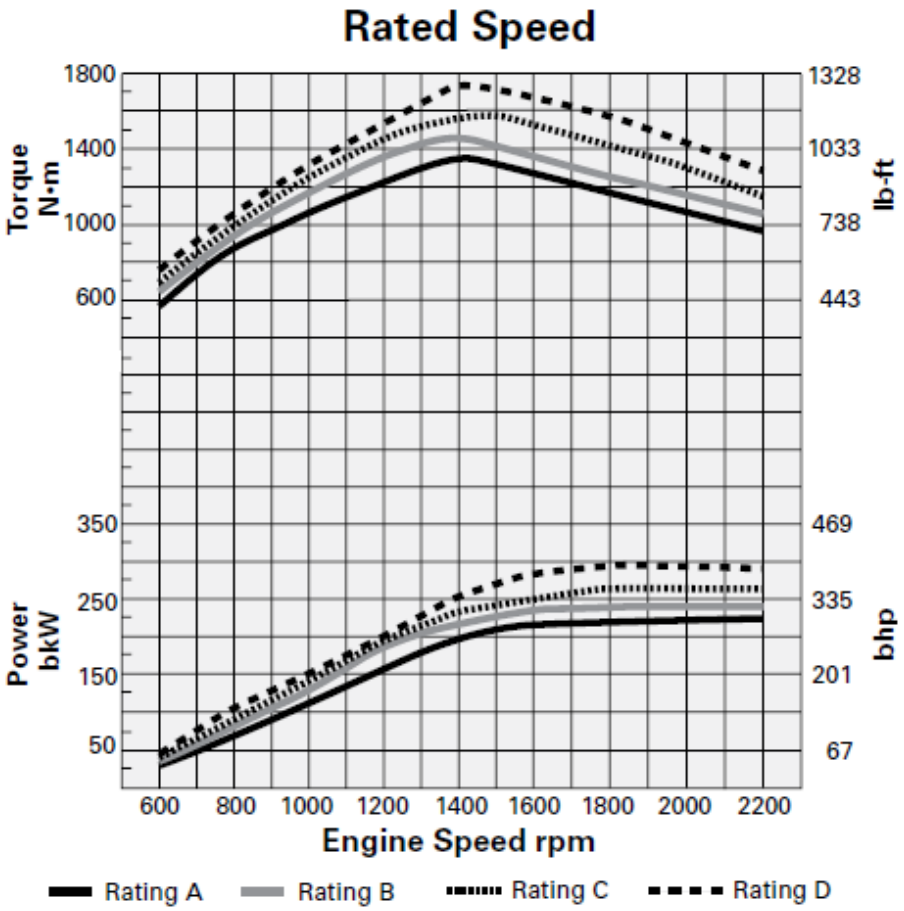


Figure 2.2: Caterpillar C9.3 Loading Diagram

Speed Range

Rating	Aspiration	Rated Speed rpm	Rated Power bkW	Rated Power bhp	Speed rpm	Peak Torque N·m	Peak Torque lb-ft
A	TA	2200	224	300	1400	1369	1009
B	TA	2200	242	325	1400	1484	1095
C	TA	2200	261	350	1400	1596	1177
D*	TA	2200	290	389	1400	1719	1268

*298 bkW (400 bhp) @ 2000 rpm also available

Figure 2.3: Caterpillar C9.3 Available Ratings

The Caterpillar C9.3 Engine is a Tier IV engine with respect to the emissions standards and therefore it is following the strictest rules regarding that matter. To achieve this level of emissions reduction, it incorporates systems like Selective Catalytic Reduction (SCR) that spray urea (ammonia) on the exhaust gases to reduce NOx, Exhaust Gas Recirculation (EGR) that returns some of the exhaust gas mass back into the cylinder in order to reduce the combustion temperature of the engine and therefore to further reduce NOx emissions and finally it also has Diesel Particulate Filter (DPF) for soot (mass of impure carbon particles resulting from the incomplete combustion process)

- an electric brake with the following characteristics:

  The electric brake is installed to simulate the load of the propeller in real life situations. The load of the propeller can be broken into a hydrodynamic part and an inertial one.

Table 2.2: The characteristics of the electric brake

ABB	315kW Low Voltage Cast Iron Induction Motor
Power(MCR)	315 kW at 1488-2200 rpm
Peak Torque	2021 Nm at 1488 rpm
Moment of inertia	6.9 kgm^2

The hydrodynamic is the torque needed from the propeller in order to keep the needed rotating speed and the thrust. The inertial is the torque that the propeller needs in order to catch up to the rotational speed of the shaft system with the addition of frictional losses from the gearing system and the rest of the shaft system of the ship. The characteristics of the electric brake can be seen in Table.2.2.

- and electric motor/generator unit with the following characteristics:

Table 2.3: The characteristics of the electric motor/generator

ABB	90kW Low Voltage Cast Iron Induction Motor
Power(MCR)	90 kW at 1483-2200 rpm
Peak Torque	579 Nm at 1483 rpm
Moment of inertia	1.73 kgm^2

The electric motor/generator can be used when the required torque by the brake is more than what the diesel engine can give on it's own, or it can be used when we need to store energy for future use. The motor/generator can be seen at Table.2.3.

The diesel engine, the brake and the motor/generator are serially connected and the range of rotation speed is from 600 rpm to 2200 rpm..

The output speed and torque for the brake and for the electric motor are controlled by the ABB drives. The type of ABB drives that is being used for the brake and for the motor is the ACS800. Using the ACS800 and the CANopen communication protocol we can collect the necessary data and therefore effectively control the brake and the motor.

Furthermore there is a torque meter between the brake and the motor which is used to measure the rotational speed of all three connected machines and the torque on the shaft between the brake and the motor.

2.2 Propeller Load Simulation Model

In this thesis for validation purposes, we used the previously developed tug vessel model with a Controllable Pitch Propeller (CPP) from [7]. In Fig. 2.4 we show this model in the MATLAB Simulink environment.

The purpose of this high fidelity simulation environment is to provide a base for simulating loads on the HIPPO-2 hybrid testbed. It encompasses a ship model for the surge movement of a tug vessel, a reduction gear model, a controllable pitch propeller model as well as a probabilistic unidirectional wave model to fill the role of the environmental disturbance. Using this system we can only simulate loading conditions of tug vessels as the power capabilities of the testbed are limited.

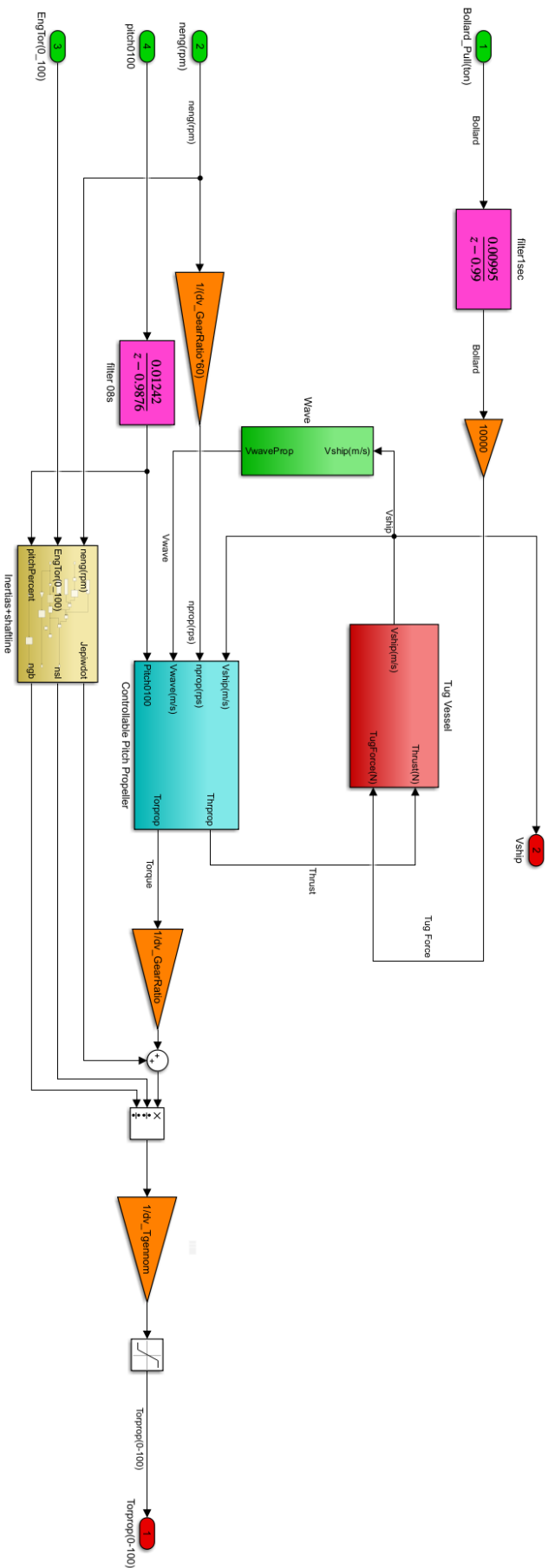


Figure 2.4: Controllable Pitch Propeller Model in MATLAB Simulink

Chapter 3

Neural Networks Theory

In this chapter we will introduce some basic theory regarding the artificial neural networks ANNs.

Artificial Neural Networks are used as statistical models and therefore we will mention some basic statistical concepts that are going to be prevalent in the whole study. Afterwards we will briefly describe the predecessor of ANNs, the Perceptron and finally we will describe the main constituents of a neural network from architecture to functionality and training.

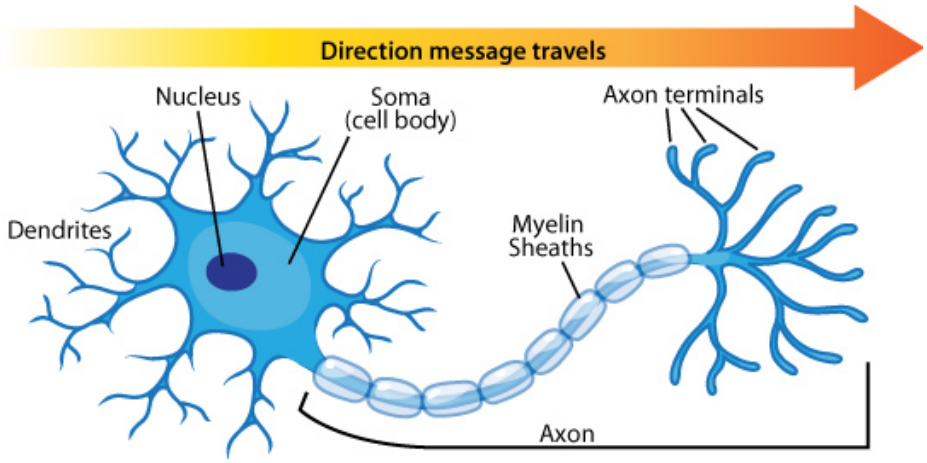


Figure 3.1: An illustration of the biological neuron with some of its biological features.

3.1 Artificial Neural Networks

The most important activity of a "developing" neuron is its ability to adapt to the surrounding environment and in the same way that this is essential to the proper functioning of biological neurons as information-processing units in the human brain, so is with neural network made up of artificial neurons. In its most general form, a *neural network* is a machine that is designed to *model* the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as "neurons". According to Haykin [8] the following definition of a neural network viewed as an adaptive machine:

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. *Knowledge is acquired by the network from its environment through a learning process.*
2. *Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.*

3.2 Statistical Modelling Concepts

In the present study we face two different problems where in order to solve them we will employ neural networks that apply two different machine learning concepts. The concept of classification and regression. While every machine learning problems is in it's core a regression problem, we find it instructive to make the distinction clear.

3.2.1 Regression Concept

In statistical modelling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analysing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer causal relationships between the independent and dependent variables. However this can lead to illusions or false relationships, so caution is advisable.

A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:

- *How many dBs is the self-noise that is generated by an aerofoil reacting to it's own turbulent boundary layer at high Reynold Numbers?*
- *What is the energy efficiency of a building as a function of its parameters?*
- *What is the fuel consumption of a ship's engine based on past reports?*

The two most common metrics for evaluating a regression model are the Mean Squared Error (MSE) measure and the R^2 measure. If x_i is the truth and $\hat{x}_i$ is the model output, for sample i, the error is given by:

$$\epsilon_i = x_i - \hat{x}_i \tag{3.1}$$

then the Mean Squared Error has the following definition:

$$MSE = \frac{1}{n}\sum_{i}^{n} \epsilon_i^2 \tag{3.2}$$

and R^2:

$$R^2 = 1 - \frac{MSE}{Var(x)} \tag{3.3}$$

In a narrower sense, regression may refer specifically to the estimation of continuous response (dependent) variables, as opposed to the discrete response variables used in classification [9]. The case of a continuous dependent variable may be more specifically referred to as metric regression to distinguish it from related problems [10].

3.2.2 Classification Concept

When working with statistics and other areas where large amounts of data are collected and analysed, it is often necessary to sort the data points into sub-groups. There are several different methods of creating a digital classification machine or classifier, a neural network is one of them.

The common ground for all classifiers is that they work by supervised learning, where the classifier is trained on data with known outcomes and then is used for inference on similar but unseen data. The **accuracy** of the network is one of the available metrics for evaluating it and it describes the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions} \tag{3.4}$$

Another metric that can be used is **precision** which attempts to answer the following question: *What proportion of positive identifications was actually correct?*. Precision is defined by:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{3.5}$$

In the case of multiclass logistic regression (multi-classification) we can create a cross precision confusion matrix.

In summary a classification model predicts discrete values that are used as labels of mutually exclusive sub categories(classes). For example, classification models make predictions that answer questions like the following:

- *Is a certain cargo vessel risky for insurance based on its characteristics?*
- *Is the auto-landing condition of a spacecraft preferable or should the pilot switch to manual?*

3.3 Rosenblatt's Perceptron

One very simple but very important linear discrimination model is the perceptron of Rosenblatt (1958), which occupies an important place in history of pattern recognition algorithms. It corresponds to a two-class model in which the input vector $\mathbf{x}$ is first transformed using a fixed non-linear transformation(function) to give a feature vector $\phi(\mathbf{x})$, and this is then used to construct a generalized linear model of the form:

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x})) \tag{3.6}$$

Where the non-linear function $f(\cdot)$ is given by a step function of the form:

$$f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases} \tag{3.7}$$

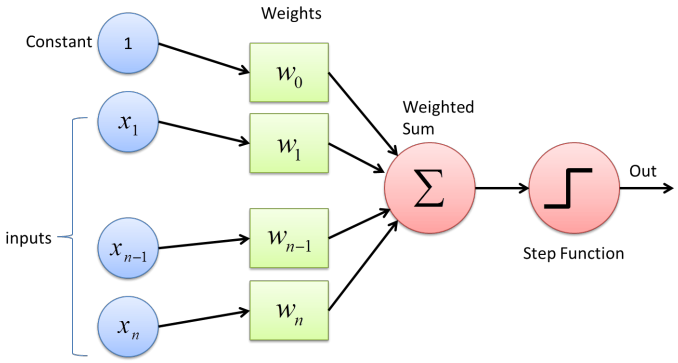


Figure 3.2: Rosenblatt's Perseptron. The basic elements consisting the perceptron's model are the inputs, the weights, the net input, the activation function and the output.

The vector $\phi(\mathbf{x})$ will typically include a bias component $\phi_0(\mathbf{x}) = 1$. In a discussion of two-class classification problems, we have focussed on a target coding scheme in which $t \in \{0, 1\}$, which is appropriate in the context of probabilistic models. For the perceptron it is more convenient to use target values $t = +1$ for class C_1 and $t = -1$ for class C_2, which matches the choice of activation function.

The algorithm used to determine the parameters $\mathbf{w}$ of the perceptron can most easily be motivated by error function minimization. A natural choice of error function would be the total number of misclassified patterns. However, this does not lead to a simple learning algorithm because the error is a piecewise constant function of $\mathbf{w}$, with discontinuities wherever a change in $\mathbf{w}$ causes the decision boundary to move across one of the data points. Methods based on changing $\mathbf{w}$ using the gradient of the error function cannot be applied, because the gradient is zero almost everywhere.

We therefore consider an alternative error function known as the *perceptron criterion*. To derive this, we note that we are seeking a weight vector $\mathbf{w}$ such that patterns $\mathbf{x}_n$ in class C_1 will have $\mathbf{w}^T \phi(\mathbf{x}_n) > 0$, whereas patterns $\mathbf{x}_n$ in class C_2 have $\mathbf{w}^T \phi(\mathbf{x}_n) < 0$. Using the $t \in \{-1, +1\}$ target coding scheme it follows that we would like all patterns to satisfy $\mathbf{w}^T \phi(\mathbf{x}_n) t_n > 0$. The perceptron criterion associates zero error with any pattern that is correctly classified, whereas for a misclassified pattern $\mathbf{x}_n$ it tries to minimize the quantity $-\mathbf{w}^T \phi(\mathbf{x}_n) t_n$. The perceptron criterion is therefore given by:

$$E_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\phi(\mathbf{x}_n)t_n \tag{3.8}$$

where $\mathcal{M}$ denotes the set of all misclassified patterns. The contribution to the error associated with a particular misclassification pattern is a linear function of $\mathbf{w}$ in regions of $\mathbf{w}$ space where the pattern is misclassified and zero in regions where it is correctly classified. The total error function is therefore piecewise linear.

We now apply the stochastic gradient descent algorithm to this error function.

The change in the weight vector $\mathbf{w}$ is then given by:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta\phi_n t_n \tag{3.9}$$

where η is the learning rate parameter and τ is an integer that indexes the steps of the algorithm. Because the perceptron function $y(\mathbf{x}, \mathbf{w})$ is unchanged if we multiply $\mathbf{w}$ by a constant, we can set the learning rate η to 1 without loss of generality. Note that, as the weight vector evolves during training, the set of patterns that are misclassified will change.

The perceptron learning algorithm has a simple interpretation, as follows. We cycle through the training patterns in turn, and for each pattern we evaluate the perceptron function. If the pattern is correctly classified, then the weight vector remains unchanged, whereas if it is incorrectly classified, then for class C_1 we add a vector $\phi(\mathbf{x}_n)$ onto the current estimate of weight vector $\mathbf{w}$ while for C_2 we subtract the vector $\phi(\mathbf{x}_n)$ from $\mathbf{w}$. The perceptron learning algorithm is illustrated in Fig. 3.3

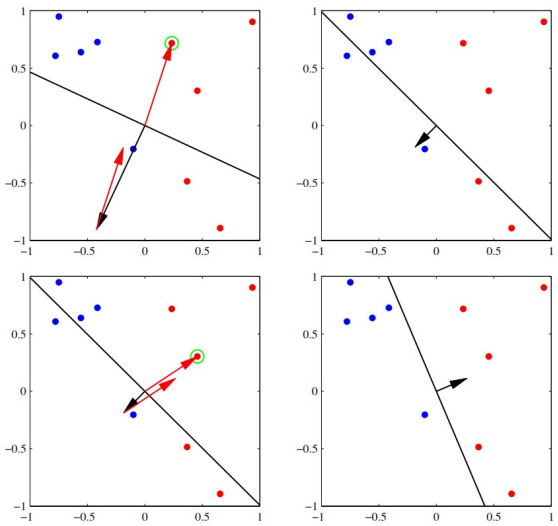


Figure 3.3: The convergence of the perceptron learning algorithm showing data from two classes (red and blue) in a two dimensional feature space (ϕ_1, ϕ_2). ([9, Bishop 2006])

Aside from difficulties with the learning algorithm the perceptron does not provide probabilistic outputs, nor does it generalize readily to $K > 2$ classes. The most important limitation, however, arises from the fact that it is based on linear combinations of fixed basis functions.

3.4 Neural Network and Training

The linear models for regression and classification are based on linear combinations of fixed non-linear basis functions $\phi_j(\mathbf{x})$ and take the form:

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) \tag{3.10}$$

where $f(\cdot)$ is a non-linear activation function in the case of classification and the identity in the case of regression. Our goal is to extend this model by making the basis functions $\phi_j(\mathbf{x})$ depend on parameters and then to allow these parameters to be adjusted, along with coefficients $\{w_j\}$, during training. There are many ways to construct parametric non-linear basis functions. Neural networks use basis functions that follow the same form as (3.10), so that each basis function is itself a non-linear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

This leads to the basic neural network model, which can be described as a series of function transformations. First we construct M linear combinations of the input variables $x_1, ..., x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3.11}$$

where $j = 1, ..., M$, and the superscript (1) indicates that the corresponding parameters are in the first "layer" of the network. We shall refer to the parameters $w_{ji}^{(1)}$ as *weights* and the parameters $w_{j0}^{(1)}$ as *biases*. The quantities a_j are known as *activations*. Each of them is then transformed using a differentiable, non-linear *activation function* $h(\cdot)$ to give

$$z_j = h(a_j) \tag{3.12}$$

These quantities correspond to the outputs of the basis function that in the context of neural networks, are called *hidden units*. The non-linear functions $h(\cdot)$ are generally chosen to be sigmoidal functions such as *logisticsigmoid* or the *tanh* function. Following the basis function equation, these values are again linearly combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{3.13}$$

where $k = 1, .., K$, and K is the total number of outputs. This transformation corresponds to the second layer of the network, and again $w_{k0}^{(2)}$ are bias parameters. Finally, the output unit functions are transformed using an appropriate activation function to give a set of network outputs y_k. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables and follows the same considerations as for linear models. Thus for standard regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$y_k = \sigma(a_k) \tag{3.14}$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{3.15}$$

Finally, for multi-class problems, an activation function of the below form is used

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)} = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{3.16}$$

which is known as the normalized exponential and can be regarded as a multi-class generalization of the logistic sigmoid. Here the quantities a_k are defined by

$$a_k = \ln p(\mathbf{x}|C_k)p(C_k) \tag{3.17}$$

The normalized exponential is also known as the *softmax function* as it represents a smoother version of the max function because, if $a_k \gg a_j$ for all $j \neq k$, then $p(C_k|\mathbf{x}) \simeq 1$, and $p(C_j|\mathbf{x} \simeq 0)$.

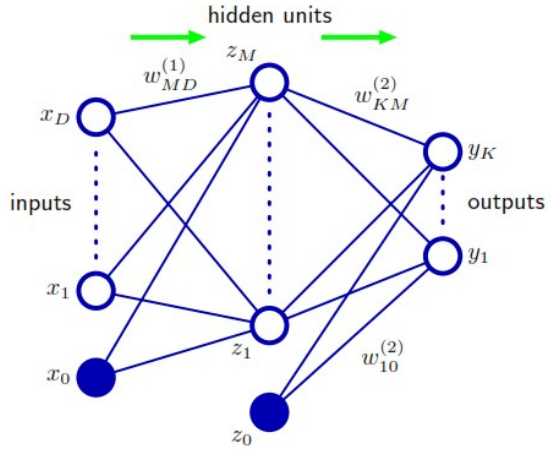


Figure 3.4: Network diagram for the two layer neural network. ([9, Bishop 2006])

We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{3.18}$$

where a set of all weight and bias parameters have been grouped together into a vector $\mathbf{w}$. Thus the neural network model is simply a non-linear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\mathbf{w}$ of adjustable parameters.

This function can be represented in the form of a network diagram as shown in Fig 3.4. The process of evaluating the network function can be interpreted as a *forward propagation* of information through the network.

At this point we can absorb the bias parameters into the set of weight parameters by defining an additional input variable x_0 whose value is clamped at $x_0 = 1$, so that

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i. \tag{3.19}$$

We can similarly absorb the second-layer biases into the second-layer weights, so the overall network function becomes

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right) \tag{3.20}$$

As it can be seen from Fig 3.4, the neural network model comprises two stages of processing, each of which resembles the perceptron model and for that reason the neural network is also called *multilayer perceptron*, or MLP. A key difference compared to the perceptron is that a neural network uses continues sigmoidal non-linearities in the hidden units, whereas the perceptron uses step-function non-linearities. This means that the neural network function is differentiable with respect to the network parameters, and this property will play a central role in network training.

The network architecture shown in Fig 3.4 is the most commonly used one in practice. However, it is easily generalized, for instance by considering additional layers of processing each consisting of a weighted linear combination of the form (3.13). Due to some confusion in the literature we will call a neural network of the Fig 3.4 to be called a two-layer network, because it is the number of layers of adaptive weights that is important for determining the network properties [9].

3.4.1 Activation Functions and Error Functions

The process of training an ANN refers to the estimation of the adaptive parameters that lead to the minimum error. Summarizing all that we have discussed, a neural network could be perceived as a parametric nonlinear function that given a set of input vectors $\mathbf{x}_n$ where $n = 1, 2, \ldots, N$ together with a corresponding set of target values $\mathbf{t}_n$. What we wish to achieve is for this approximation to be as accurate as possible and this can be expressed in mathematical form with an error function:

$$\mathscr{E}(\hat{\mathbf{w}}) = \frac{1}{2} \sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n||^2 \tag{3.21}$$

However we can provide a much more general view of network training by first giving a probabilistic interpretation to the network outputs.

3.4.1.1 Regression Problems

We start by discussing regression problems and particularly one where we have a single target variable $t \in R$. We assume that t has a Gaussian distribution with an $\mathbf{x}$-dependent mean, which is given by the output of the neural network so that

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}, \beta^{-1}) \tag{3.22}$$

where β is the precision (inverse variance) of the Gaussian noise. For the conditional distribution in Eq.(3.22) we assume that the network's output function is the identity,

because such a network can approximate any continuous function from $\mathbf{x}$ to y. Suppose we are given a data set of N independent and identically distributed observations $X = x_1, \ldots, x_N$ and the corresponding target values $t = t_1, \ldots, t_N$, we construct the likelihood function :

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x_n}, \mathbf{w}, \beta) \tag{3.23}$$

Taking the negative logarithm we obtain the error function:

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}ln\beta + \frac{N}{2}ln(2\pi) \tag{3.24}$$

In the neural network literature it is usual to consider the minimization of the error function instead of the maximization of the (log)likelihood and we will also follow this convention. Consider first the determination of $\mathbf{w}$. Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\}^2 \tag{3.25}$$

where additive and multiplicative terms have been discarded. The value of $\mathbf{w}$ that minimized the error function will be denoted as $\mathbf{w}_{ML}$ because it corresponds to the maximum likelihood solution. The error function is nonconvex due to the nonlinearity of the network's function $y(\mathbf{x}, \mathbf{w})$ and thus in practice we can find local maxima of the likelihood function that correspond to local minima of the error function. Assuming $\mathbf{w}_{ML}$ is found we can then find β from (3.24) as:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)\}^2 \tag{3.26}$$

If we consider the case of K(multiple) target variables and we assume that they are independent conditional on $\mathbf{x}$ and $\mathbf{w}$ with shared noise precision β, then the conditional probability distribution of the target values is given by:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|y(\mathbf{x}, \mathbf{w}, \beta^{-1}\mathbf{I}) \tag{3.27}$$

Following the same process and the same assumptions we get that the noise precision is now given by:

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}||y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)||^2 \tag{3.28}$$

Furthermore, we shall introduce one more essential equation that occurs from the natural pairing of the error function and the output unit activation function. In the regression case the output activation function is the identity and therefore the error function is the sum-of-squares functions so we have:

$$\frac{\partial\mathscr{E}}{\partial a_k} = y_k - t_k \tag{3.29}$$

3.4.1.2 Classification Problems

Now for the case of binary classification in which we have a single target variable t such that $t = 1$ denotes class $\mathcal{C}_1$ and $t = 0$ denotes class $\mathcal{C}_2$. We consider a network having a single output whose activation function is a logistic sigmoid

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)} \tag{3.30}$$

so that $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$. We can interpret $y(\mathbf{x}, \mathbf{w})$ as the conditional probability $p(\mathcal{C}_1|\mathbf{x})$, with $p(\mathcal{C}_2|\mathbf{x})$ given by $1 - y(\mathbf{x}, \mathbf{w})$. The conditional distribution of targets given inputs is then a Bernoulli distribution [9] of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t} \tag{3.31}$$

If we consider a training set of independent observations, then the error function, which is given by the negative log likelihood, is then a *cross-entropy* error function of the form

$$\mathscr{E}(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{3.32}$$

We can see that there is no analogue of the noise precision β because the target values are assumed to be correctly labelled. According to Simard *et al.*(2003) found that using the cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization.

If we have K separate binary classifications to perform, then we can use a network having K outputs each of which has a logistic sigmoid activation function. Associated with each output is a binary class label $t_k \in \{0, 1\}$, where $k = 1, \ldots, K$. If we assume that the class labels are independent, given the input vector, then the conditional distribution of the targets is

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} [1 - y_k(\mathbf{x}, \mathbf{w})]^{1-t_k} \tag{3.33}$$

Taking the negative logarithm of the corresponding likelihood function then gives the following error function

$$\mathscr{E}(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{3.34}$$

Same as the regression case, the derivative of the error function with respect to the activation for a particular unit takes the form of (3.29).

Finally, we consider the standard multiclass classification problem in which each input is assigned to one of K mutually exclusive classes. The binary target variables $t_k \in \{0, 1\}$ have a 1-of-K coding scheme indicating the class, and the network outputs are interpreted as $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$, leading to the following error function

$$\mathscr{E}(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w}) \tag{3.35}$$

We can see that the output unit function is given by the softmax function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \tag{3.36}$$

which satisfies $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$. Note that $y_k(\mathbf{x}, \mathbf{w})$ are unchanged if a constant is added to all the $a_k(\mathbf{x}, \mathbf{w})$, causing the error function to be constant for some directions in weight space. This degeneracy is removed if an appropriate regularization term is added to the error function.

Once again the derivative of the error function with respect to the activation for a particular output unit takes the same form of (3.29).

Summarizing, there is a natural choice of both output unit function and matching error function, according to the type of problem we are trying to solve. For regression we use linear outputs and sum-of-squares error, for (multiple independent) binary classification we use logistic sigmoid outputs and a cross-entropy error function, and for multiclass classification we use softmax outputs with the corresponding multiclass cross-entropy error function.

3.4.2 Parameter Optimization with Backpropagation

We can now consider the issue of determining a weight vector $\mathbf{w}$ that minimizes the defined error function. One can imagine the error function as a surface on the weight space, where the components of the vector $\mathbf{w}$ correspond to the main axis of space. Finding the coordinates of local or global minima on the surface is the solution to our problem. Consider an initial position for the vector $\mathbf{w}$ and then a small step in the weight space from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$. The change in the error function could then be approximated as: $\delta\mathscr{E} \simeq \delta\mathbf{w}^T \nabla\mathscr{E}(\mathbf{w})$, where the vector $\nabla\mathscr{E}(\mathbf{w})$ is the gradient of the error function at this point. Because the function $\mathscr{E}(\mathbf{w})$ is a smooth continuous function of $\mathbf{w}$, when we get to the point where the gradient vanishes i.e. $\nabla\mathscr{E}(\mathbf{w}) = 0$, we have encountered a stationary point which may be a minima, maxima or saddle point. This happens because otherwise we could take a step in the direction of $-\nabla\mathscr{E}(\mathbf{w})$ and thereby reduce even further the error. The described geometric view of the problem is illustrated in Fig.3.5

The main issue we are dealing with is that the error function has a highly nonlinear dependence on the weights and bias parameters and that means a severely complex surface for the error function with a lot of points where the gradient vanishes. It can be proven from the symmetrical properties of the neural network that for any point $\mathbf{w}$, that is a local minimum, there exist numerous other points in the weight space that are equivalent minima. For a FNN with two-layers for each such point exist $M!2^M$ equivalent points. Since close form analytic solutions for so complex functions are impossible to be found we shall resort to iterative numerical solutions. The optimization of continuous nonlinear functions is a widely studied problem and there exists an extensive literature on the matter. Most techniques involve the following three steps

- Initialization of the weight vector $\mathbf{w}$ to $w^{(0)}$
- Successive steps through the weight space in the form of $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$
- Utilization of gradient information for the update of the weights, i.e. $\Delta\mathbf{w}^{(\tau)}$ is a function of $\nabla\mathscr{E}(\mathbf{w})$

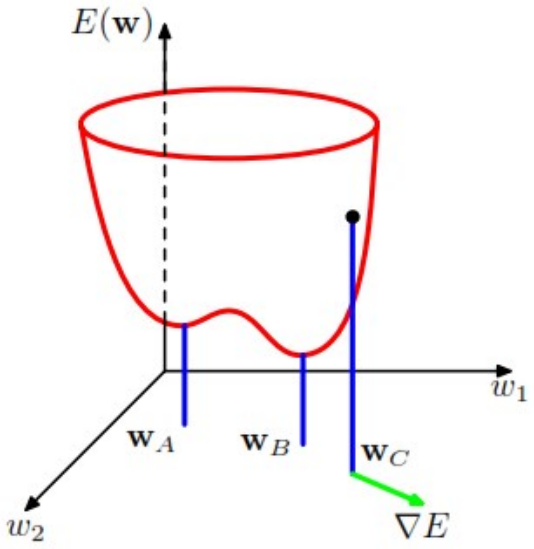


Figure 3.5: Geometric view of the error function $\mathscr{E}(\mathbf{w})$ as a surface on the weight space. ([9, Bishop 2006])

The idea of utilizing the gradient information seems pretty effective but that information is not always simple to accumulate. It is often computationally demanding to get a precise value of the gradient of the error function and thus we resolve to approximations. A satisfactory approximation of $\nabla\mathscr{E}(\mathbf{w})$ can be produce with a Taylor expansion of $\mathscr{E}(\mathbf{w})$ around $\mathbf{w}$ up to first or second terms.

Based on the analysis so far, a few more essential elements that are utilized in the majority of the ANN training algorithms shall be introduced. For instance, many training algorithms that make use of the gradient information further modify $\Delta\mathbf{w}^{(\tau)}$ by inserting a positive scaling factor η and so we have

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta\nabla\mathscr{E}(\mathbf{w}^{(\tau)}) \tag{3.37}$$

This parameter η is known as the *learning rate* and is responsible for controlling the step size in the weight space. In some algorithms it has a constant value while in others it is adjustable in order to escape from regions with vanishing gradient.

Furthermore the gradient of the error function is actually a function of the derivative of the error with respect to every weight w_{ji} as the value $\frac{\partial\mathscr{E}}{\partial w_{ji}}$. By considering the definition of the derivative of a function we get the meaning of $\frac{\partial\mathscr{E}}{\partial w_{ji}}$, which is how much will a small change in w_{ji} affect the value of the error function $\mathscr{E}$.

Many error function of practical interest comprise a sum of terms, one for each data point in a training set, so that

$$\mathscr{E}(\mathbf{w}) = \sum_{n=1}^{N} \mathscr{E}_n(\mathbf{w}) \tag{3.38}$$

Here we will consider the problem of evaluating $\nabla\mathscr{E}_n(\mathbf{w})$ for one such term in the error function.

If we consider a simple linear model with it's outputs y_k as linear combinations of the input variables x_i so that

$$y_j = \sum_i w_{ki} x_i \tag{3.39}$$

together with an error function that for the particular input pattern n takes the form

$$\mathscr{E}_n(\mathbf{w}) = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{3.40}$$

The gradient of this error function with respect to a weight w_{ji} is given by

$$\frac{\partial \mathscr{E}}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{3.41}$$

which can be interpreted as a 'local' computation involving the product of an 'error signal' $y_{nj} - t_{nj}$ associated with the output end of the link w_{ji} and the variable x_{ni} associated with the input end of the link. The same formula arises with a logistic sigmoid activation function together with the cross-entropy error function and similarly for the softmax activation function together with its matching cross-entropy error function. Next we see how this simple result extends to the more complex setting of multilayer feed-forward networks.

In a general feed-forward network, each unit computes a weighted sum of its inputs in the form

$$a_j = \sum_i w_{ji} z_i. \tag{3.42}$$

where z_i is the activation of a unit, or input, that send a connection to unit j and w_{ji} is the weight associated with the connection. We will not deal with biases explicitly as they can be grouped with weights as we saw earlier. The sum in (3.42) is then transformed by a nonlinear function $h(\dot{})$ to give the activation z_j of unit j in the form

$$z_j = h(a_j) \tag{3.43}$$

Now consider the evaluation of $\frac{\partial \mathscr{E}_n}{\partial w_{ji}}$. The outputs of the various units will depend on the particular input pattern n. However, in order to keep the notation clean we will omit the n subscript from the network variables. First we note that $\mathscr{E}_n$ depends on the weight w_{ji} only via the summed input a_j to unit j. We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial \mathscr{E}_n}{\partial w_{ji}} = \frac{\partial \mathscr{E}_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{3.44}$$

We introduce a useful notation

$$\delta_j = \frac{\partial \mathscr{E}_n}{\partial a_j} \tag{3.45}$$

Now using (3.42) we can write

$$\frac{\partial \mathscr{E}_n}{\partial w_{ji}} = \delta_j z_i \tag{3.46}$$

Equation (3.46) tells us that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end with the value of z at the input end of the weights($z = 1$ for a bias). We can easily notice that this is the same as the derivation for the simple linear model which we analysed earlier in this section. Therefore in order to evaluate the derivatives we only need to calculate the δ_j for each hidden and output unit in the network and then apply (3.46).

We remember that for the output units we have

$$\delta_k = y_k - t_k \tag{3.47}$$

To evaluate the δ's for hidden units, we simple make use of the chain rule for partial derivatives,

$$\delta_j \equiv \frac{\partial \mathscr{E}_n}{\partial a_j} = \sum_k \frac{\partial \mathscr{E}_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{3.48}$$

where the sum runs over all unit k to which j units are connected. The arrangement of units is illustrated in Fig.3.6

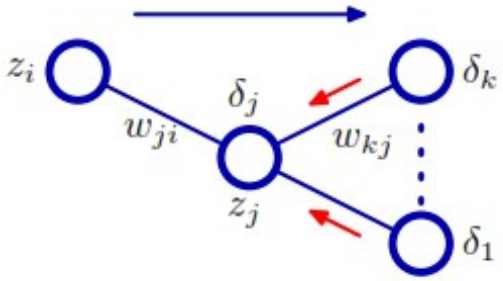


Figure 3.6: Illustration of the calculation of δ_j. ([9, Bishop 2006])

The meaning behind (3.48) is that variations in a_j give rise to variations in the error function only through variations in a_k. We can substitute the definition of δ from (3.45) into (3.48) and make use of (3.42) and (3.43) to obtain the following *backpropagation* formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{3.49}$$

which tells us that the value of δ for a particular hidden unit can be obtained by propagation the δ's backwards from units higher up in the network. Because we already know the values of δ's for the output units, it follows that by recursively applying (3.49) we can evaluate the δ's for all of the hidden units in a feed-forward network, regardless of its topology.

3.5 Bias-Variance Tradeoff

In order to validate a machine learning method and a neural network is no exception, we have to understand what it means for a multilayer perceptron neural network to be well trained. According to [8], neural learning means the synaptic weighs and and thresholds of the network are updated by successfully and efficiently applying the back-propagation algorithm with the purpose of creating a mapping between the inputs $\{x_i\}$ and the outputs $\{y_k\}$ of the network.

Nevertheless before we can understand the validation methods that we used, it is instructive to have a clear distinction between two very important types of errors, i.e. *bias* and *variance*. Gaining a proper understanding of these errors will not only help build accurate models but also to avoid the mistake of overfitting and underfitting.

We can express the above two meanings mathematically. Let's say we are trying to predict values for a variable (response) D using variables $\mathbf{X}$ and parameters $\mathbf{w}$. We will use the generic linear regression model to motivate our point but note that this done as a simple example without loss of generality [9].

$$D = \phi(\mathbf{X}, \mathbf{w}) + \epsilon \tag{3.50}$$

where $\phi(\mathbf{X}, \mathbf{w})$ is a deterministic function of $\mathbf{X}$ and parameters $\mathbf{w}$ and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is the error term. This model is a mathematical description of a stochastic environment and it's purpose is to *explain* or *predict* the value of Y produced by $\mathbf{X}$.

Now we create a model of the environment with the purpose of encapsulating the empirical knowledge represented by the training sample $\mathcal{Q}$. This training will lead to an *estimation* $\hat{\mathbf{w}}$ of the real unknown parameter vector $\mathbf{w}$. Schematically the above are denoted by:

$$\mathcal{Q} \longrightarrow \hat{\mathbf{w}} \tag{3.51}$$

In effect the estimation model provides an *approximation* to the regression model of (3.50). We will denote this input-output function that is realized by the estimation model by

$$y = \Phi(\mathbf{X}, \hat{\mathbf{w}}) \tag{3.52}$$

Where y is a realization of the random variable Y. Given the training sample $\mathcal{Q}$ of (3.51), the estimator $\hat{\mathbf{w}}$ is minimizing the error function, which in term of squared differenced between the desired and the actual responses takes the form:

$$\mathscr{E}(\hat{\mathbf{w}}) = \frac{1}{2}\sum_{i=0}^{N}(d_i - \Phi(\mathbf{X}_i, \hat{\mathbf{w}}))^2 \tag{3.53}$$

Letting $\mathbb{E}_{\mathcal{Q}}$ denote the average operator and using (3.51) we can write (3.53) in the form

$$\mathscr{E}(\hat{\mathbf{w}}) = \frac{1}{2}\mathbb{E}_{\mathcal{Q}}[(d - \Phi(\mathbf{X}, \mathcal{Q}))^2] \tag{3.54}$$

with further manipulation of the equations that we already have we can reduce (3.53) into the form

$$\mathscr{E}(\hat{\mathbf{w}}) = \frac{1}{2}\mathbb{E}_{\mathcal{Q}}[\epsilon^2] + \frac{1}{2}\mathbb{E}_{\mathcal{Q}}[(\phi(\mathbf{X}, \mathbf{w}) - \Phi(\mathbf{X}, \mathcal{Q}))^2] \tag{3.55}$$

The first term is the variance of the expectational error ϵ over the data sample $\mathcal{Q}$. It's called the *intrinsic error* because it is independent of $\hat{\mathbf{w}}$ and thus it is irreducible. It is a measure of the amount of noise in our data and therefore no matter how good we make our model, our data will have certain amount of noise or irreducible error that can not be removed.

Since the first term is irreducible this means that the total error is reduced only by reducing the second term. Therefore $\hat{\mathbf{w}}$ that minimized the error function also minimizes the ensemble average of the squared distance between the regression function $\phi(\mathbf{X}, \mathbf{w})$ and the approximation function $\Phi(\mathbf{X}, \hat{\mathbf{w}})$. This means that the effectiveness of $\Phi(\mathbf{X}, \hat{\mathbf{w}})$ as a predictor of the desired outcome d is defined as:

$$\mathscr{L}_{av}(\hat{\mathbf{w}}) = \mathbb{E}_{\mathcal{Q}}[(\phi(\mathbf{X}, \mathbf{w}) - \Phi(\mathbf{X}, \mathcal{Q}))^2] \tag{3.56}$$

It can be shown that for the case of regression the optimal solution $\phi(\mathbf{X}, \mathbf{w})$ is actually the conditional expectation $\mathbb{E}[d|\mathbf{X}]$. Therefore we can rewrite (3.56) as:

$$\mathscr{L}_{av}(\hat{\mathbf{w}}) = \mathbb{E}_{\mathcal{Q}}[(\mathbb{E}[d|\mathbf{X}] - \Phi(\mathbf{X}, \mathcal{Q}))^2] \tag{3.57}$$

The difference between the operators $\mathbb{E}_{\mathcal{Q}}$ and $\mathbb{E}$ should be very carefully noted. The variables and their functions under $\mathbb{E}_{\mathcal{Q}}$ represent entities in the sample $\mathcal{Q}$. Contrary to that the statistical operator $\mathbb{E}$ acts on the whole ensemble (band) of $\mathbf{X}$ and D which includes $\mathcal{Q}$ as a subset.

Expression (3.57) may be viewed as the average value of the estimation error between the regression function $\phi(\mathbf{X}, \mathbf{w})$ and the approximation function $\Phi(\mathbf{X}, \hat{\mathbf{w}})$ evaluated over the sample $\mathcal{Q}$. Next we can write

$$\mathbb{E}[d|\mathbf{X}] - \Phi(\mathbf{X}, \mathcal{Q}) = \mathbb{E}[d|\mathbf{X}] - \mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})] + \mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})] - \Phi(\mathbf{X}, \mathcal{Q})$$

where we simply added and subtracted the term $\mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})]$. By putting this back in (3.57) we end up with the final form:

$$\mathscr{L}_{av}(\hat{\mathbf{w}}) = (\mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})] - \mathbb{E}[d|\mathbf{X}])^2 + \mathbb{E}_{\mathcal{Q}}[(\Phi(\mathbf{X}, \mathcal{Q}) - \mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})])^2] \tag{3.58}$$

where the above terms are defined as *Bias* (3.59) and *Variance* (3.60).

$$\mathscr{B}(\hat{\mathbf{w}}) = \mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})] - \mathbb{E}[d|\mathbf{X}] \tag{3.59}$$

and

$$\mathscr{V}(\hat{\mathbf{w}}) = \mathbb{E}_{\mathcal{Q}}[(\Phi(\mathbf{X}, \mathcal{Q}) - \mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})])^2] \tag{3.60}$$

and then finally

$$\mathscr{L}_{av}(\hat{\mathbf{w}}) = \mathscr{B}^2(\hat{\mathbf{w}}) + \mathscr{V}(\hat{\mathbf{w}}) \tag{3.61}$$

- *Bias* is the difference between $\mathbb{E}_{\mathcal{Q}}[\Phi(\mathbf{X}, \mathcal{Q})]$, i.e. the average prediction of our model, and $\mathbb{E}[d|\mathbf{X}] = \phi(\mathbf{X}, \mathbf{w})$, i.e. the correct value which we are trying to predict. Thus $\mathscr{B}(\hat{\mathbf{w}})$ represents the inability of the model represented by $\Phi(\mathbf{X}, \hat{\mathbf{w}})$ to accurately approximate $\phi(\mathbf{X}, \mathbf{w})$. We may therefore view the bias $\mathscr{B}(\hat{\mathbf{w}})$ as an *approximation error* [8].

- *Variance* is the variance of the approximating function $\Phi(\mathbf{X}, \hat{\mathbf{w}})$ measured over the entire training sample $\mathcal{Q}$. Thus $\mathscr{V}(\hat{\mathbf{w}})$ expresses the inadequacy of the empirical knowledge contained in the training sample $\mathcal{Q}$ about the regression function $\phi(\mathbf{X}, \mathbf{w})$ and we therefore can view it as an *estimation error* [8].

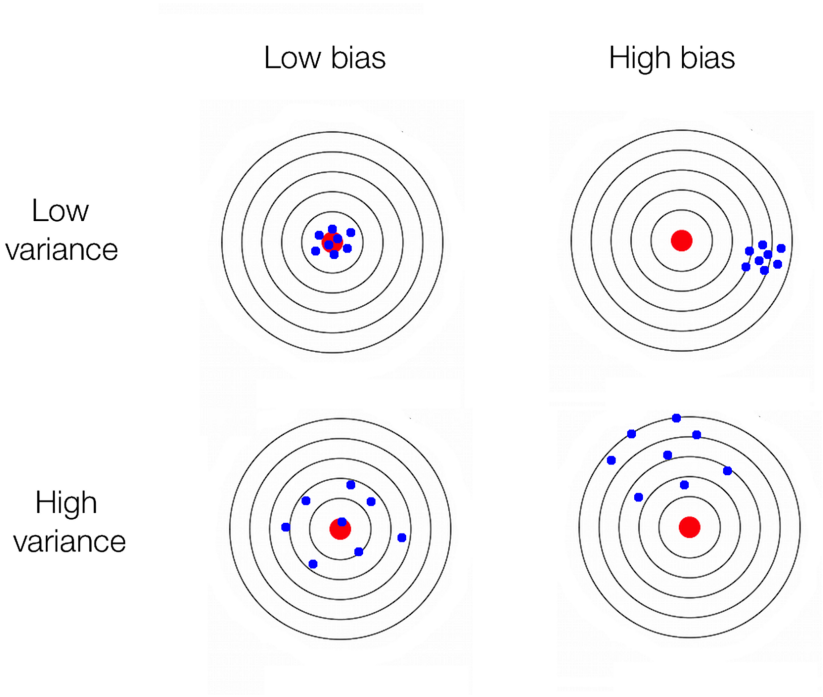


Figure 3.7: Bias and Variance using bulls-eye diagram.

Now that we have a very clear picture of the difference between *bias* and *variance* we may rightfully think that for a model to achieve good performance, both of them must be small. Unfortunately we cannot achieve this ideal situation because we can only have a finite amount of data for training. In principal only with an infinitely large training sample we can eliminate both *bias* and *variance* and this is the famous *bias-variance dilemma*. The consequence of this is increasingly, and therefore prohibitively, slow convergence to out target [11].

In supervised learning, underfitting happens when a model is unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with nonlinear data. Also, these kind of models are very simple, thus unable to capture the complex patterns in data like linear and logistic regression.

On the other side, overfitting happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy data. These models have low bias and high variance and are in general very complex like Decision trees which are prone to overfitting.

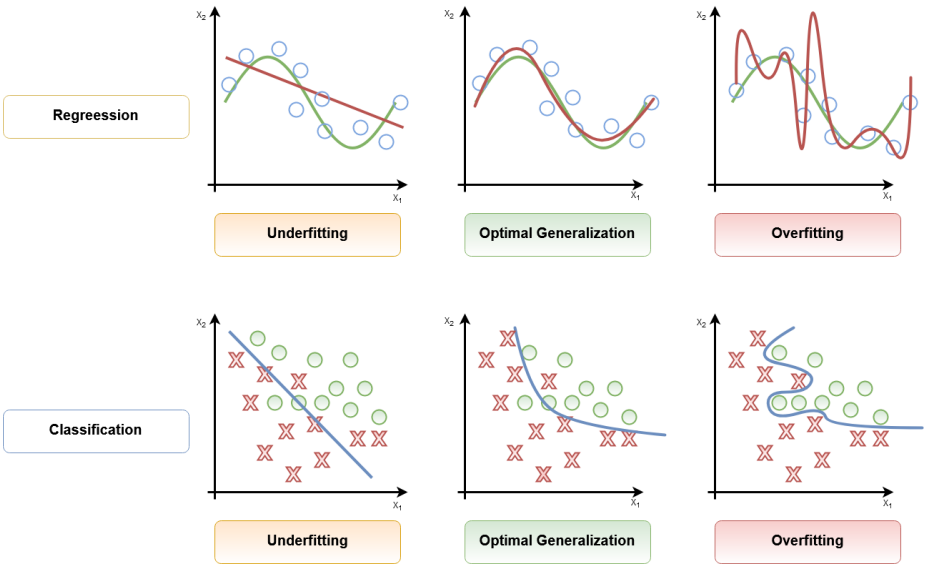


Figure 3.8: Underfitting, Overfitting and Optimal Generalization in Regression and Classification.

3.6 Validation Methods

With that being said let's assume that we fit our model on a training dataset and use the same data to estimate how well it performs in practice. The model can be too simple and thus suffer from underfitting (high-bias) or it can very complex therefore it suffers from overfitting (high-variance) A further problem with cross-validation is that we might end up with a lot of parameters for a single model and thus exploration of all possible combinations of all these parameters will require a number of training runs that grow exponentially with time [9]

For the purposes of this thesis, the very popular k-fold cross validation method will be used but we find it instructive to also presented the classic holdout cross validation method in order to compare and justify our final choice.

3.6.1 Holdout Cross Validation Method

The holdout cross validation method is the most classic approach for estimating the generalisation performance of machine learning models, like neural networks. The process starts with the split of our initial dataset into a separate training and test dataset where the former is used for model training and the latter to estimate performance. But in order to achieve the ultimate purpose of a machine learning application, i.e. to perform predictions on unseen data, we need to tune a set of different parameters (**hyperparameters**) for further improvement in performance. However by reusing the same test dataset repeatedly, during the aforementioned tuning, it will effectively become part of our training data and thus the whole process will most likely lead to overfit. This is a fundamental flaw of the classic holdout method and it thus calls for an improvement.

This improvement is done by separating the data into three parts: a training set, a validation set and finally a testing set. The training set is used to the same reason as before and the validation set is used to tune the parameters as in place of the testing set from the classic method. There is an advantage that we might miss at first, and that is that the test dataset has never been seen from the model, neither for training nor for parameter tuning. Thus by finalizing the process with the predictions on the test dataset the we completely held out, we can obtain a less biased estimate of the models ability to generalize to new data. Both the classic and enhanced holdout methods are show in 3.9.

After this, a disadvantage still exist even with the enhanced method. The partition of the dataset is done only one time, and because the performance estimate is directly coupled with the way we partition the dataset, at the end the estimate will vary for each different sample of the data. The way to fix that flaw is by using the k-fold cross validation method, where essentially we repeat the holdout method k times on k subsets of the training data.

3.6.2 K-fold Cross Validation Method

Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

Cross-validation is a re-sampling procedure used to evaluate machine learning models

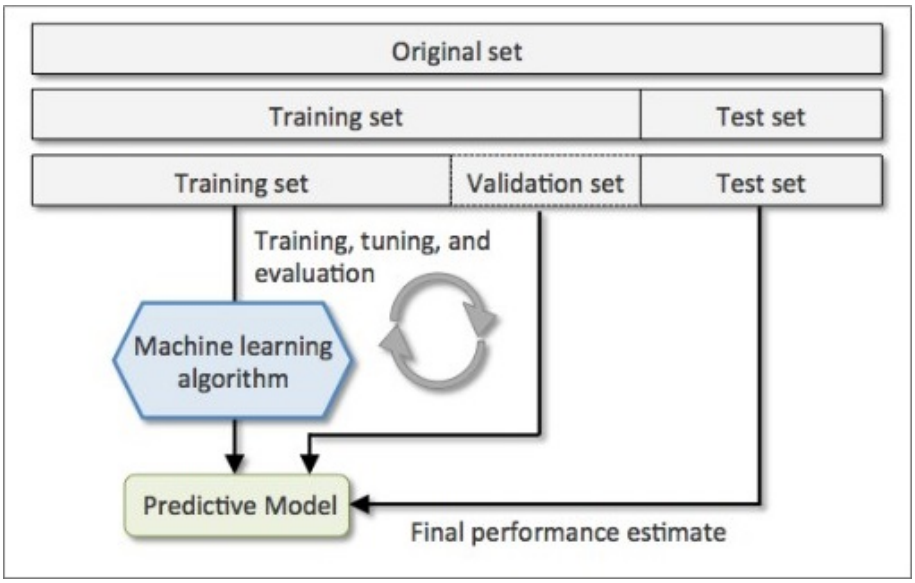


Figure 3.9: The enhanced holdout cross validation technique ([12, Raschka 2015]).

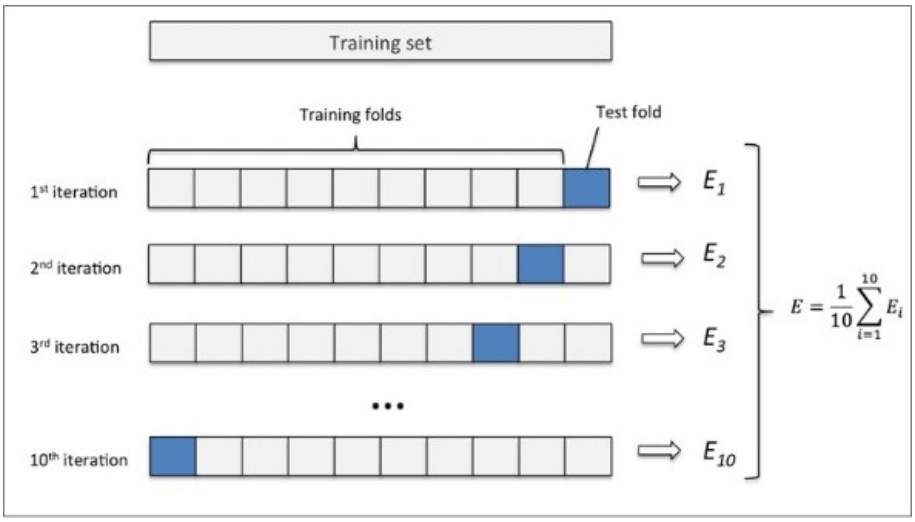


Figure 3.10: The k-folds cross validation technique ([12, Raschka 2015]).

on a limited data sample and the procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, it is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as $k = 10$ becoming 10-fold cross-validation. This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

The general procedure is as follows:

1. Shuffle the dataset randomly.

2. Split the dataset into k groups.
3. For each unique group:
   3.1. Take the group as a hold out or test data set.
   3.2. Take the remaining groups as a training data set.
   3.3. Fit a model on the training set and evaluate it on the test set
   3.4. Retain the evaluation score and discard the model.
4. Summarize the skill of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $k - 1$ times which yields a lower variance-estimate of the model compared to the holdout method.[12]

It is also important that any preparation of the data, for example normalization, prior to fitting the model occur on the CV-assigned training dataset within the loop rather than on the broader data set. This also applies to any tuning of hyperparameters. A failure to perform these operations within the loop may result in data leakage and an optimistic estimate of the model skill.

The value of k, according to [12], is negatively proportional to the size of the training set. For example, if we are dealing with relatively small training sets, we might need to increase the number of folds. If we increase the value of k, more training data will be used in each iteration, which results in a lower bias towards estimating the generalization performance by averaging the individual model estimates. However, large values of k lead to increased runtime for the cross-validation algorithm and yield estimates with higher variance since the training folds will be more similar to each other. On the other hand, if we are working with large datasets, we can choose a smaller value for k, for example, $k = 5$, and still obtain an accurate estimate of the average performance of the model while reducing the computational cost of refitting and evaluating the model on the different folds.

3.7 Neural Networks With Python and Tensorflow

TensorFlow [13] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as Graphics Processing Units (GPU) cards. The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. The TensorFlow interface and an implementation of that interface that was built at Google is presented at [13].

We can use lower-level APIs to build models by defining a series of mathematical operations. Alternatively, we can use higher-level APIs (like *tf.estimator*) to specify predefined architectures, such as linear regressors or neural networks. The following figure shows the current hierarchy of TensorFlow toolkits:

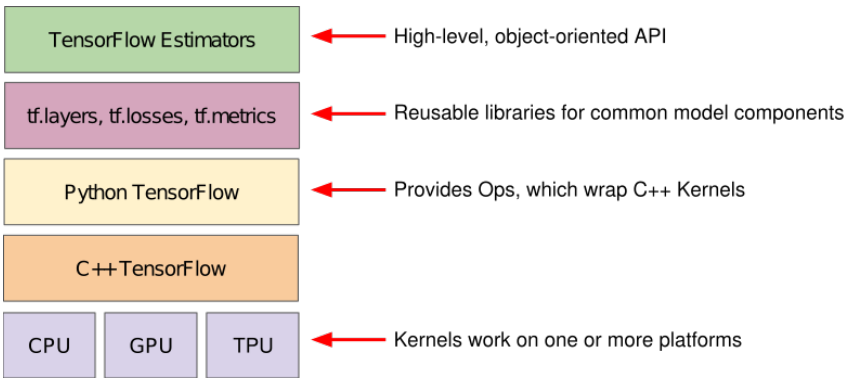


Figure 3.11: TensorFlow toolkit hierarchy. (Source: Google)

TensorFlow consists of the following two components:

- *A graph protocol buffer.*
- *A runtime that executes the (distributed) graph.*

These two components are analogous to Python code and the Python interpreter. Just as the Python interpreter is implemented on multiple hardware platforms to run Python code, TensorFlow can run the graph on multiple hardware platforms, including CPU, GPU, and TPU.

The following table summarizes the purpose of each level of abstraction:

With these presented choices a question of the context "Which API(s) should I use?" comes into play. According to Google the reality is that we should use the highest level of abstraction that solves the problem at hand. The higher levels of abstraction are easier to use, but are also (by design) less flexible. Thus we start with the highest-level API first and get everything working. If we need additional flexibility for some special modelling

Table 3.1: The Tensorflow Toolkit Depth

Toolkit(s)	**Description**
Estimator (tf.estimator)	High-level, OOP API.
tf.layers/tf.losses/tf.metrics	Libraries for common model components.
TensorFlow	Lower-level APIs

concerns, move one level lower. Note that each level is built using the APIs in lower levels, so dropping down the hierarchy should be reasonably straightforward.

Chapter 4

Data for Cruising Pattern Prediction

Cruising pattern exhibited in real cruising is the product of the instantaneous decisions of the vessel operator to cope with the physical marine environment. It plays a core role in power distribution and therefore needs to be predicted during real time operation. Cruising patterns can be defined in terms of the operation profile of the vessel between time interval $[t - \Delta w, t]$, where t is the current time and $\Delta w > 0$ is the window size that characterizes the length of the operation profile that should be used to explore cruising patterns.

In every machine learning project is it essential to collect the right data. As data we define any unprocessed fact that can be interpreted and analysed. They are the most important part of all Data Analytics, Machine Learning, Artificial Intelligence. Without data, we can't train any model and all modern research and automation will go in vain. Therefore in this section we will briefly discuss about the data and their preprocessing in order to predict the cruising and trend patterns.

4.1 Cruising and Propeller Pitch Angle Patterns

Cruising patterns can be observed in the speed profile of the vessel in a particular maritime environment. In order to develop an intelligent system that can effectively predict the cruising type and cruising trend of a vessel, the selection of a good set of features is one of the most important steps. It has been shown in pattern recognition that too many features may degrade system performance. This can be explained by the increased hardware and computation costs due to the acquisition and processing of the new features, especially in an embedded implementations. In the automotive industry an extensive research has been conducted and the outcome is that a small subset of features can give satisfying prediction results according to [14].

In general a cruising pattern can be described as a composition of different cruising types such as port manoeuvring, open sea etc. This integral part was developed in [15], where two different methods, Euclidean Distance (ED) and Dynamic Time Warping (DTW) were used to create clusters of similar data from real ship operation profiles. In both methods the data were clustered into seven and nine clusters, each containing 1730 seconds. For the purposes of this study we considered only the seven clusters edition of the DTW.

Table 4.1: The statistics of the 7 Controllable Pitch Propeller β angle Patterns

β Pattern	β_{max}	β_{min}	β_{avg}	β_{std}	25%	50%	75%	Duration (sec)
Pattern 1	0.597	0.019	0.505	0.094	0.518	0.537	0.554	1730
Pattern 2	0.464	0	0.228	0.186	0.0002	0.318	0.422	1730
Pattern 3	0.732	0	0.403	0.282	0.005	0.528	0.651	1730
Pattern 4	0.467	0	0.179	0.174	0.0014	0.168	0.357	1730
Pattern 5	0.723	0	0.679	0.079	0.678	0.708	0.711	1730
Pattern 6	0.708	0	0.311	0.309	0	0.211	0.648	1730
Pattern 7	0.721	0	0.231	0.290	0	0.003	0.637	1730

One of the main goals of this study is to to address multiple systems with similar dynamics therefore the pitch angle patterns are normalized, which means that since β ranges in $[0°, 90°]$ a $\beta = 0.5$ is the same as $\beta = 45°$. The statistics of these normalized patterns are presented in Table.4.1, where β_{avg} is the average pitch angle, β_{max} is the maximum, β_{min} is the minimum, β_{std} is the standard deviation and 25^{th}% 50^{th}% and 75^{th}% percentages. Also we present the pitch angle time-plots in 4.1.

After studying these statistics it becomes apparent that Patterns 3, 6 and 7 share a lot of similarities. One of these is that they contain a lot of idle time which can be seen by the 25^{th}% that is 0 in all three of them and for Pattern 7 even the 50^{th} percentile is nearly 0. Idle time is defined as the time with $\beta = 0$ which is not characteristic of a tug vessel's operation profile. These similarities will make it difficult, even for the neural network, to effectively distinguish between them. Therefore we have to perform some corrections in our patterns so they can fit the purpose of our study in a better way. We focus our efforts on the two following directions:

- Eliminate unnecessary similarities.
- Minimize idle time as much as possible.

Correlation estimations are commonly used in various data mining applications and based on the above a correction mechanism based on correlation was devised. In our case,

patterns are nonlinear time series and thus they might share nonlinear correlation. Due to the absence though of a nonlinear correlation method we explore the similarity between two patterns with a simplistic method in which the following two statements must hold simultaneously:

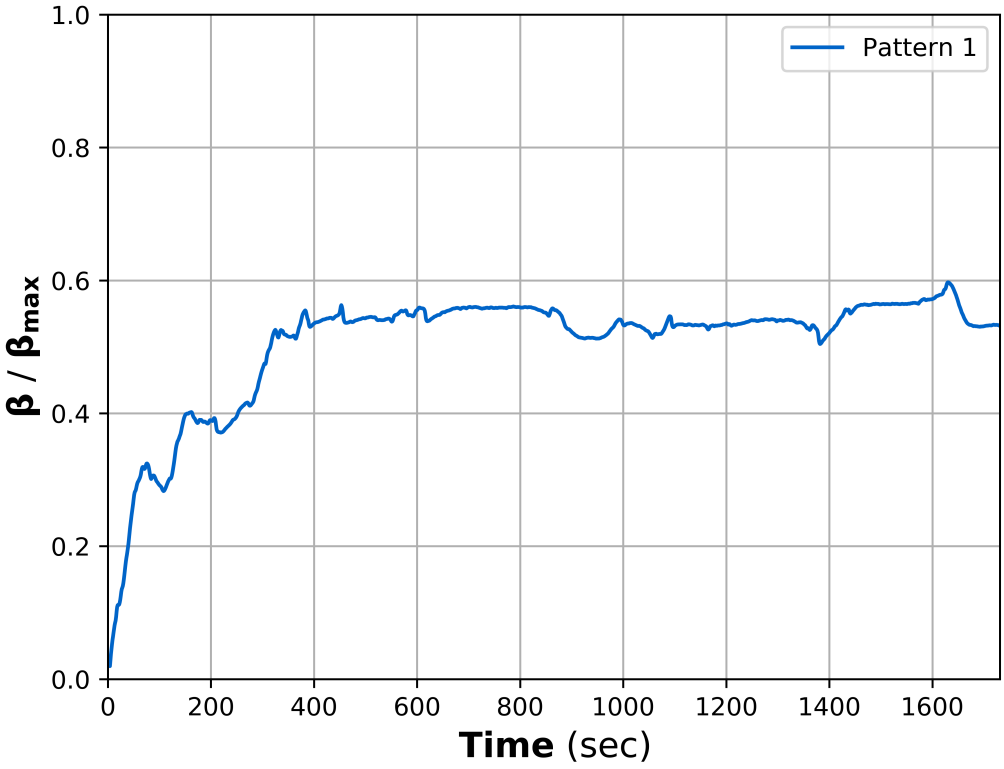
- The patterns have akin ranges of values regardless of time.
- The patterns report a positive Pearson Correlation Coefficient larger that 0.5.

Pearson's correlation coefficient is defined as the covariance of two variables divided by the product of their standard deviations. In Fig. 4.2 we show the Pearson Correlation Matrix for the patterns. It is important to note that Pearson's method describes the strength of a linear relationship between two variables and thus it provides only half of the picture that we need to determine their similarity. Therefore we must interpret it's results with caution so that we do not falsely identify two patterns as similar.

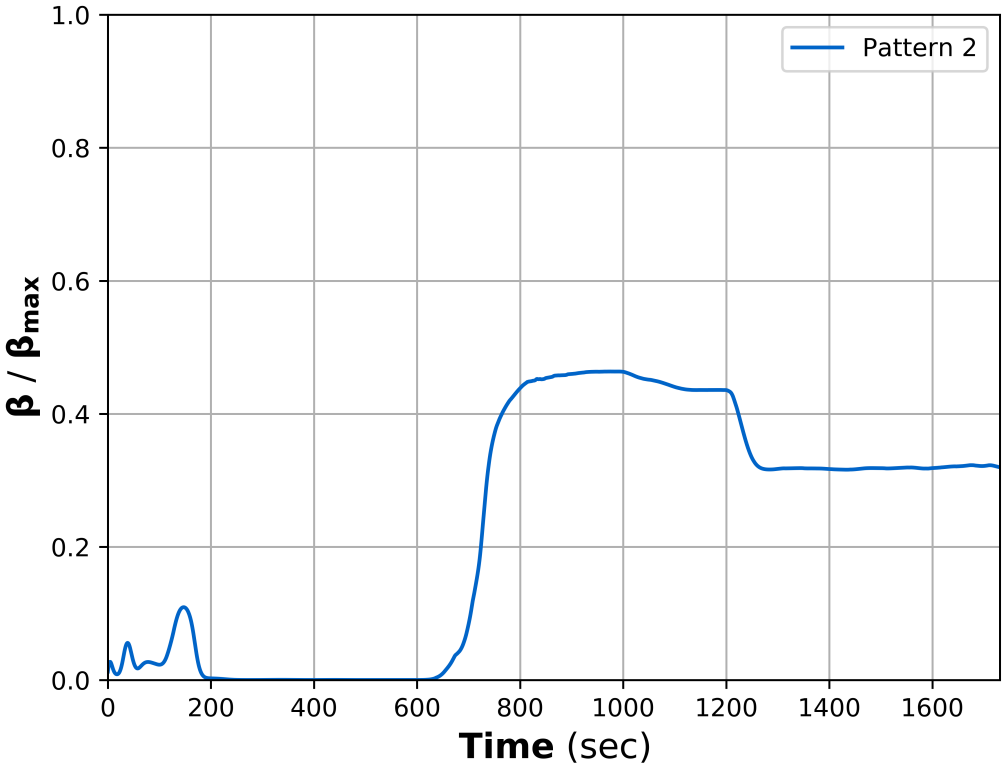
We refer to the correlation coefficient between Pattern A and Pattern B as $\rho_{A,B}$. The coefficient $\rho_{3,6} = 0.65$ agrees with our initial expectation of strong correlation between Pattern 3 and 6. Taking into account the statistic values from Tab. 4.1 as well as the ranges of values in these two patters we conclude that they are indeed similar. Additionally from the time plots we can see that the part from the beginning to 450 seconds of Pattern 7 is nearly the same as the part from 800 seconds to 1200 seconds of Pattern 6 and this is further validated from the mild correlation coefficient $\rho_{6,7} = 0.28$. Since our goal is to minimize as much idle time as possible and to eliminate unnecessary similarities we will discard Pattern 6 and then combine the part from the start until 1200 seconds of Pattern 3 with the part from 1200 seconds to the end of Pattern 7. This will be our new Pattern 5 and it's shown in Fig.4.3.

For the other patterns the coefficients $\rho_{5,1} = 0.84$, $\rho_{2,4} = 0.7$ are above our threshold, but they do not share analogous ranges of values hence they are not similar.

Finally from hereafter we will refer to these patterns as Controllable Pitch Propeller(CPP) β angle(pitch) Patterns and it is possible to investigate the operation of a Controllable Pitch Propeller as a sequence of these patterns. For the convenience of description, we label these 5 patterns as $PC_1, \dots PC_5$.

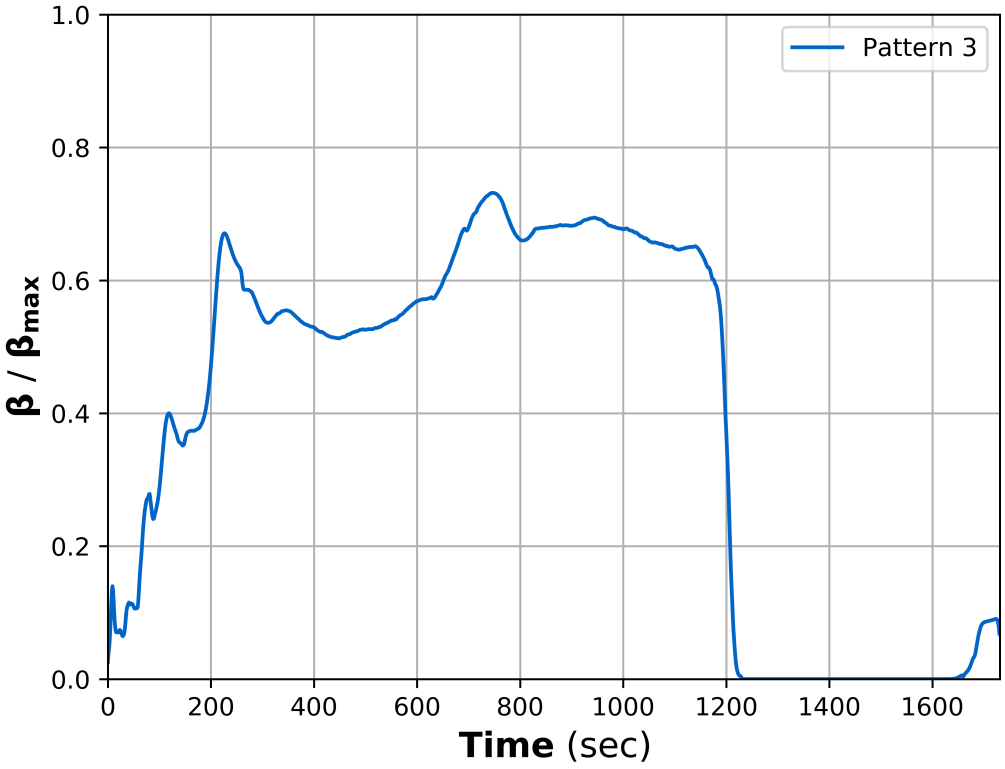


(a) Pitch Angle β Pattern 1

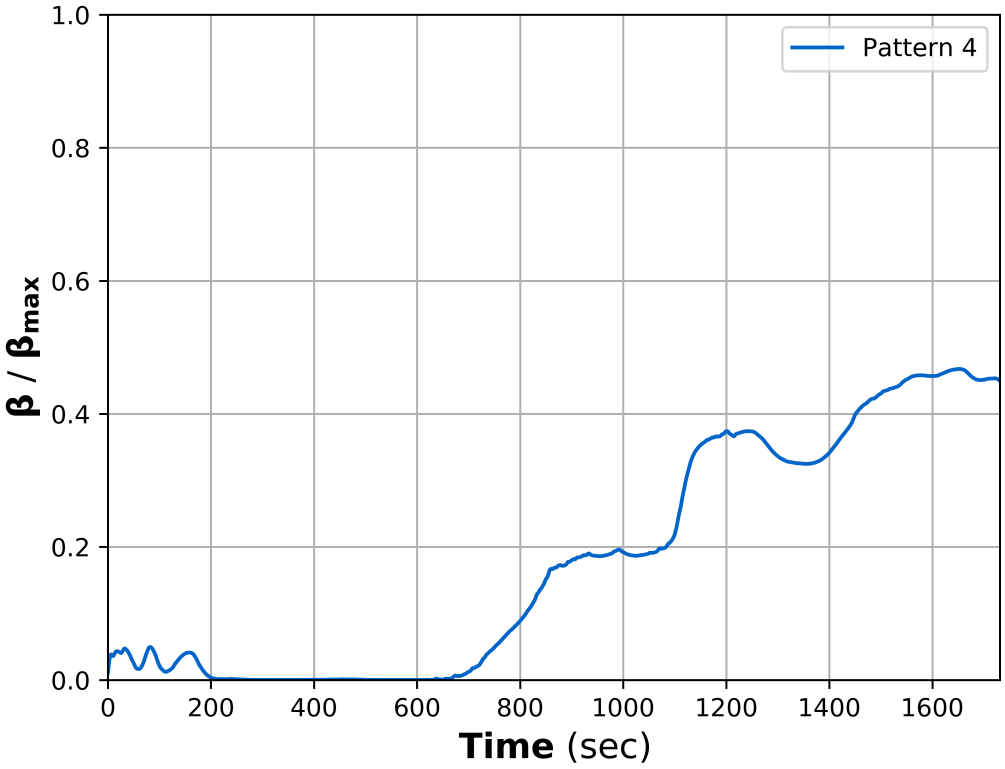


(b) Pitch Angle β Pattern 2

Figure 4.1: Considered Operation Pitch Angle β Patterns

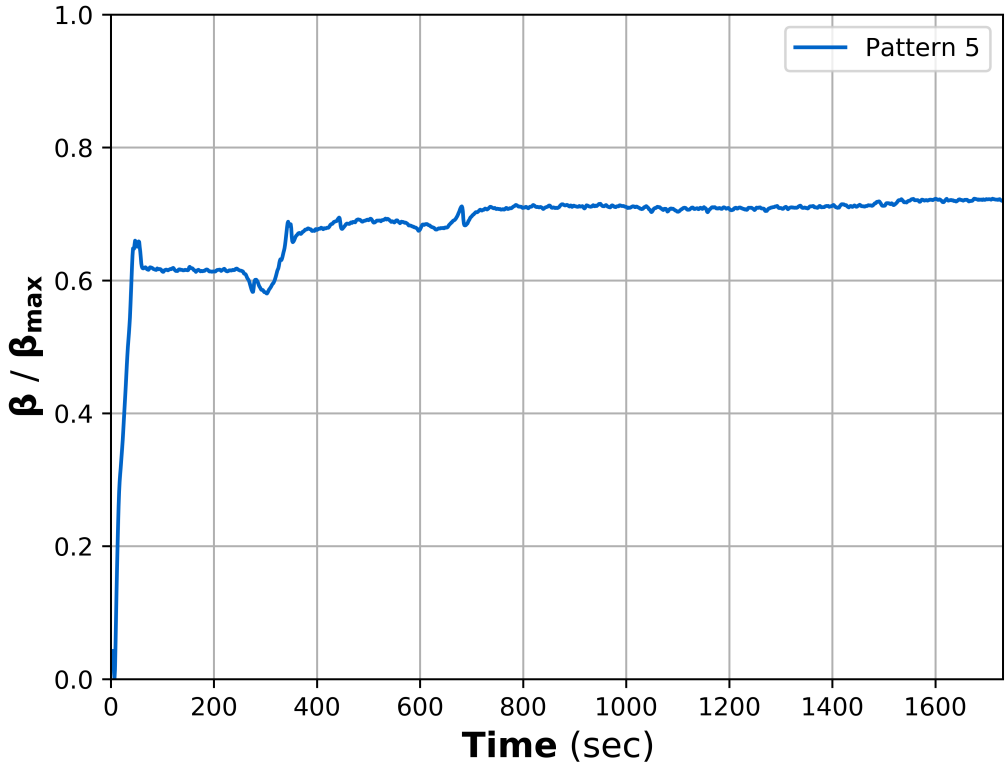


(c) Pitch Angle β Pattern 3

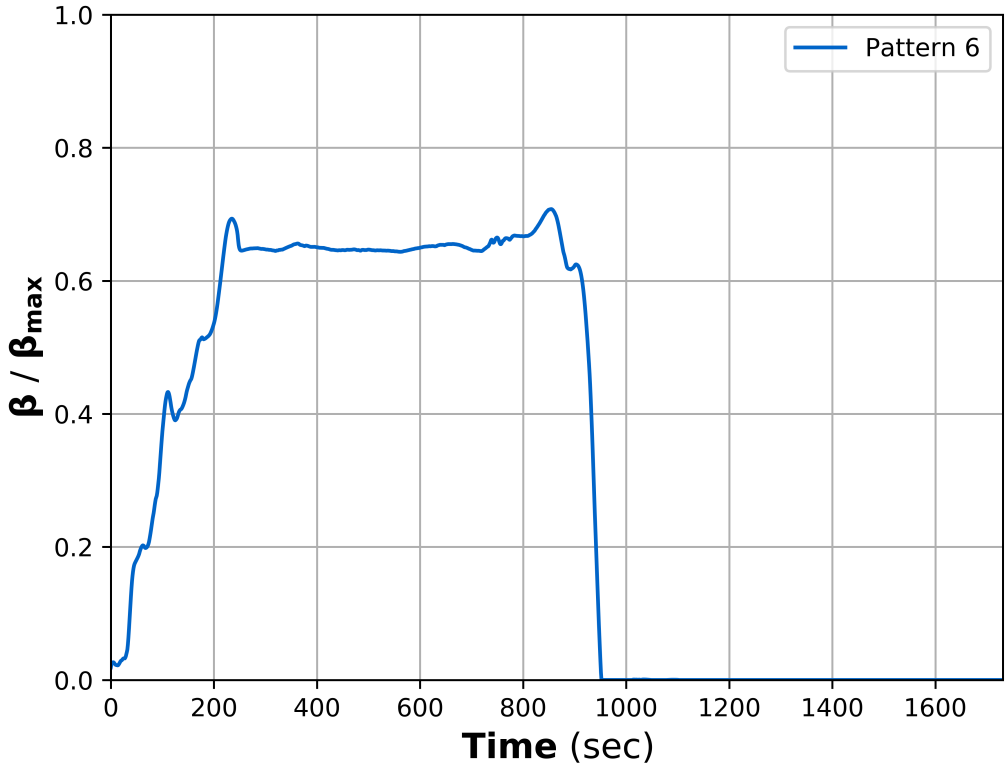


(d) Pitch Angle β Pattern 4

Figure 4.1: Considered Operation Pitch Angle β Patterns (cont.)

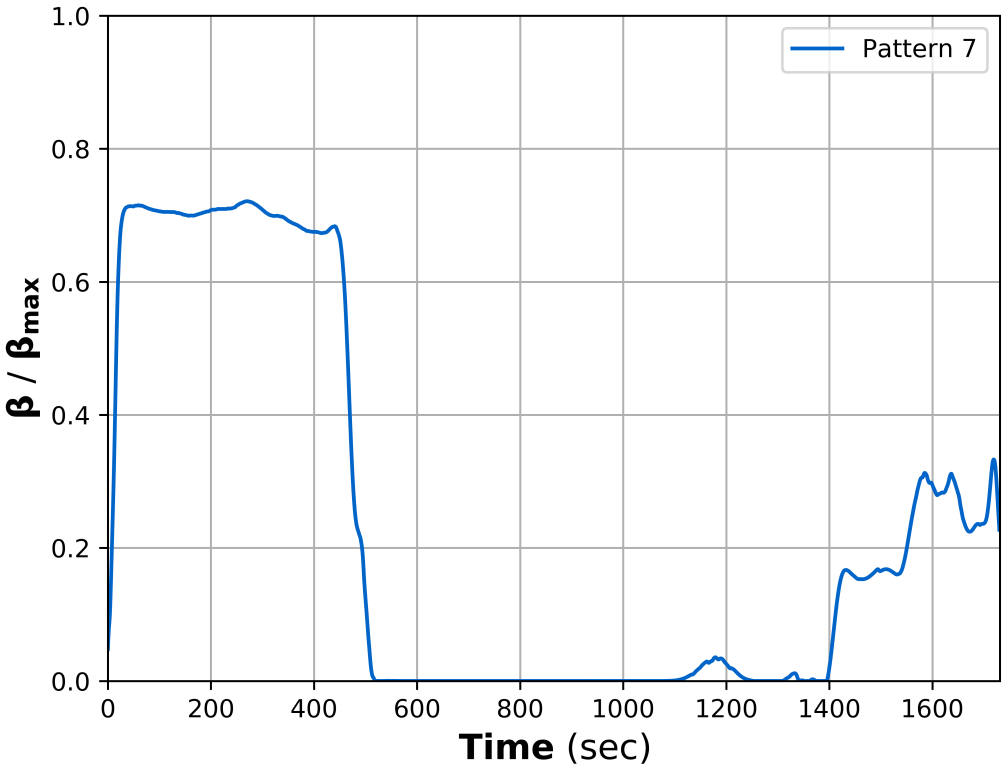


(e) Pitch Angle β Pattern 5



(f) Pitch Angle β Pattern 6

Figure 4.1: Considered Operation Pitch Angle β Patterns (cont.)



(g) Pitch Angle β Pattern 7

Figure 4.1: Considered Operation Pitch Angle β Patterns (cont.)

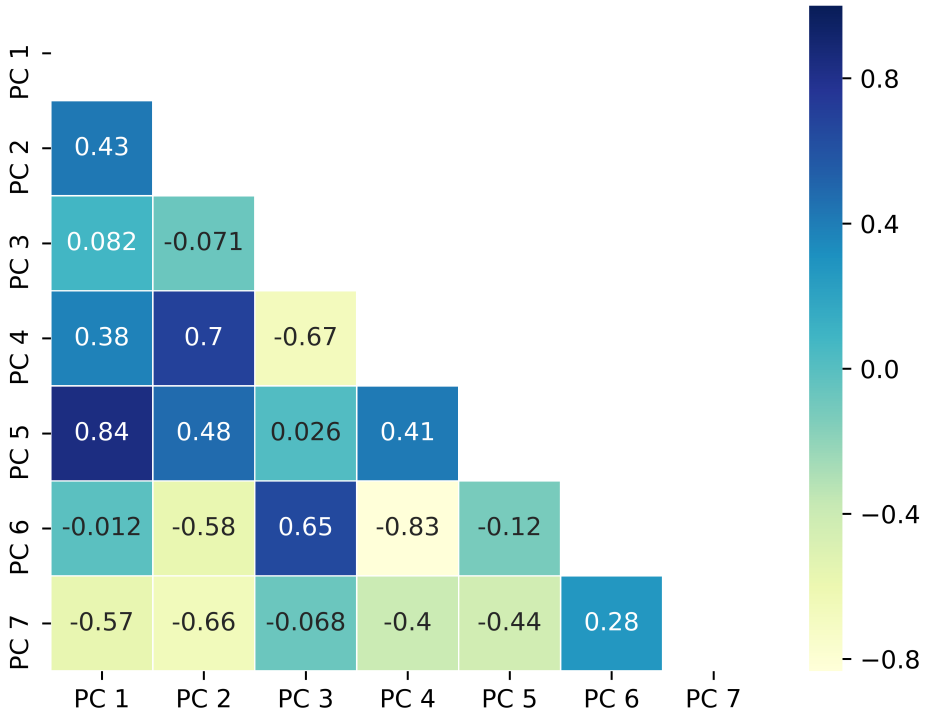


Figure 4.2: Pearson Correlation Matrix of Pitch Angle β Patterns

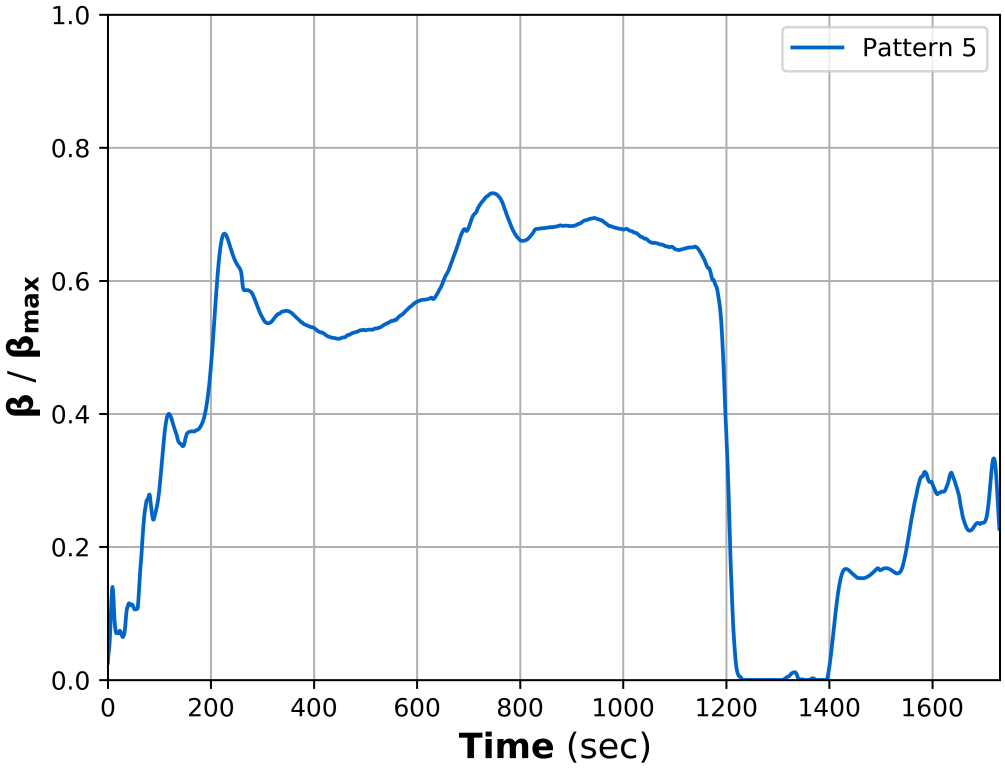


Figure 4.3: The new Pitch Angle β Pattern 5

Chapter 5

Machine Learning Framework

In the present chapter the holistic framework of the investigated control scheme will be presented. We will start with the design procedure, then continue with the tuning and finish with the training reports of the developed framework where further explanation will take place. As we stated earlier in Chapter 3, neural networks that are tackling classification and regression problems require a supervising mechanism for their training. With that in mind, for the design procedure we took the following steps: 1) Supervised Learning Setup for the different Networks, 2) Creation and Analysis of the Basic Framework Constituents, 3) Grouping of Constituents, 4) Simulation of the Framework to acquire Results and finally 5) Assessment of the Results to see if we achieved our initial objectives. From the above 1, 2 and 3 are presented here whereas 4 and 5 are discussed inn the next chapter.

The framework and the graphical user interface (GUI) A that supports it, are developed using the programming language *Python* and the networks are specifically engineered with *Tensorflow*, the Machine Learning Library from Google. A brief description of Tensorflow has been given in Section 3.7 so here we will focus on its implementation for the Neural Network Control scheme.

The purpose of the developed framework is to efficiently perform the power split between the two power units of the propulsion plant using a Nonlinear Model Predictive Controller as the supervising teacher. The ultimate objective of this study is to learn from the NMPC scheme and investigate the possibility of it's replacement with a machine learning approach based on Neural Networks. As we stated earlier we will achieve an indirect regulation of the NOx emissions by smoothing the transient operations of the Diesel Engine via the operation of the Electric Motor by controlling the torque of both power units directly.

In general the rule of thump for Neural Networks, is for the input values to be real life values and not simulated values/estimations from a model. Unfortunately in out case this is not possible because all the necessary parameters are not on deck at every step of the operation. To be more precise, torque load in a marine power plant is not measured directly in most cases because the measured signals are not systematic enough due to noise or non trivial time delay. To solve this problem we will adopt the same Moving Horizon Estimation (MHE) scheme as in [6], an efficient method of estimating the load torque which will be use as an integral input feature for our control phase networks.

5.1 The Framework Strategy

In this section, we explain the attributes and constituents of the developed framework LME-NNPower.

We devise the problem of optimal power split on a marine hybrid diesel-electric propulsion plant as follows. Consider that for any given pitch angle pattern $PC(t)$ ($t \in [0, t_e]$, where t_e denotes the end of a pattern) at any given t, the vessel is operating according to one of the 5 standard pitch angle patterns $PC_1, .., PC_5$ defined in 4.1. We will use these patterns as the basis for calculating the optimal control commands of the NMPC given each one of these patterns. In that way we create 5 different time sequences of torque commands for the Engine and 5 for the Electric Motor that we use as training data for our neural networks. Lastly we replace NMPC with these trained networks for direct engine control. In other words the Neural Networks conduct the power splitting directly, by controlling simultaneously both the torque of the Electric Motor (TQ-MOT) and the torque of the Diesel Engine (TQ-ENG) opposed to *indirect engine control* where only the Electric Motor is controlled.

In Fig. 5.1 a diagram for the proposed machine learning framework is presented featuring all of it's core functionalities. It is instantly clear that it contains two distinct stages.

- The prediction stage.
- The control stage.

Each stage features neural networks that are employed to do specific calculations that we are going to analyse.

5.2 The Prediction Stage

The prediction of the operational status of the propulsion plant is broken down into two core predictions.

- The prediction of future Pitch Angle Pattern (PC).
- The prediction of near future Cruising Trend (CT).

The prediction regarding the Pitch Angle Pattern that the power plant operates in, can be though of as a mid-term prediction and the one about the Cruising Trend a short-term one.

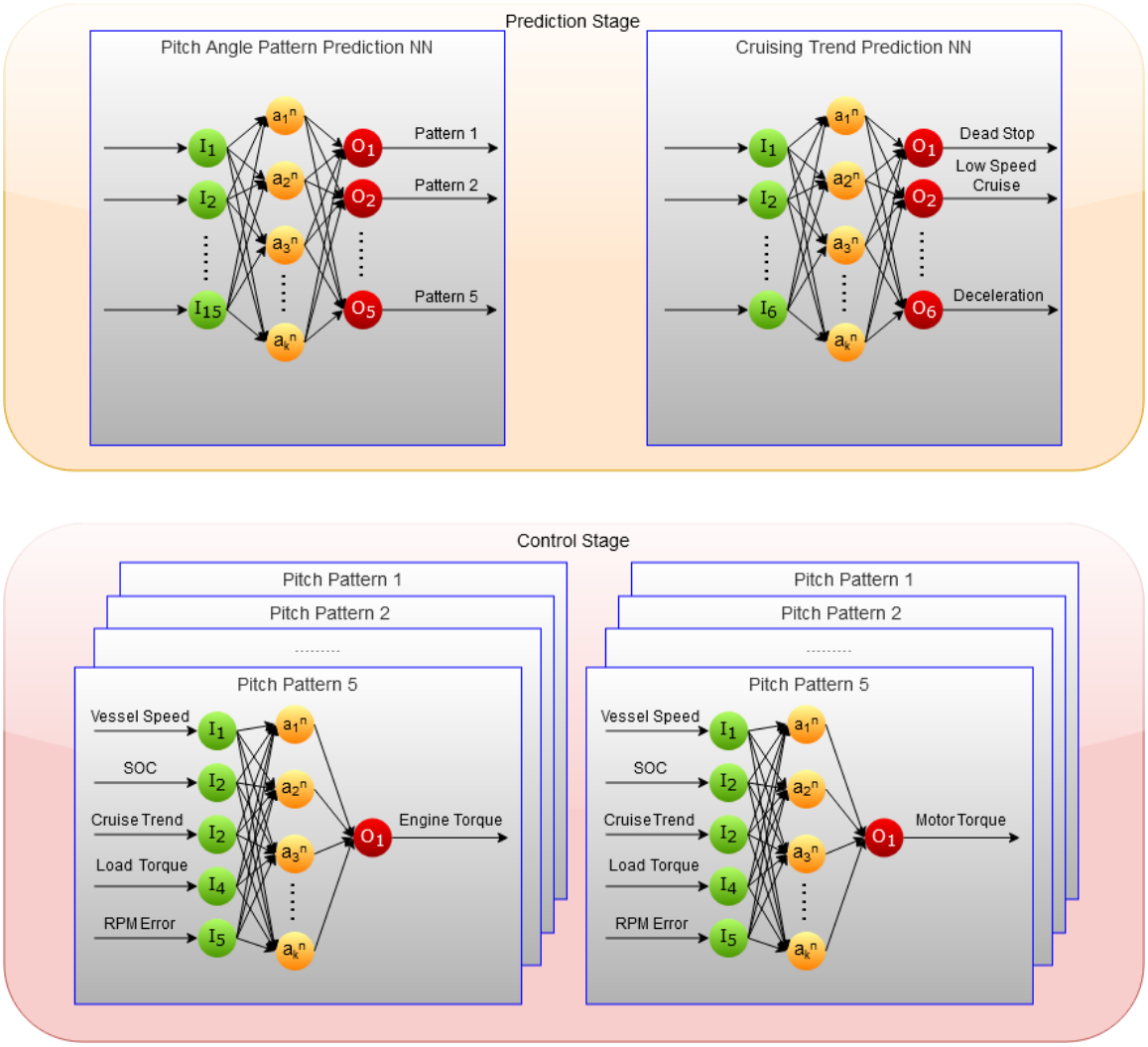


Figure 5.1: LME-NNPower: The Proposed Machine Learning Framework.

5.2.1 Pitch Angle Pattern Prediction

The Pitch angle β Patterns have been presented with detail in 4.1. These patterns are a distillate of a vast amount of operational data, therefore we can use them as adequate representatives of real-world operational profiles for a specific range of operations. Let $PC[t]$ be the pitch patterns sequence that the vessel is operating in order to complete a

trip, with $t = 0, 1, \ldots, t_c, \ldots, t_e$ where t_c is the current time and t_e is the ending time of the trip. For example a possible sequence can be $\{PC_3, PC_1, PC_5, PC_3, PC_2\}_{t=0}^{t_e}$. Thus at any given moment t_c, we assume that $PC(t_c) \in \{PC_i \mid i = 1, \ldots, 5\}$. We will predict the pitch angle pattern in the future based on the mid-term history of the operation profile. In other words, the predictions are made on a step-by-step overlapping time frame.

5.2.1.1 Features Extraction

To predict the pitch pattern at time t_c, we will extract features from the operation profile in the segment $[t_c - \Delta W_{PC}, t_c]$. The non negative value ΔW_{PC} is defined as the window size of the segment that we use to make the prediction. As we stated earlier we will make these predictions on a moving time frame fashion, i.e. at time steps $k\Delta t_{PC}$, $k = 1, 2, \ldots$ and these predictions will be used for calculating the optimal control strategy in the future period $[t_c, t_c + \Delta t_{PC}]$. Fig 5.2 illustrates the functionality that we described above where the x-axis represents time and the y-axis is the normalised β pitch angle. Note that the values for the parameters are only set to $\Delta W_{PC} = 150$ and $\Delta t_{PC} = 100$ for illustration purposes and in reality for this algorithm to have a realist effect they must be smaller.

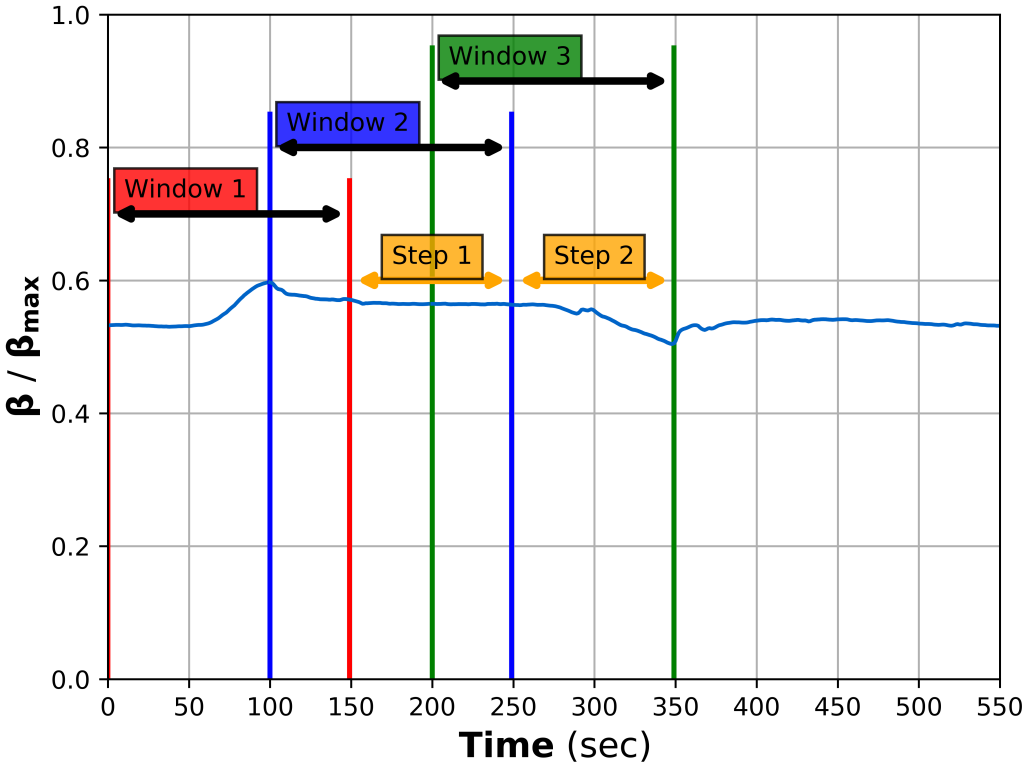


Figure 5.2: Segmentation Algorithm for the step-by-step overlapping time frame.

For the Pitch Patterns classification problem, the data features that we are going to use are given in Table. 5.1. After we have chosen the parameters that we want to use, i.e. by choosing a pair for window size and time step, we obtain these 15 features from the pitch angle operation profile.

We analysed multiple pairs because they are very important for the accuracy and realism

Table 5.1: The selected features for the Prediction of Pitch Propeller β angle Patterns

Feature Name	Description
β_{max}	Maximum Pitch Angle in $[t - \Delta W_{PC}, t)$
β_{ave}	Average Pitch Angle in $[t - \Delta W_{PC}, t)$
ω^{+}_{max}	Maximum Pitch Acceleration in $[t - \Delta W_{PC}, t)$
ω^{+}_{ave}	Average Pitch Acceleration in $[t - \Delta W_{PC}, t)$
ω^{+}_{std}	Standard Deviation of Pitch Acceleration in $[t - \Delta W_{PC}, t)$
ω^{-}_{max}	Maximum Pitch Deceleration $[t - \Delta W_{PC}, t)$
ω^{-}_{ave}	Average Pitch Deceleration in $[t - \Delta W_{PC}, t)$
Π_{03}	Percentage where $0 \leq \beta(t) < 0.3$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{35}	Percentage where $0.3 \leq \beta(t) < 0.5$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{57}	Percentage where $0.5 \leq \beta(t) < 0.7$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{A1}	Percentage where $0 \leq \omega^{+}(t) < 0.003$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{A2}	Percentage where $0.003 \leq \omega^{+}(t)$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{D1}	Percentage where $-0.005 \leq \omega^{-}(t) < 0$ for all $t \in [t - \Delta W_{PC}, t)$
Π_{D2}	Percentage where $\omega^{-}(t) < -0.005$ for all $t \in [t - \Delta W_{PC}, t)$

of the prediction and for real-time implementation. Since the prediction depends on features that we extract from the operation profile in the short term $[t_c - \Delta W_{PC}, t_c]$, the value of ΔW_{PC} is directly related to the amount of information we use for our calculations. That means that it can not be too big as that would lead to a lot of obsolete information in the segment and contrary it can not be too small as that would mean that a lot of information was not taken into account. Furthermore Δt_{PC} is linked with the amount of times we make predictions. Therefore we should carefully chose it's value because if it very small it might cause computational overhead as we make predict very frequently. On the other side it cannot be too big as that would completely nullify the objective of an online controller. Next we are going to describe the training process.

5.2.1.2 Neural Prediction Training

We developed NN-PAC, a multilayer multiclass fully connected feedforward neural network, with the task to predict the pitch angle pattern according to the aforementioned process.

In our search to find the best architecture for our network we analysed multiple layer configurations base on the pair of $\Delta W_{PC} = 60$ and $\Delta t_{PC} = 1$. It should be noted that although we present the results with this pair, an equivalent outcome occurs with the pair $\Delta W_{PC} = 30$ and $\Delta t_{PC} = 1$.

In the beginning we tested neural network with 2 layers by the definition in 3, therefore with 1 hidden layer that had $5, 10, 20, \ldots, 100$ neurons.

All potential networks are trained with the same procedure, a 5-fold cross validation method in order to achieve better generalization. More specifically, we automatically generate the dataset, namely Θ from the pitch pattern operation profile for all 5 patterns. Now we are ready to follow the 4 steps presented in 3.6.2. First we randomly shuffle the dataset Θ and then we split it into 5 subsets Θ_i where $i = 1, 2, \ldots, 5$. Afterwards we create 5 identical neural networks and the network NN_i where $i = 1, 2, \ldots, 5$ is trained for 500 epochs on subsets Θ_j where $j \neq i$ and it's tested on Θ_i. The final accuracy is measured as the average of all accuracies and the best network is kept for later inference. In Fig.5.3 we show the training and testing accuracy chart with respect to the different number of

neurons in the hidden layer.

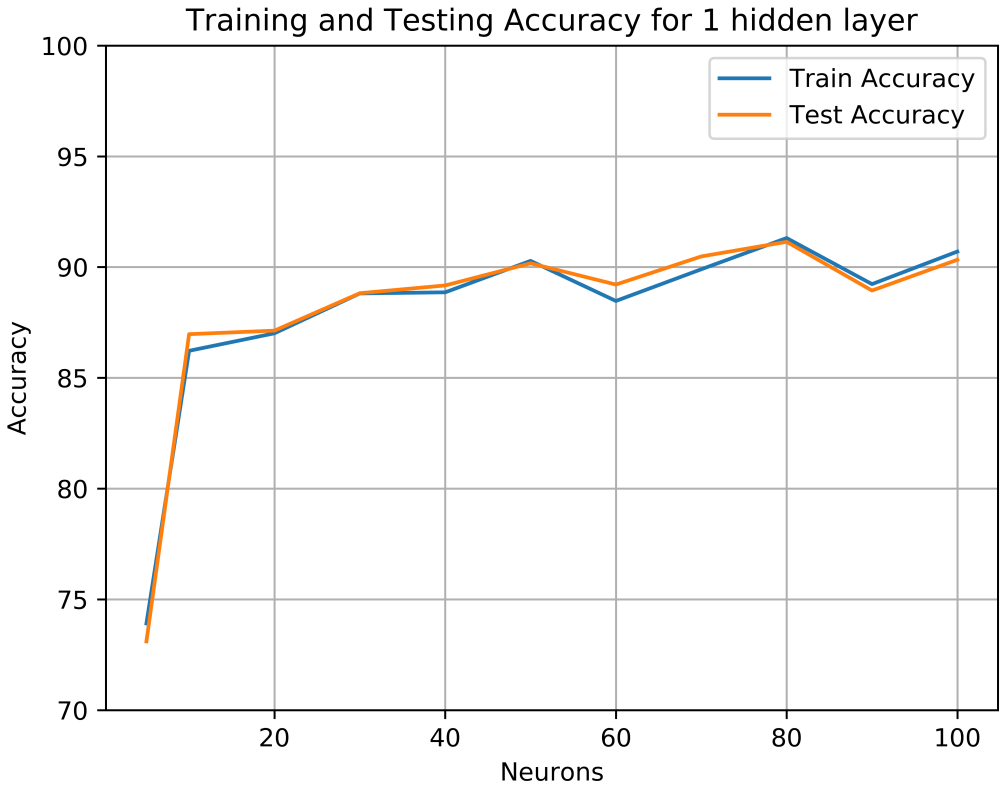


Figure 5.3: Number of hidden layer neurons vs accuracy for NN-PAC with 1 hidden layers.

From this chart it can be seen that the highest value is 93.14 with 80 neurons and after that point the network's performance doesn't increase any further. In order to improve the prediction accuracy we investigate further by adding one more hidden layer. Once again we follow the same process as before and we experiment with different hidden layer neuron sizes, this time with the sets of values $[10-10], [20-10], [30-20], [40-30], [50-40], [60-50]$. In Fig. 5.4 we show the training and testing accuracy chart with respect to the different number of neurons in the hidden layers. We can see that it reaches it's peak performance at 96.68 with the set $[40-30]$ and after that point not a lot of improvement occurs.

The final architecture of the network is shown in Fig.5.5. The input layer has 15 nodes for the features that we described in 5.1, two hidden layers with 40 and 30 nodes and finally the output layer with 5 nodes representing each pattern $\{PC_1, .., PC_5\}$. In order for Tensorflow to work, it requires the outputs to be encoded using the one-hot encode method. One-hot encoding is often used for indicating the state of a state machine. When using binary or Gray code, a decoder is needed to determine the state. A one-hot state machine, however, does not need a decoder as the state machine is in the nth state if and only if the nth bit is high. This means that each class is assigned a binary string with 5 digits and all digits are 0 except for the digit that represents the class. An example would be that class 1 is encoded as 00001, class 2 is 00010 and so on.

The detailed final training and test results, i.e. prediction accuracy of the NN-PAC is presented in Fig. 5.6a for training and in Fig. 5.6b for testing, where the values of the parameters are $\Delta W_{PC} \in \{30, 60, 90, 180\}$ and $\Delta t_{PC} \in \{1, 3, 5, 10, 20\}$.

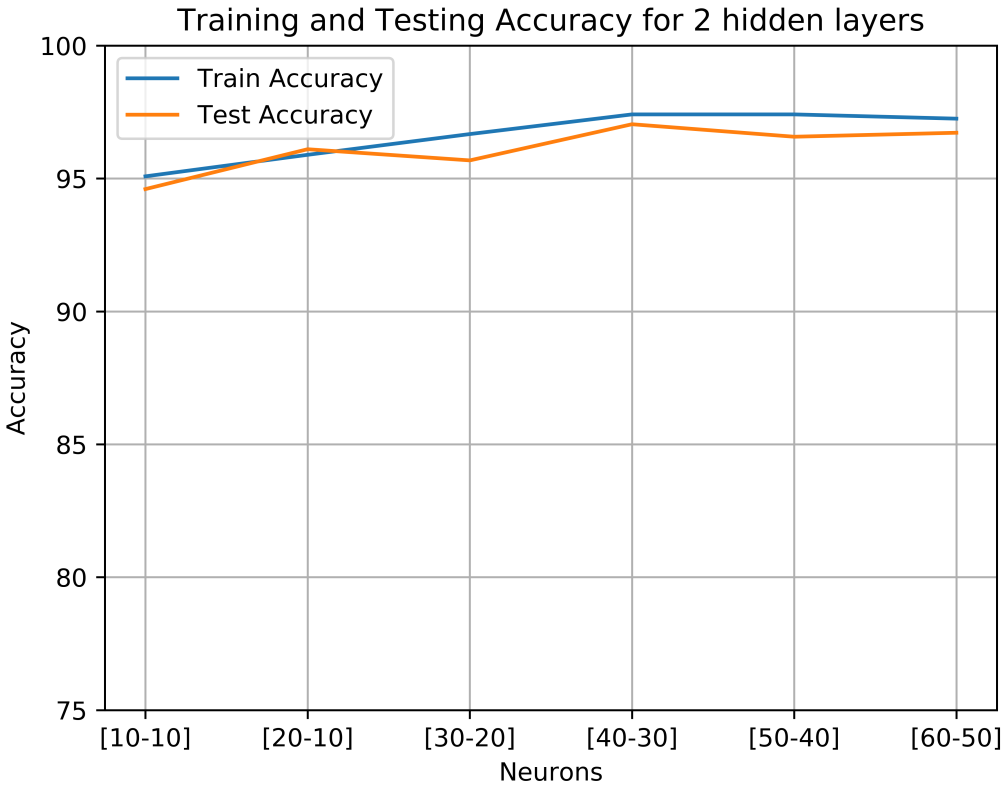


Figure 5.4: Number of hidden layer neurons vs accuracy for a NN-PAC with 2 hidden layers

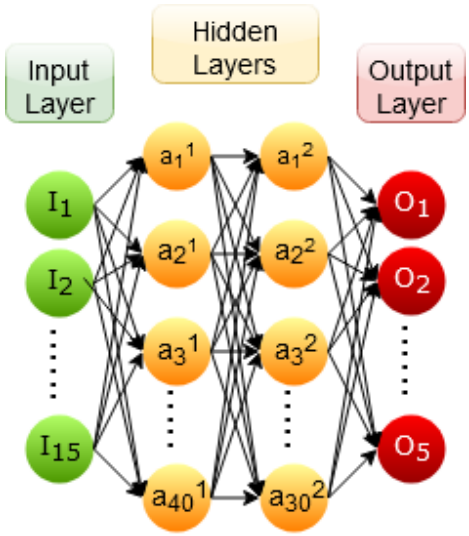
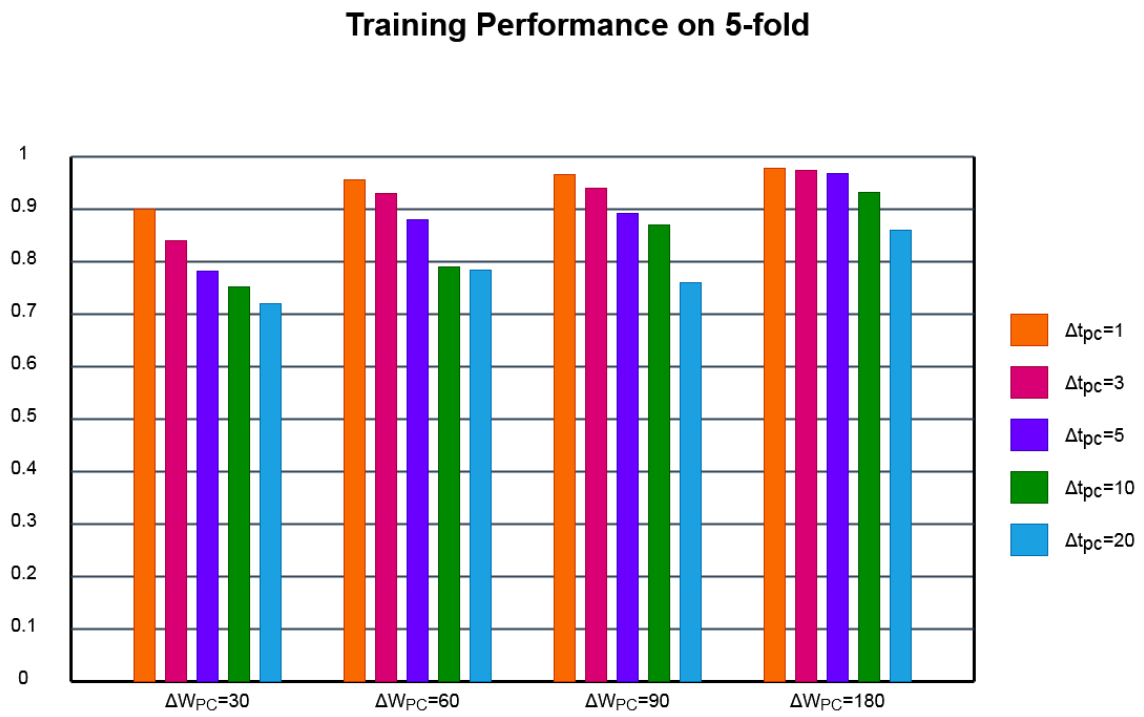
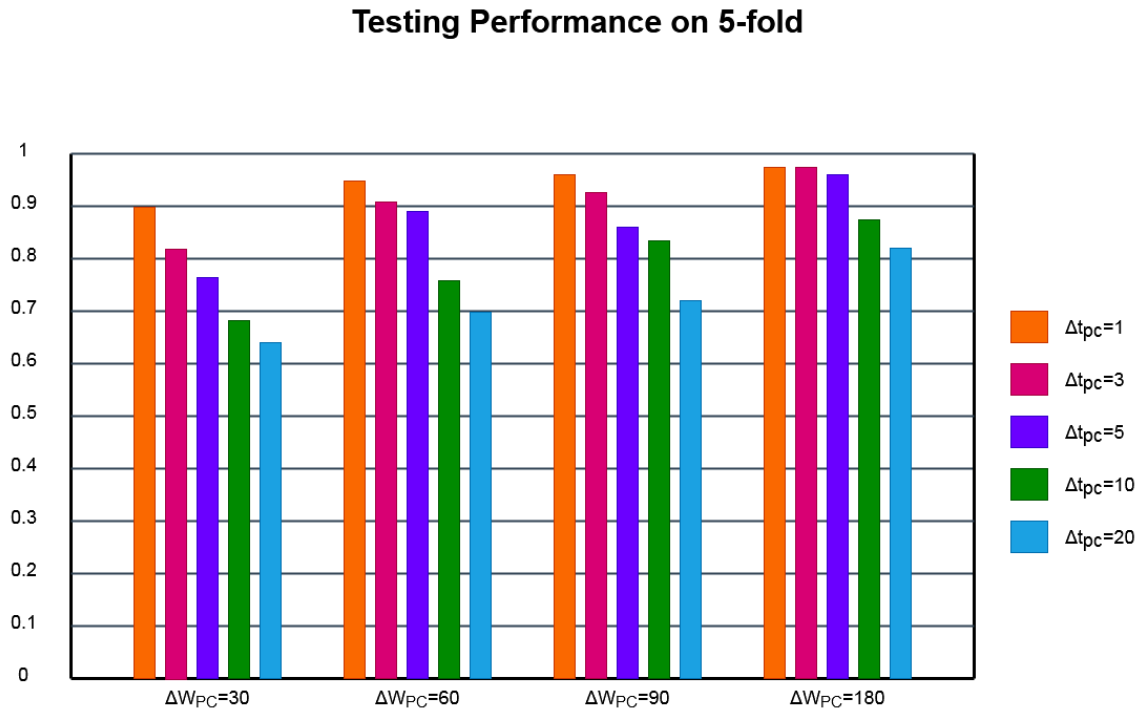


Figure 5.5: Architecture of NN-PAC.

From the histogram we focus our attention on the testing performances. We seek the pair that combines a as small as possible window size with a logical time step parameter. As a whole for the window size we can see that the results stabilize after $\Delta W_{PC} = 60secs$. As for the time step in general we can see that as it increases the performance decreases and for all cases $\Delta t_{PC} = 1sec$ gives the best results. This is a reasonable outcome in the sense that the more frequently the network makes a predictions the more prone it will be to catch



(a) NN-PAC Training Performance



(b) NN-PAC Testing Performance

Figure 5.6: NN-PAC Prediction accuracies for different time parameters.

all the pattern transitions. As we stated time step is directly coupled with computation overhead, but we will keep $\Delta t_{PC} = 1sec$ for the rest of our study.

Thus when we use the NN-PAC for online prediction of the pitch angle pattern, we use $\Delta W_{PC} = 60secs$ and time step $\Delta t_{PC} = 1sec$. In Fig.5.7 we show the rise in accuracy of

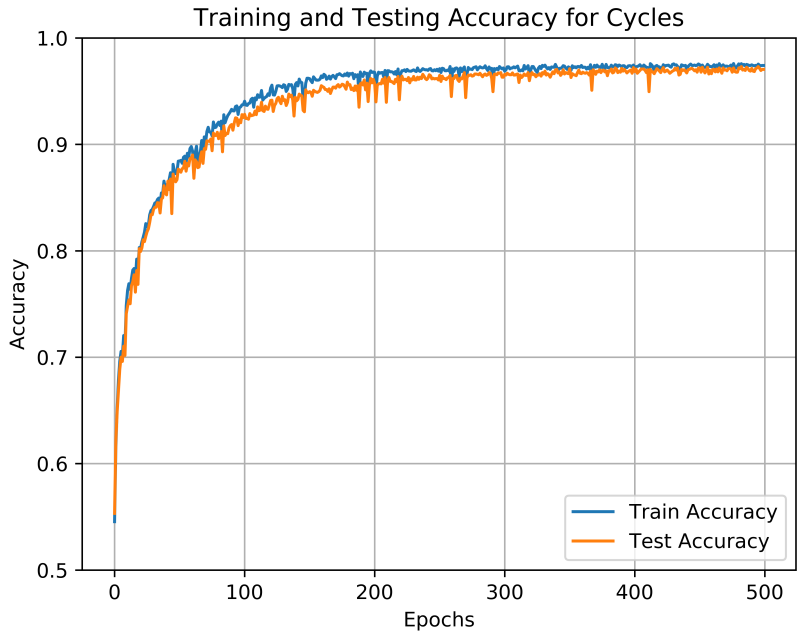


Figure 5.7: NN-PAC: The rise in accuracy of the network during training.

the network and in Fig.5.8 the reduction of the cost function as the epochs progress during the training process.

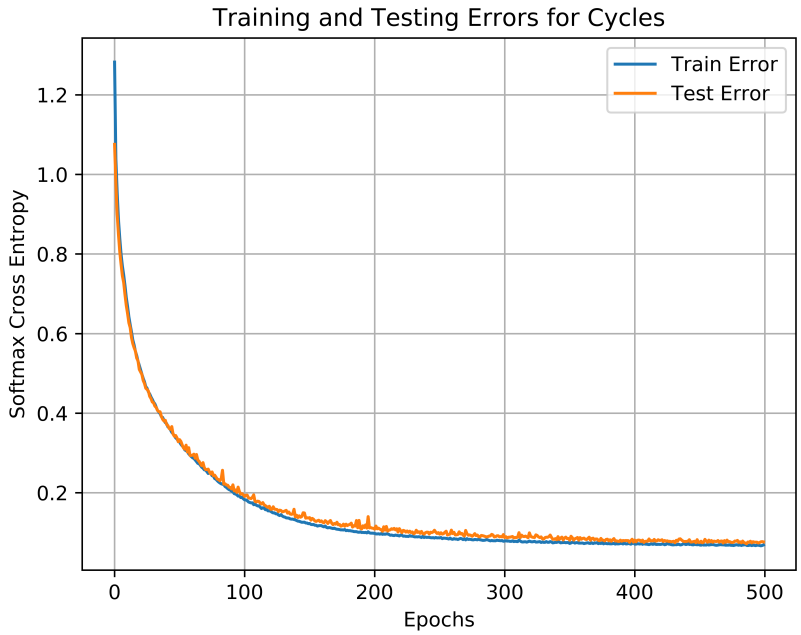


Figure 5.8: NN-PAC: The reduction of the cost function during training.

The output from the NN-PAC is the pitch angle pattern. This prediction will be used as a switch mechanism that will only activate the set of neural networks from the control stage that correspond to the predicted pattern. In this way the hybrid power split is conducted in the time interval $[t_c, t_c+1s]$. In Fig. 5.9 we can see an example of a potential operational

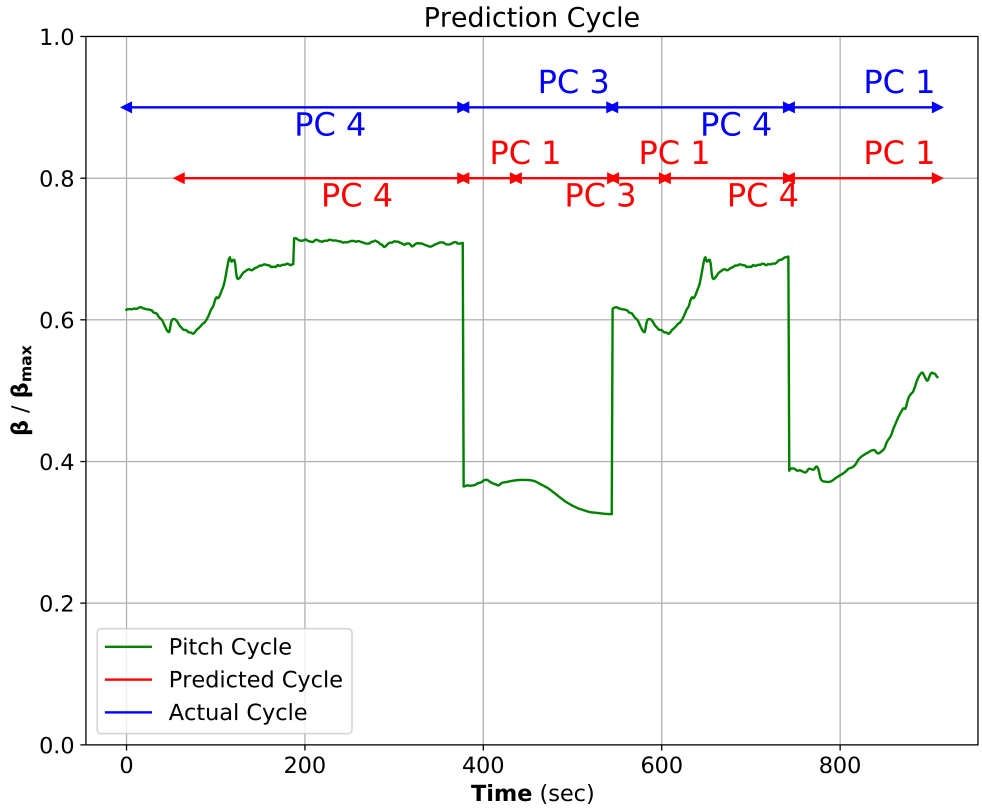


Figure 5.9: NN-PAC Performance Predictions Visualization on a custom Operation Profile CPC1

profile. The red line is the actual labelled pitch angle pattern and the blue line is the prediction of the neural network. At the beginning we can see a delay for 60 seconds, this is completely expected as the algorithm needs at least one window of data to perform the prediction. Furthermore we tested for 5 more custom pitch patterns (CPC) in order to test the accuracy of the network. The prediction results can be seen in Table 5.2. The percentages are measured by summing the correct predictions of the network and then dividing that sum with the total length of the dataset.

Table 5.2: Prediction Accuracy of NN-PAC over 5 test pitch angle patterns

Custom Pitch Pattern	NN Accuracy (%)
CPC1	**92.81**
CPC2	**88.34**
CPC3	**85.74**
CPC4	**90,21**
CPC5	**88.56**

5.2.2 Cruising Trend Prediction

In addition to the prediction of the operation pitch angle pattern, the prediction stage of the framework contains a second type of neural network. The task of this neural network is to predict the short term cruising trend at any given time t, thus the name of these networks is NN-CRT. The motivation of using such a network to make a prediction regarding the short term cruising trend can be explained through analogy with a similar problem in the automotive industry and then by formatting this analogy into a marine application by analysing the throttle control activity of the captain.

5.2.2.1 Motivation

In the automotive industry it is well known, even to the common driver, that the driving style is closely related to the emissions of the engine. According to a similar study [3], it has been proven that in an intelligent power management system the incorporation of a driving trend prediction increases the accuracy of the machine learning approach as a whole. We use this as the motivation we needed to investigate if it the cruising trend plays an important role in a marine applications.

In order to understand what features will be necessary for our network in order to make a realistic prediction, one must understand how the vessel speed is controlled by the captain. If the vessel employs a fixed pitch propeller (FPP) for it's propulsion, then in that case the cruising trend would have been the short term decision made by the captain to reduce or increase the speed of the engine by adjusting the position of the throttle lever. On the other hand if a controllable pitch propeller (CPP) is used, then the captain does not actually manipulate the speed of the engine but instead with the use of the throttle lever he is changing the pitch angle of the installed propeller blades.

5.2.2.2 Features Extraction

The process of making a trend prediction will also happen on a moving time frame and since it's a short term decision we will use a different pair of time parameters to solve this problem compared to the Pattern prediction. To predict the cruising trend at time t_c, we will again extract features from the operation profile in the segment $[t_c - \Delta W_{CT}, t_c]$, where ΔW_{CT} is the window size of the segment that we use to make the prediction at time steps $k\Delta t_{CT}$, $k = 1, 2, \ldots$ and that predictions will be used for calculating the optimal control strategy in the future period $[t_c, t_c + \Delta t_{CT}]$. Fig 5.2 illustrates the functionality that we described above where the x-axes represents time and the y-axis is the normalised β pitch angle.

The NN-CRT is trained using the features in Table 5.3 that we extract from the pitch angle operation profile of the vessel and the outcome of the prediction can be one of the six classes described in Table 5.4. The quantitative criterion is used in order for us to automatically label all the data that we are going to use from the 5 Pitch Angle Patterns.

At this point it's important to note that neural networks can be used to make a prediction about the cruising trend with either of the aforementioned propulsion schemes. This is due to the broad usefulness and generalization capabilities of the neural networks as non-linear solvers. For this study we will only focus on the controllable pitch propeller case but the features we will use as inputs to our network can also work in the fixed pitch propeller case where someone should use the speed operation profile instead of the propeller pitch

Table 5.3: The selected features for the Prediction of the Cruising Trend

Feature Name	Description
β_{max}	Maximum Pitch Angle in $[t - \Delta W_{CT}, t)$
β_{min}	Minimum Pitch Angle in $[t - \Delta W_{CT}, t)$
β_{ave}	Average Pitch Angle in $[t - \Delta W_{CT}, t)$
β_{in}	Pitch Angle at $t - \Delta W_{CT}$
β_{out}	Pitch Angle at t
ω_{ave}	Average Pitch Acceleration in $[t - \Delta W_{CT}, t)$

Table 5.4: The six classes of Cruising Trend

Cruising Trend Class	Description	Quantitative Criterion
0	Dead Stop	$\beta_{ave} = 0$ & $\beta_{max} = 0$ & $\beta_{min} = 0$
1	Low Speed Cruise	$0 < \beta_{ave} < 0.3$ & $\omega_{ave} < 0.001$
2	Mid Speed Cruise	$0.3 < \beta_{ave} < 0.75$ & $\omega_{ave} < 0.001$
3	High Speed Cruise	$0.75 < \beta_{ave} < 1$ & $\omega_{ave} < 0.001$
4	Acceleration	$\omega_{ave} \geq 0.001$
5	Deceleration	$\omega_{ave} < -0.001$

angle operation profile.

5.2.2.3 Neural Prediction Training

We developed NN-CRT, a multilayer multiclass fully connected feedforward neural network, with the task to predict the cruising trend in the short-term future according to the previously described process.

For the architecture of this network we follow the same process as we did with NN-PAC. We train all potential networks with the same 5-fold cross validation method base on the pair of $\Delta W_{CT} = 9$ and $\Delta t_{CT} = 1$. All networks have 1 hidden layer with $5, 10, 15, 20, 25, 30$ neurons. In Fig.5.10 we show the training and testing accuracy as a function of the neurons in the hidden layer. Even with 5 neurons the network is performing quite remarkably, but for optimal results we chose 20 neurons in the hidden layer. The final structure of the network is shown in Fig.5.11.

Again in order to determine the best choice for window size ΔW_{CT} we experiment with various values. The detailed final training and test results, i.e. prediction accuracy of the NN-CRT for all considered pairs, are presented in Fig.5.12 for training and testing, where the values of the parameters are $\Delta W_{CT} \in \{5, 9, 15, 30\}$ and $\Delta t_{CT} = 1$.

Although the performance is very high with every parameter set, it is important to remember that when we are using big ΔW_{CT} we are taking into account possibly obsolete information. Therefore in an online environment we believe that $\Delta W_{CT} = 9secs$ is the best choice as it catches all the important transient shifts while still being computationally feasible. Finally in Fig.5.13 we record the growth in accuracy of the network as well as the decline of the cost function as the epochs progress during the training process.

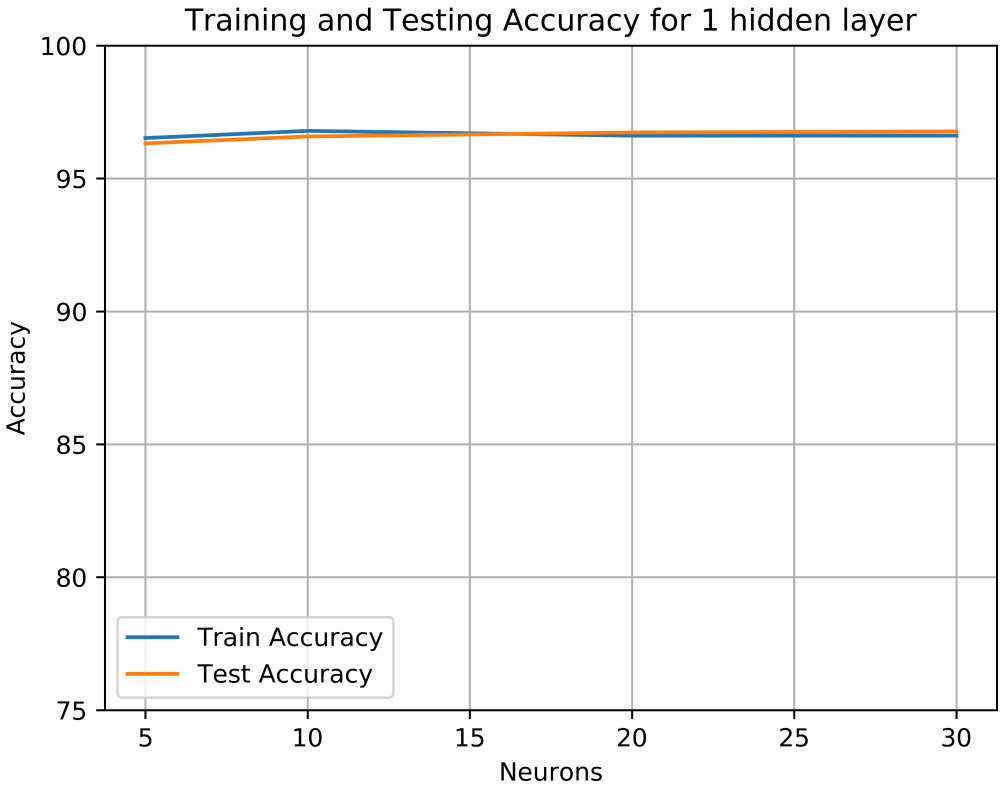


Figure 5.10: Number of hidden layer neurons vs accuracy for NN-CRT with 1 hidden layer

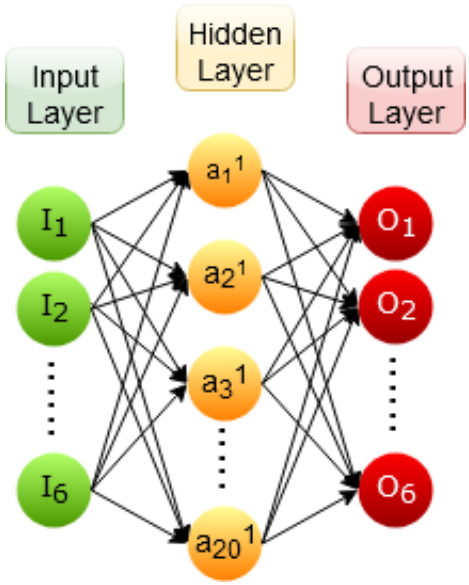


Figure 5.11: Architecture of NN-CRT.

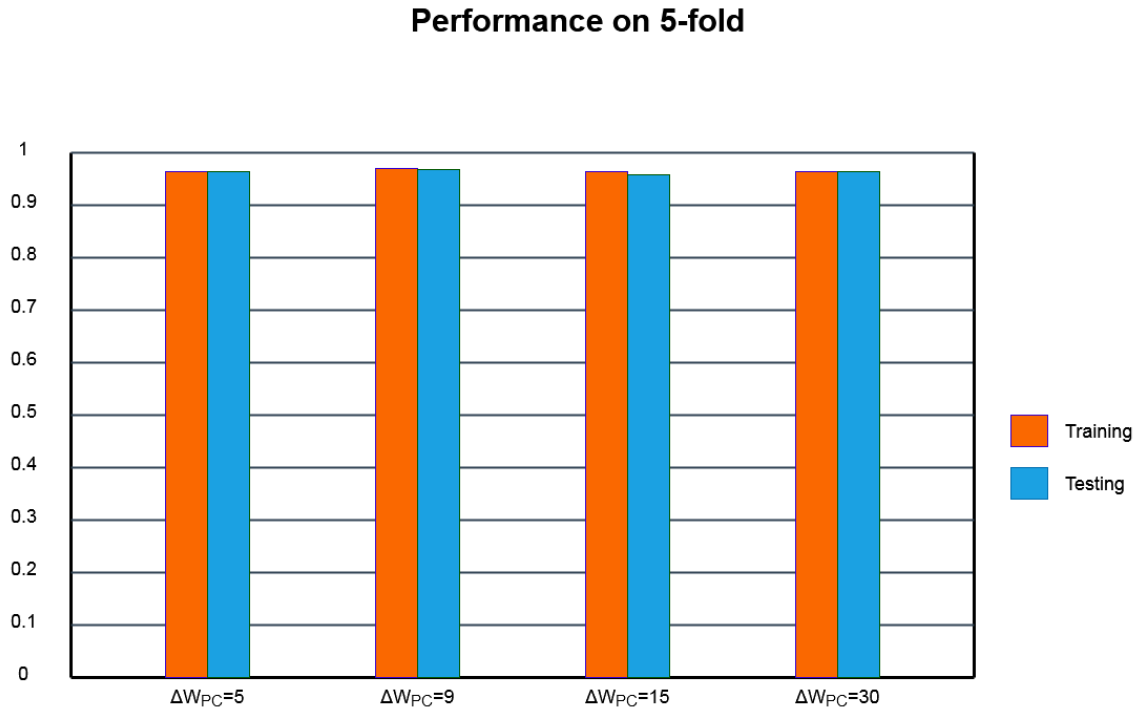
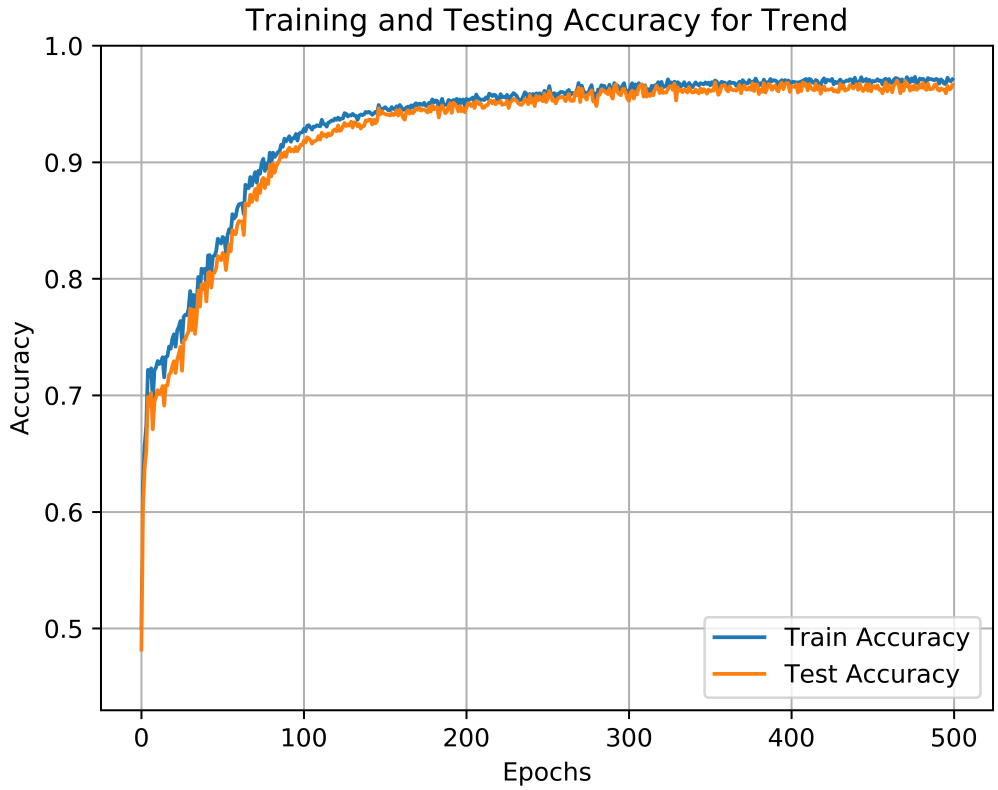
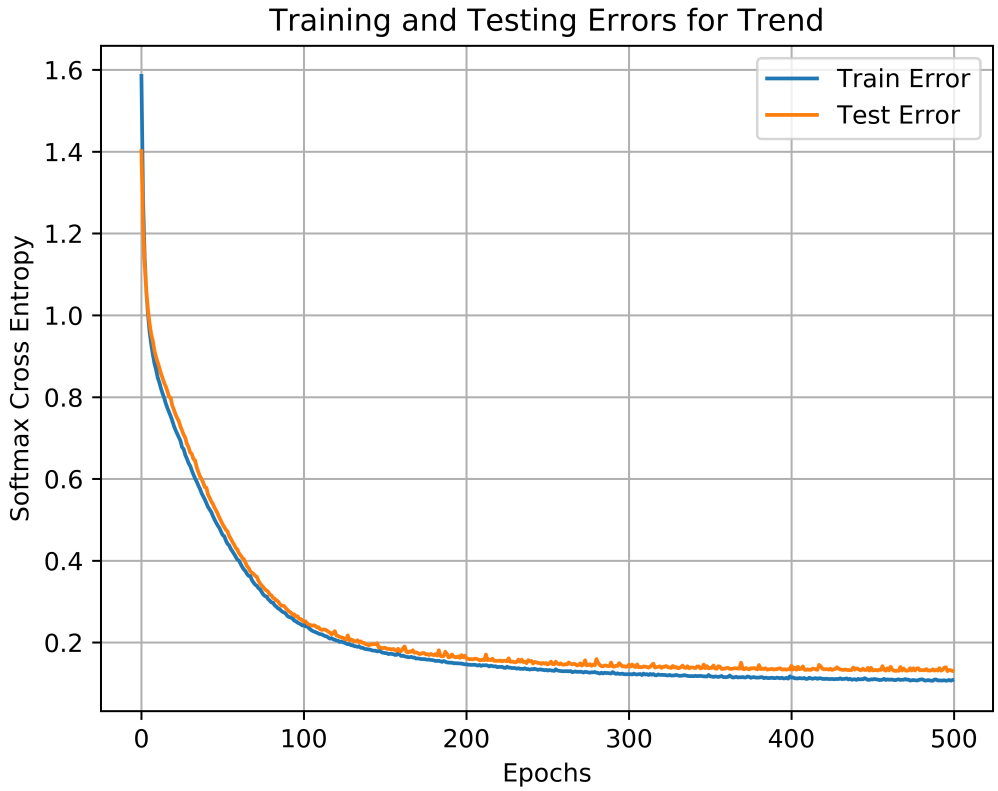


Figure 5.12: NN-CRT Prediction accuracies for different time parameters.



(a) The rise in accuracy of the network during training.



(b) The reduction of the cost function during training.

Figure 5.13: NN-CRT training process evaluation metrics.

5.3 The Control Stage

The machine learning algorithm NN-CON is developed with the goal of learning the optimal power split settings for all standard patterns and then generalize this knowledge to online energy management through neural learning for any operation. The objective is for the Electric Motor to launch operation when it's needed and thus assist the internal combustion engine by compensating the required torque. This is done to ensure a steadily increasing rate of torque from the engine instead of pushing it to accelerate fast in order to manage the load on it's own. To accomplish this, as we stated earlier, two sets of neural networks will be employed to emulate the Nonlinear Model Predictive Control scheme based on the Hybrid power plant of Hippo-2 testbed.

We commence our work flow by creating the necessary data for the training process of the neural networks. After acquiring the data we device the training method for the networks and then we follow up with the validation stage that is presented in the next chapter. From this point on, the proposed machine learning framework is complete and its good standing condition can be tested as a whole. For our first validation we compare the power split performed by the framework against the control of the NMPC for a real life maritime scenario.

5.3.1 Data Creation Phase

For the creation of the necessary data we combine the high fidelity vessel model from Section. 2.2 with the NMPC and MHE scheme presented in [6]. With the model ready and set, we use the 5 Standard Pitch Angle Patterns presented in 4.1 as the loading condition inputs for the propeller model. One by one we simulate the operation of the complete model and we collect a file with the selected variables in the form of time sequences. The labels of the acquired data from the simulation are presented in Table.5.5.

For this simulation an important step was the realisation that the developed framework is destined to replace the NMPC controller and therefore an equivalent sample time of control commands is needed. For this reason the simulation of the patterns is set to 173 seconds with a sample time of $0.1s$. In this way we still have 1730 values but we are able to capture the correct control commands from the NMPC. This result is opposed to a case of a slower sample time like $1s$, where the motor would not get activated as the condition would not be transient enough.

Table 5.5: The acquired features from the simulation.

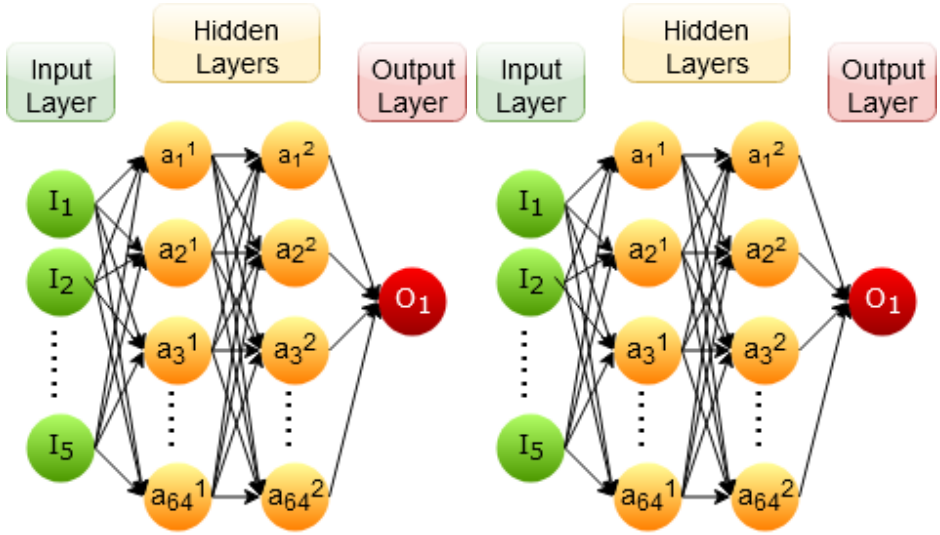
Acquired Variable	Description
TQ_{ENG}	Diesel Engine Torque
TQ_{MOT}	Electric Motor Torque
TQ_{LOAD}	Load Torque
V_{SHIP}	Vessel's speed
SOC	Battery's State of Charge
N_{ERROR}	Engine Speed Deviation from 1800 rpm

5.3.2 Engine and Electric Motor Torque Control

We develop two sets of neural networks. The first one is NN-ENG that is employed to predict the torque of the diesel engine TQ_{ENG} and the other one is NN-MOT that predicts

the torque of the electric motor TQ_{MOT}. Each set contains 5 neural networks NN_i^{ENG} and NN_i^{MOT} that have been developed to learn the optimal power split for PC_i where $i = 1, 2, \ldots, 5$. According to the notation given in Table.5.5, the inputs for NN_i^{ENG} and NN_i^{MOT} are $N_{ERROR}(t)$, $V_{SHIP}(t)$, $TQ_{LOAD}(t)$, $SOC(t)$ and $CT(t)$ where $CT(t)$ is the cruising trend at time t and it can take one of the 6 possible values defined in 5.4.

The architecture of the neural networks for the control stage can be seen in Fig.5.14. It should be noted that the hidden layers for each network and for each different pattern can be different but for this study we kept everything the same.



(a) The Diesel Engine control neural network NN_i^{ENG}

(b) The Electic Motor control neural network NN_i^{MOT}

Figure 5.14: Architecture of the energy management Neural Networks Operation Pitch Angle β Patterns

The performance of the neural networks is measured by Mean Absolute Error(MAE) defined as

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|tar_i - output_i| \tag{5.1}$$

Where N is the length of the pattern, so in our case $N = 1730$, output is the NN output and target is the truth value. In order to show that the trend prediction is improving the performance of the control phase networks we tested both with and without it. In Table.5.6 we present the testing results of the neural network compared with the NMPC while the trend prediction was inactive and in Table.5.7 while it was working. It is clear that with the prediction the performance increased in all but 2 cases.

Below we present the comparison between the neural networks and the NMPC for every pattern.

5.3.3 Training Analysis

Starting with the observed MAE we conclude that for all cases we are inside the acceptable region and from the graphs it can be observed that the results generated by the neural

Table 5.6: Testing Accuracy of NN-ENG and NN-MOT over the 5 pitch angle patterns

Pitch Pattern	$NN_{ENG-NOCRT}$ (MAE)	$NN_{MOT-NOCRT}$ (MAE)
PC 1	**12.74**	**9.72**
PC 2	**13.41**	**4.0217**
PC 3	**7.62**	**4.78**
PC 4	**22.85**	**14.72**
PC 5	**43.44**	**24.86**

Table 5.7: Testing Accuracy of NN-ENG and NN-MOT over the 5 pitch angle patterns

Pitch Pattern	$NN_{ENG-CRT}$ (MAE)	$NN_{MOT-CRT}$ (MAE)
PC 1	**11.55** (-9.3%)	**7.94** (-18.3%)
PC 2	**13.81** $(+2.9\%)$	**4.90** (-5.8%)
PC 3	**7.14** (-6.3%)	**4.07** (-14.8%)
PC 4	**24.46** $(+7.04\%)$	**12.06** (-18.1%)
PC 5	**35.62** (-18.1%)	**24.21** (-2.6%)

networks are very close to that of the NMPC method. Although deviations do exist this is actually a wanted behaviour as one of the main reasons that we are using neural networks is to generalise well in the cases of unseen data, therefore a complete fit would in effect hinder the performance due to overfitting as we discussed in 3.5

Pitch Pattern 1 was chosen to represent the case of a mild acceleration at start. In this pattern the NMPC forces the EM to lightly assist the diesel engine at the beginning and therefore an increase in torque occurs. When the full acceleration has taken place the EM shifts to generating mode to recharge the battery and the required torque is mainly provided by the diesel engine. In both occurrences, the NN correctly captures the desired effect and because pattern 1 is the only mild acceleration from idle to $55 - 58\%\beta_{max}$ the expected behaviour is for this set of networks to generalise well in their region as they will most likely be the ones activated when this type of operation profile occurs.

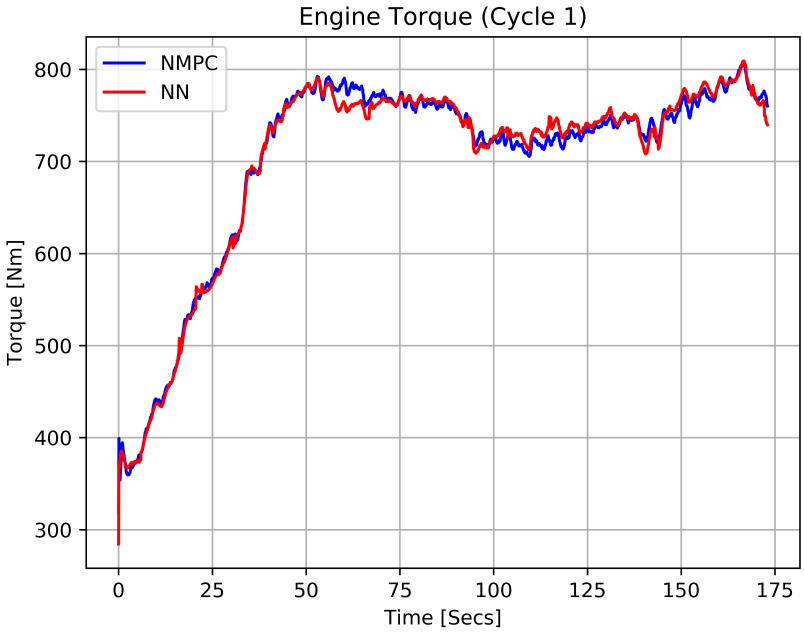
Pitch Pattern 2 represents the case of fast acceleration from idle to mid range operation values of around $42 - 45\%\beta_{max}$ and fast deceleration from mid range to low. Once again, the EM is required to assist the engine in the acceleration but this time a faster response is needed compared to the one in pattern 1. The NN captures this action and because only this pattern was chosen to represent this type of acceleration it is also expected to perform at a high level in it's region.

Pitch Pattern 3 is not necessarily the most active pattern but this type of operation profile usually occurs before a fast acceleration. In the operation of a tug vessel it can represent the case where a call has been made for a client to be towed subsequently after the one before him and the captain chooses not to dock but to remain in low speed cruising which translates to about $30 - 40\%\beta_{max}$. For this case since the first part is in the range of Pattern 1 as well as in Pattern 2 a difference in the rate of acceleration again makes the all important distinction possible. Therefore when this Pattern is activated we expect once more for the framework to conduct a correct power split.

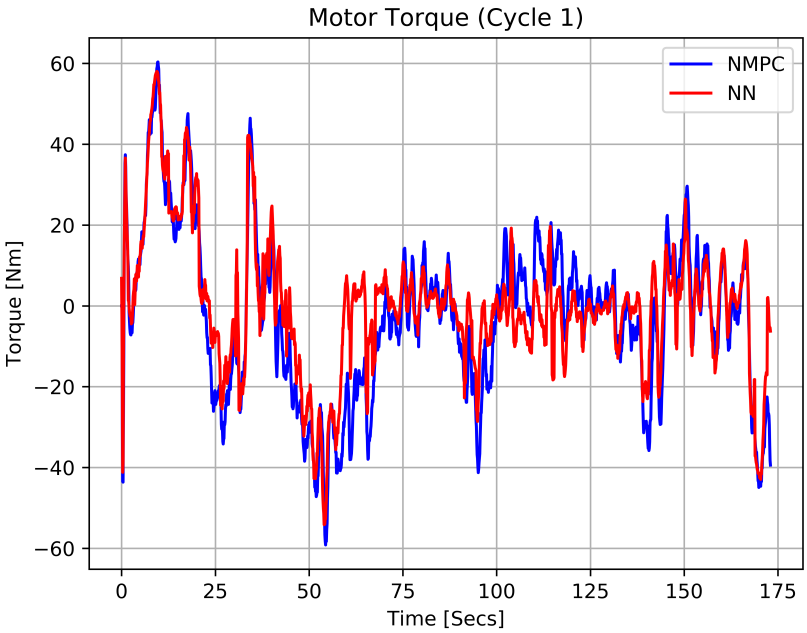
Pitch Pattern 4 is a representing the case of a very fast acceleration from idle to high range of $70 - 72\%\beta_{max}$. This pattern is tug vessel operation would mostly occur after the client has been given to the captain and he has to respond to the call at short time notice. Since it is a fast transient the diesel needs heavy assistance from the motor at the beginning and the NN picks that intention as well. Once more the uniqueness of this pattern translates

into high expected performance when this pattern is switched on.

Finally Pitch Pattern 5 covers a range of operation regions and therefore is expected to occur the most out of all patterns. The unique characteristic of this pattern is the fast deceleration from high values to idle. This fast transient again is handled by the EM in order to give room to the heavier diesel to slow down with a low rate pace. Although this pattern is the one where the biggest MAE value arises, this is mostly because it is characterised by transient shifts and lacks steady states and that initially made us believe that this NN would not perform at acceptable performances. As we will see in the next chapter with the simulation result due to the effective training of the networks and the avoidance of overfitting this network fills it's purpose.

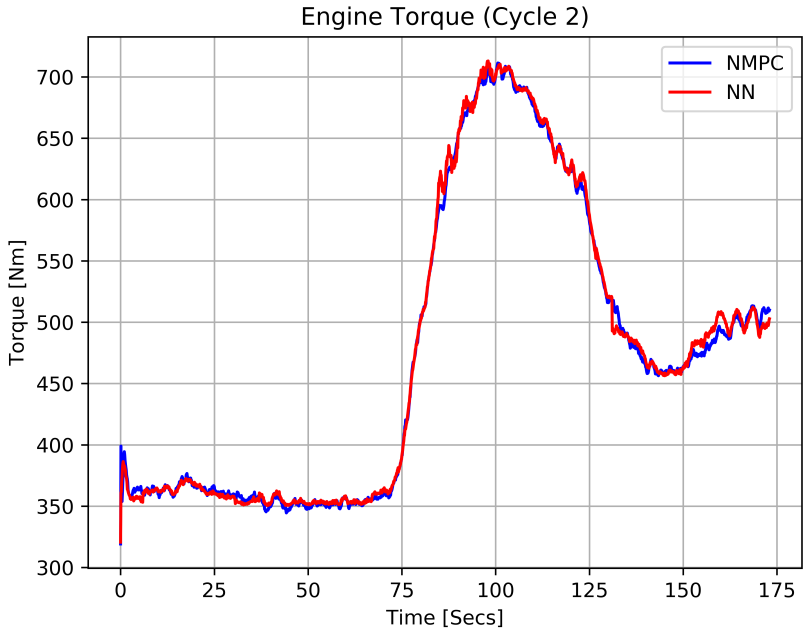


(a) The Diesel Engine Torque Command

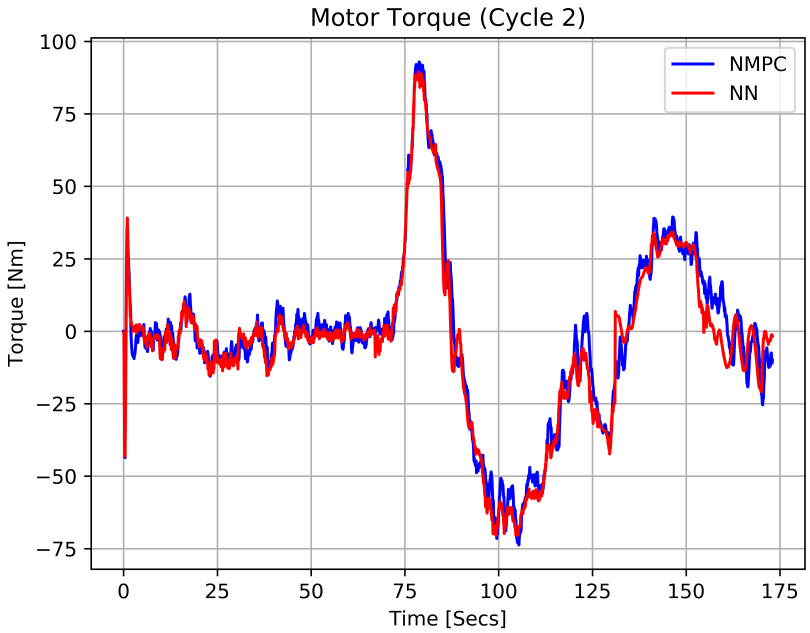


(b) The Electric Motor Torque Command

Figure 5.15: Power split comparison between NMPC and Neural Network for Pitch Angle β Pattern 1

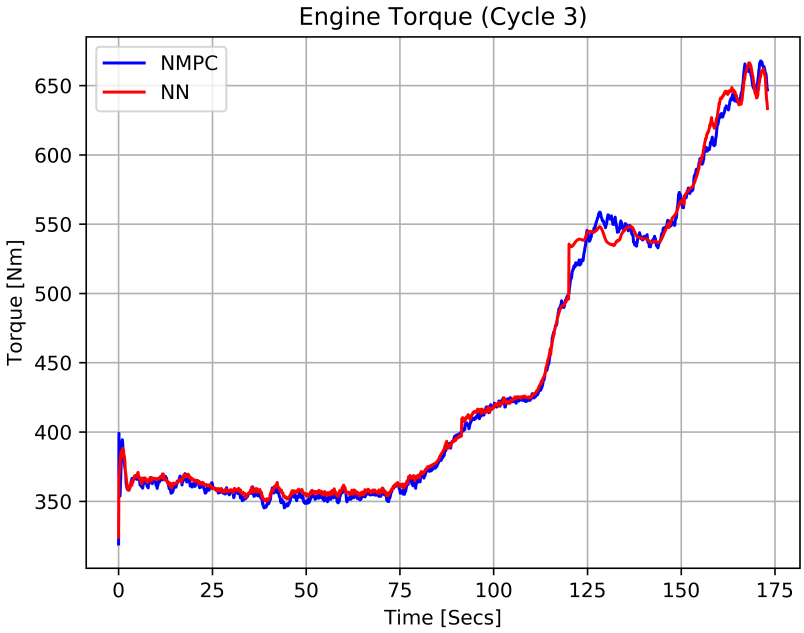


(a) The Diesel Engine Torque Command for Pattern 2

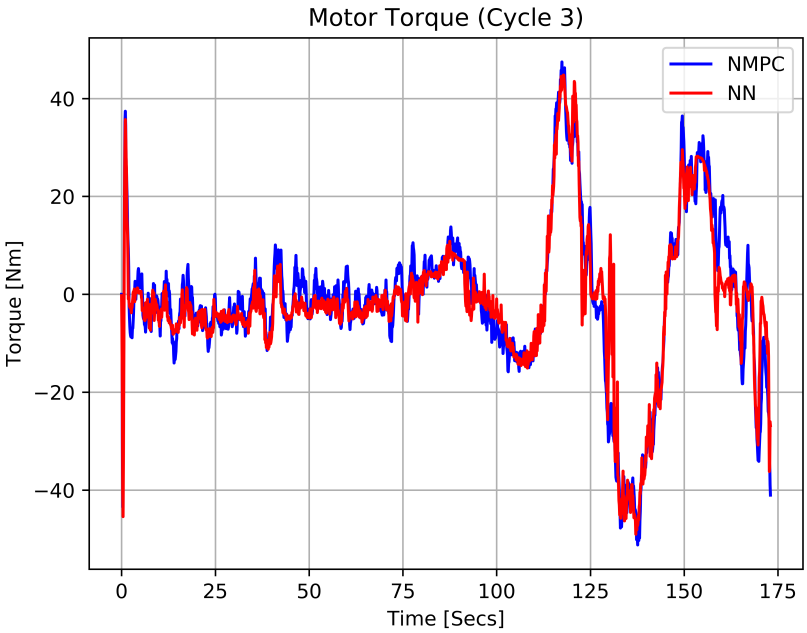


(b) The Electric Motor Torque Command for Pattern 2

Figure 5.16: Power split comparison between NMPC and Neural Network for Pitch Angle β Pattern 2

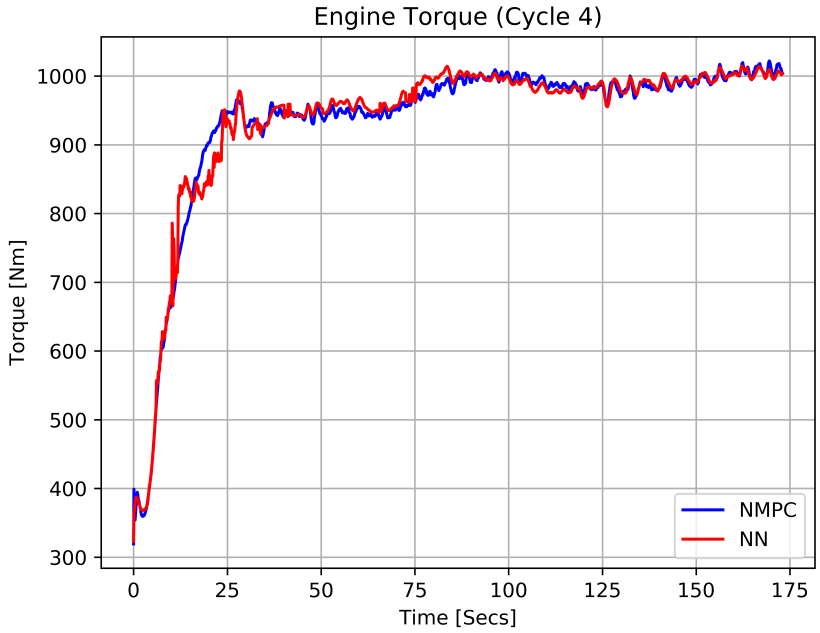


(a) The Diesel Engine Torque Command

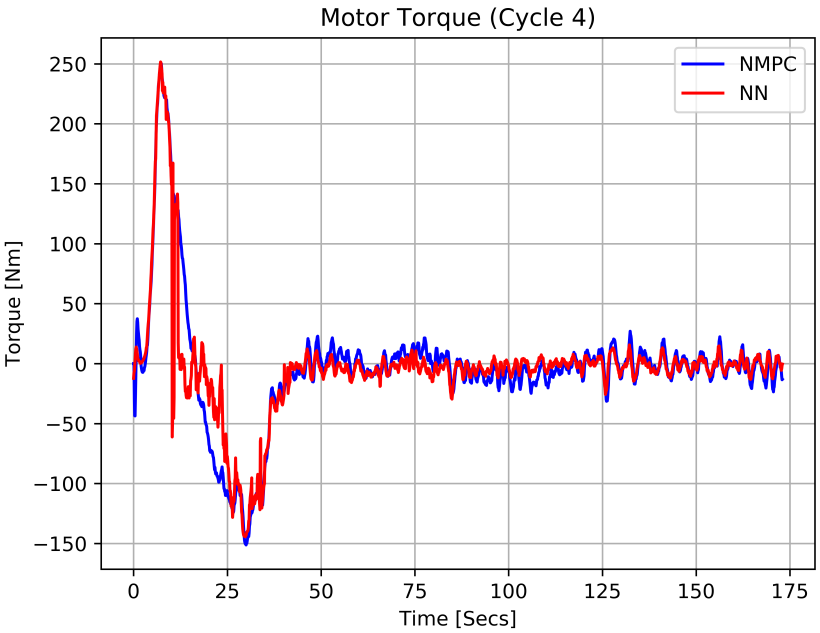


(b) The Electric Motor Torque Command

Figure 5.17: Optimal Power split comparison between NMPC and Neural Network for Pitch Angle β Pattern 3

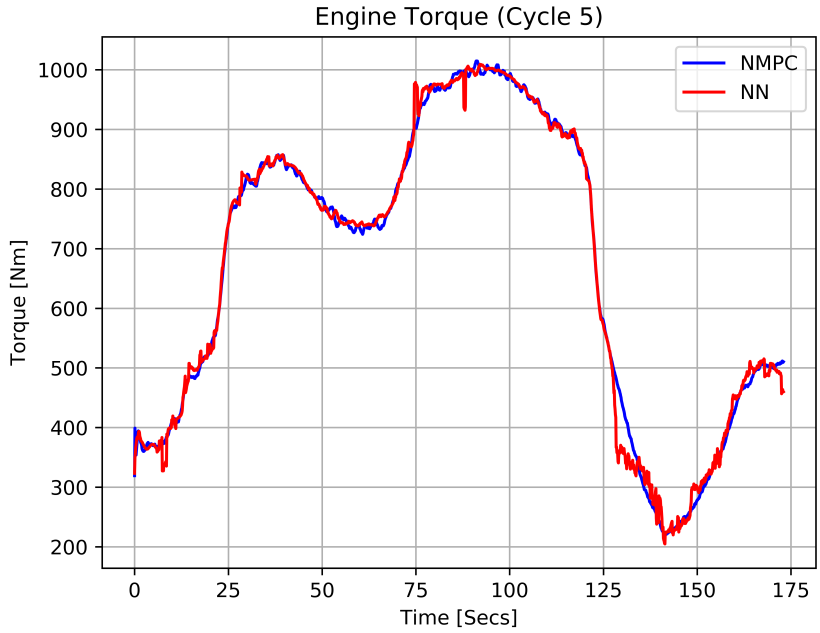


(a) The Diesel Engine Torque Command

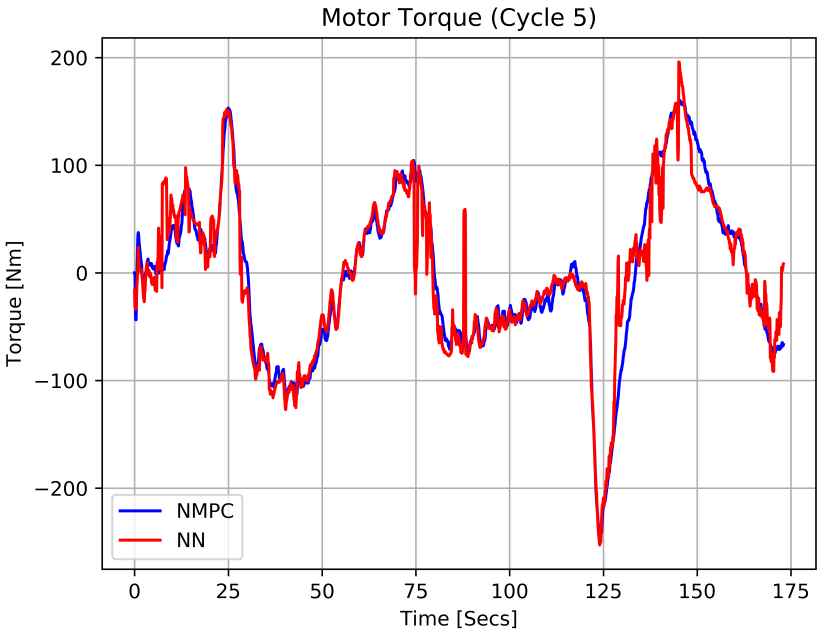


(b) The Electric Motor Torque Command

Figure 5.18: Optimal Power split comparison between NMPC and Neural Network for Pitch Angle β Pattern 4



(a) The Diesel Engine Torque Command



(b) The Electric Motor Torque Command

Figure 5.19: Optimal Power split comparison between NMPC and Neural Network for Pitch Angle β Pattern 5

Chapter 6

Results of the Framework

For the validation of the developed framework, four very common real marine applications, were chosen to be simulated. The simulations were conducted by employing the same propeller load simulator that we used for the training of the neural networks. In every case the purpose was different but above all we sought for stability and therefore the neural network framework is required to keep the system in a controllable state for it to be deemed successful.

6.1 Framework Set-Up

Essentially the framework that was presented in 5 was implemented in the simulation environment MATLAB Simulink and it's schematic can be seen in Fig. 6.1. The NMPC controller is still present but it's purpose is not to control the plant but to keep it in a steady state for the first 60 seconds where the neural network that predicts the Pitch Cycle is not actively making any prediction because it's still gathering data to make its first window. Therefore at this time the neural networks of the control stage are also inactive and thus we need a mechanism to keep the system in a steady state. Note that the NMPC here is a luxury and it's not really needed as it is, in fact in real condition the internal controller that most engine ECUs offer would suffice.

Sim. No	*Purpose*	*Engine Speed [rpm]*	*Duration [sec]*
1	6.2 Acceleration	1800	120
2	6.3 Deceleration	1800	200
3a	6.4 Acceleration & Deceleration	1800	200
3b	6.5 Step Acceleration Comparison with NMPC	1800	250

Table 6.1: Index of simulations which were conducted

Here again for easy of presentation we list once more all the necessary attributes of the framework:

- NN-PAC is a classifier with 2 layers of 40 and 30 neurons [Activations: $Tanh$].
- NN-CRT is a classifier with 1 layer of 20 neurons [Activation: $Tanh$].
- The cost function for classification is the softmax cross entropy.
- NN^{ENG} are regression models with 2 layers of 64 neurons [Activations: $ReLU$].
- NN^{MOT} are regression models with 2 layers of 64 neurons [Activations: $ReLU$].
- The cost function for regression is the mean squared error (MSE).
- The tug boat employs a Controllable Pitch Propeller (CPP) and the model accounts for weather disturbances as well.
- The Engine Speed (SE) is referenced at $1800RPM$.
- The Motor torque is limited between $522Nm$ and $-522Nm$.
- The Battery's State of Charge is referenced at 50% and limited to above 20%.
- Movement Horizon Estimator (MHE) is used to estimate the brake load.

The sample time of the controller networks was set up to 0.1 s and for the prediction networks was 1 s. In the next section, results from the experiment are presented. For all simulations the same setup is being used.

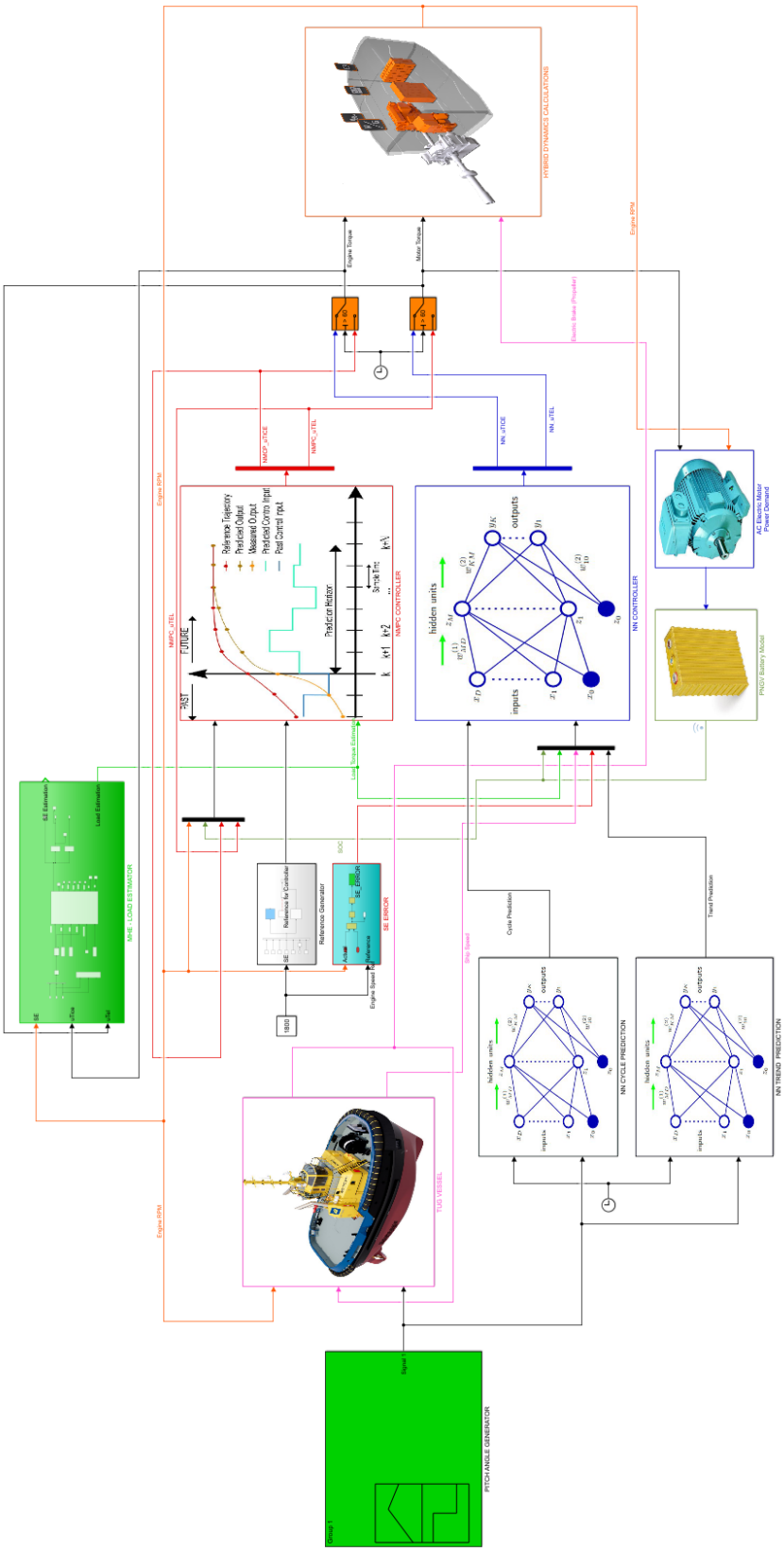


Figure 6.1: NN-PAC: The reduction of the cost function during training.

6.2 Marine Application: Acceleration and Steady

The first scenario was chosen to represent the fast acceleration of a tug vessel. The purpose here is for the EM to initially accelerate the vessel, by giving the required torque and for the engine to follow up but with a slower rate of torque increase. After the EM has given the starting boost it should shift to generating mode and recharge the battery. In Fig. 6.2 we can see this operation profile.

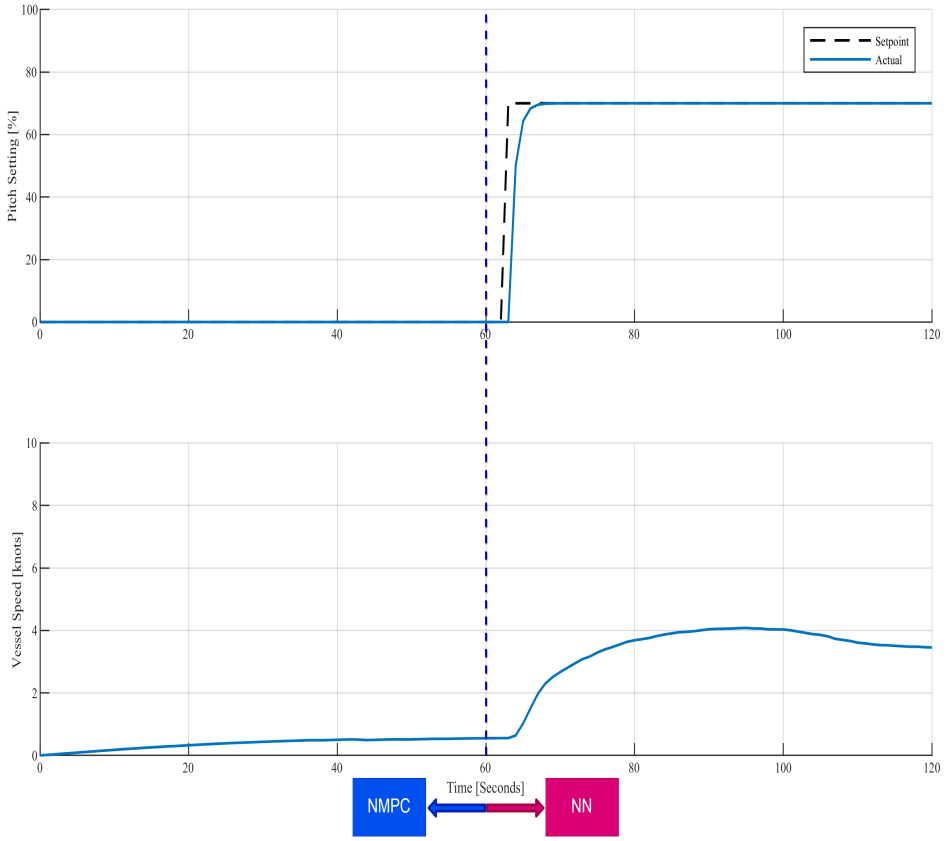


Figure 6.2: Pitch Angle Setting and developed Vessel Speed during acceleration simulation

6.2.1 Simulation Results

In the following figures the simulation results for the fast acceleration scenario in which the speed reference is constant and equals 1800 rpm. Fig., 6.4 and 6.5 are referring to the output torques of the ICE, the EM and the electric brake (load) and the engine speed with the battery's state of charge respectively. Moreover, the results of the predictive phase neural networks are shown in Fig. 6.3.

6.2.2 Simulation Analysis

Initially as we explained the NMPC is holding the plant in idle position. The framework kicks in at 60 seconds and the acceleration commences at 63 seconds.

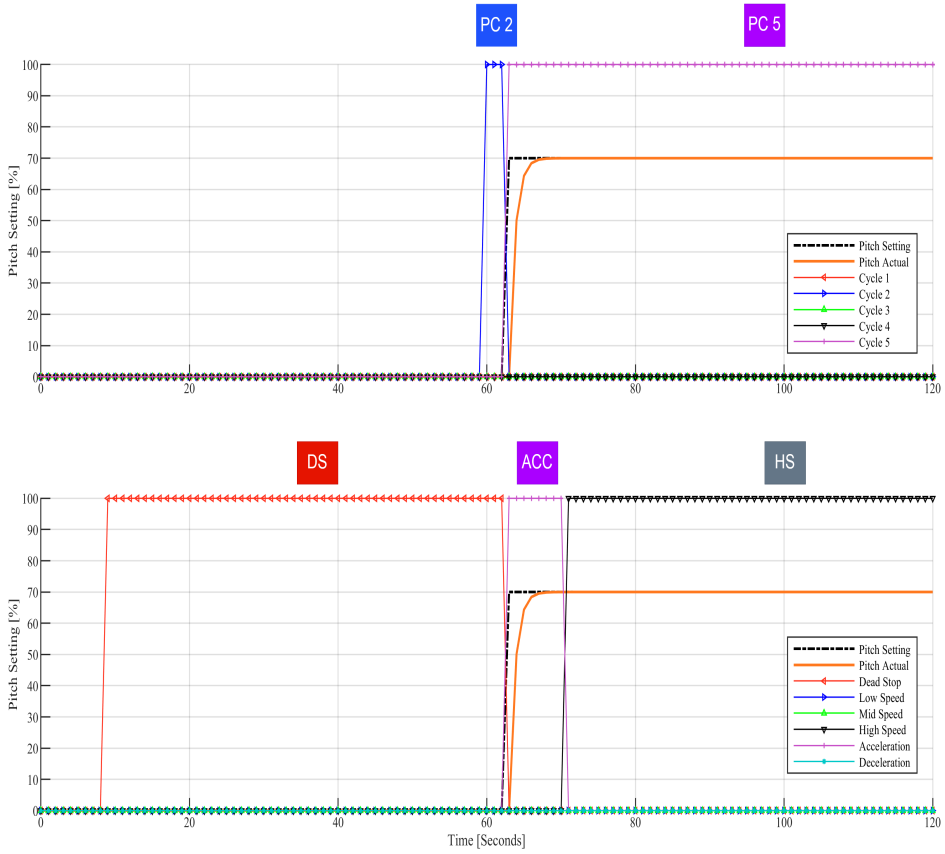


Figure 6.3: Pitch Angle and Cruising Trend Predictions during acceleration simulation

The prediction stage neural networks are accurately predicting the situation they are presented with. In the time range of $60 - 63 secs$ where the plant remains in idle, NN-PAC predicts Pitch Cycle 2 but instantly when the fast acceleration is seen the prediction shifts to Pitch Cycle 5. In reality Pitch Cycle 4 and 5 would be correct here. As for the NN-CRT it starts predicting at $9 seconds$ but the important part is after 60 for here as well. It effectively predicts the acceleration and the High Speed steady afterwards. In general a very pleasing outcome from the prediction networks.

The control stage neural networks also perform very well. As we can see the initial torque is handled by the EM as the ICE torque slowly rises, resembling a steady state condition. According to [16] in steady state, the efficiency of the ICE is increased and its emissions are mostly regulated by the engine internal and after-treatment systems (EGR, SCR). In transient loads, these systems usually have lower efficiency. After the acceleration the motor is generating to recharge the battery back to 50% and although the engine speeds up it's well inside the safe region.

From the results it is obvious that the framework successfully handled this fast acceleration transient and conducted the power split effectively.

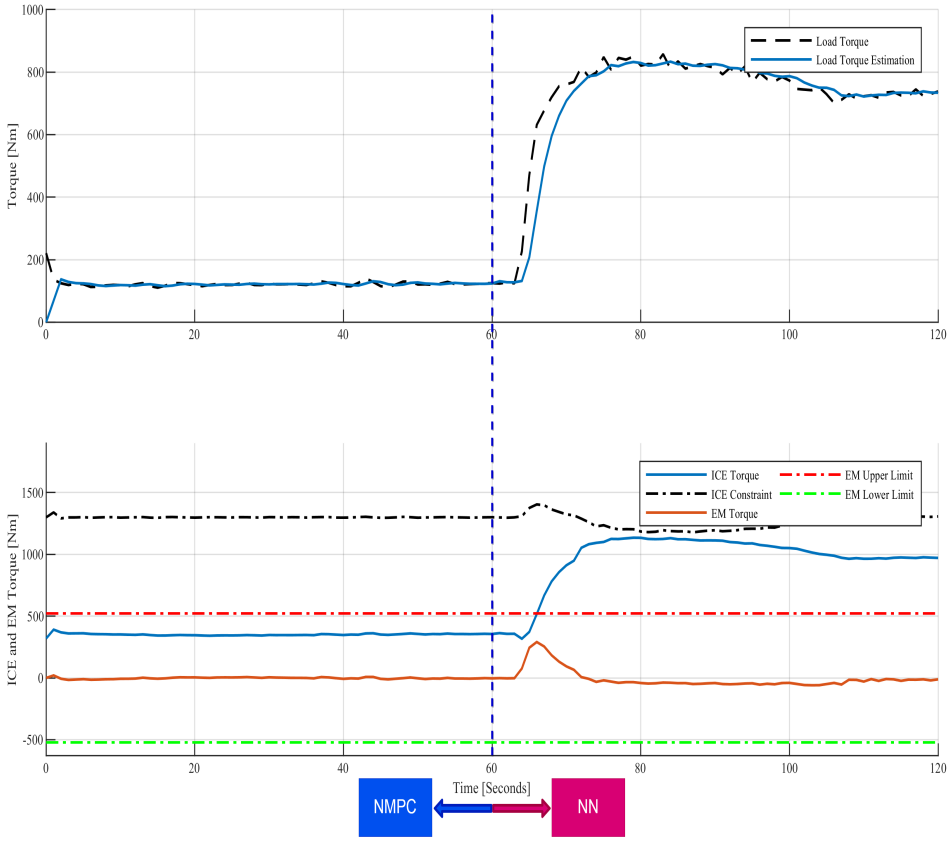


Figure 6.4: Diesel Engine and Electric Motor Torque during acceleration simulation

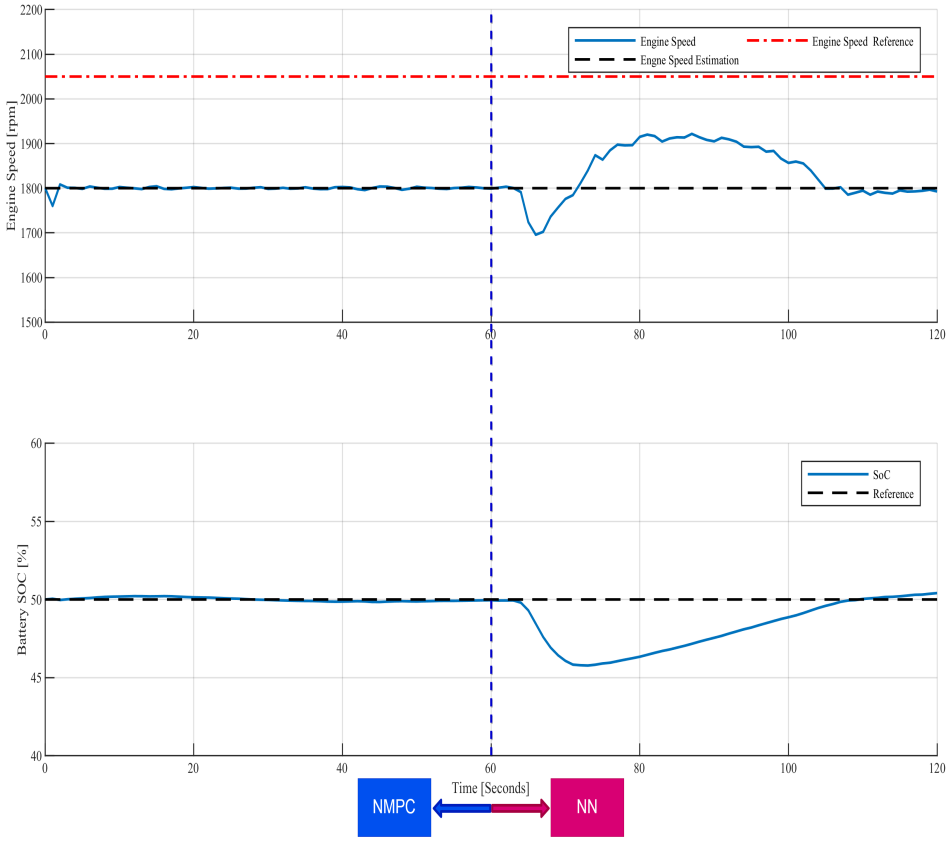


Figure 6.5: Engine Speed and Battery State of Charge during acceleration simulation

6.3 Marine Application: Deceleration

The second scenario was a slow 42s deceleration of the tug vessel, a very common operation manoeuvre for a tug vessel of the considered size. In this case we seek for the EM again to kick in and hold the torque as the engine slows down. Afterwards it is possible for the motor to again change into generating mode to charge the battery for a potential future use. In Fig. 6.6 we can see the operation profile that we just described.

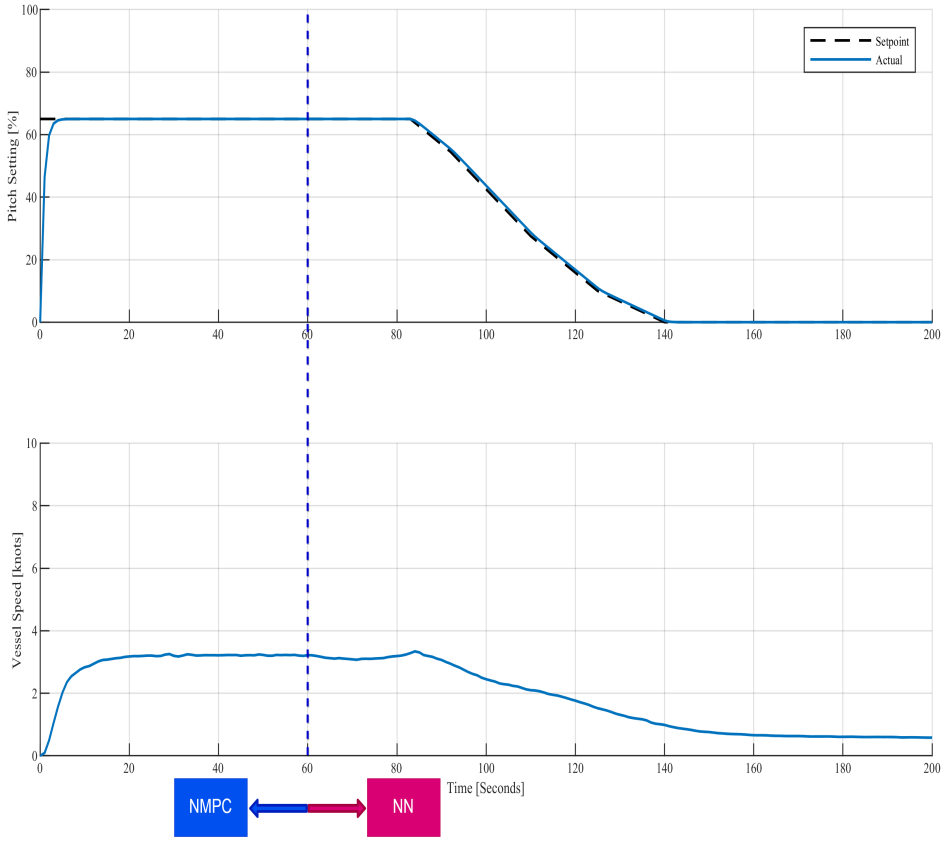


Figure 6.6: Pitch Angle Setting and developed Vessel Speed during deceleration simulation

6.3.1 Simulation Results

In the following figures the simulation results for the slow deceleration scenario in which the speed reference is constant and equals 1800 rpm. Fig., 6.8 and 6.9 are attributing to the output torques of the ICE, the EM and the electric brake (load) and the engine speed with the battery's state of charge respectively. Additionally, the results of the predictive phase neural networks are shown in Fig. 6.7.

6.3.2 Simulation Analysis

The framework again starts at 60 seconds and the deceleration is issued at 83 seconds and ends at 143 seconds.

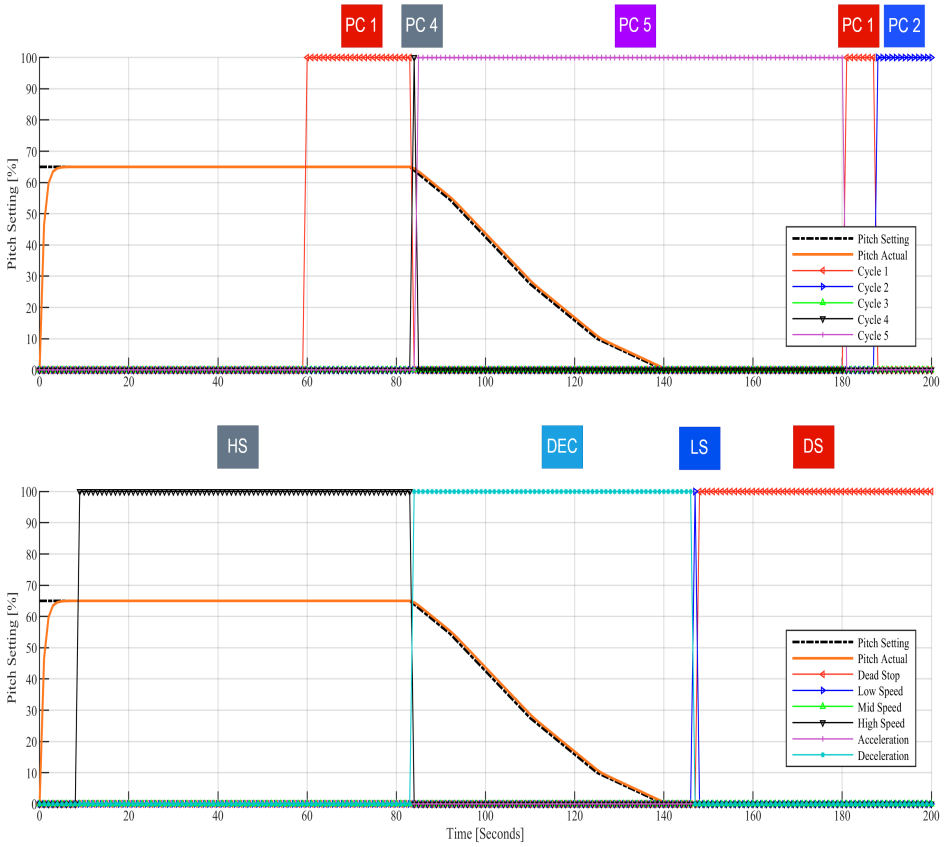


Figure 6.7: Pitch Angle and Cruising Trend Predictions during deceleration simulation

The prediction stage neural networks are accurately predicting the situation they are presented with. In the time range of $60 - 83secs$ where the plant remains at 65%, NN-PAC predicts Pitch Cycle 1 but instantly when deceleration starts the prediction changes to Pitch Cycle 5. Note that only 5 would be correct here and after a point maybe cycle 2 as well. As for the NN-CRT It effectively predicts the deceleration initially but at the end it would have been more acceptable to predict dead stop earlier. This can be explained because we consider dead stop as a total 0 case and obviously this can't happen if the oldest value in the window is not zero which happens in this case. In general again a fairly pleasing outcome from the prediction networks.

The control stage neural networks also perform well. As we can see the initial torque is handled by the EM as the ICE torque very slowly comes to idle. After the acceleration the motor is generating with the aim to recharge the battery back to 50%. Although it passes this mark by 5%, this deviation is not catastrophic. For the matter of engine speed we can see an initial speed up and then a small slow down but both are well inside the safe region again.

From the results we conclude that the framework successfully handled this slow deceleration although a little more effort could be given by the EM as shown by the remaining SOC of the battery.

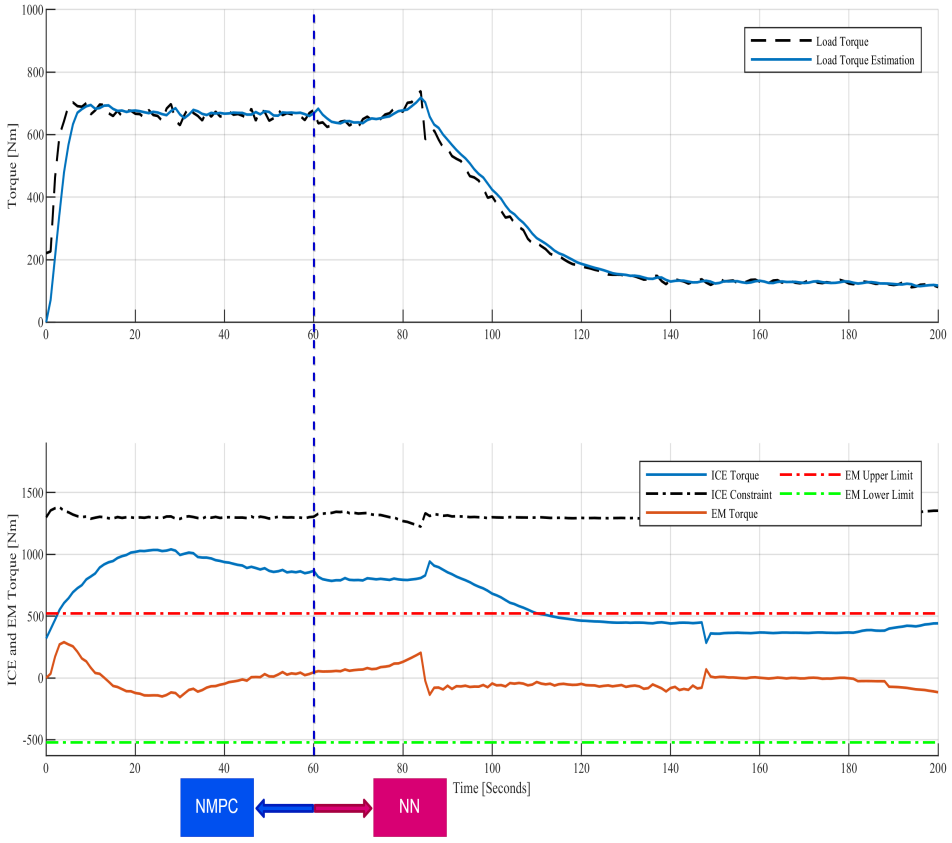


Figure 6.8: Diesel Engine and Electric Motor Torque Control during deceleration simulation

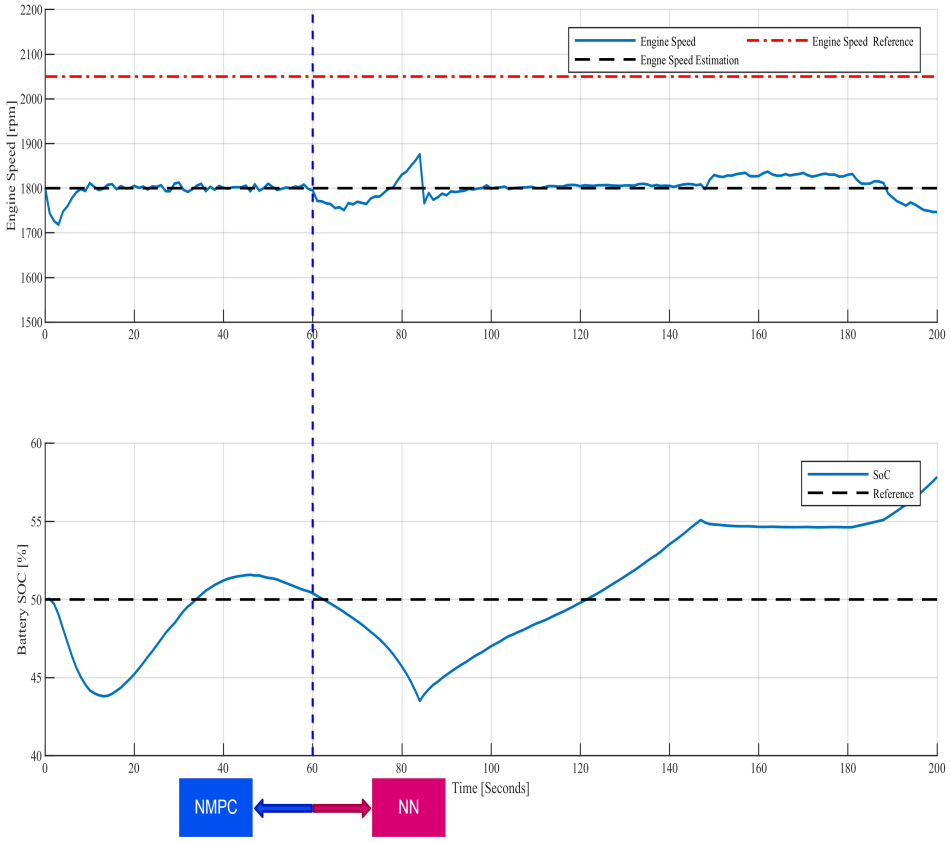


Figure 6.9: Engine Speed and Battery State of Charge during deceleration simulation

6.4 Marine Application: Acceleration and Deceleration

The third scenario was chosen to represent a fast acceleration followed by a deceleration of a tug vessel. The purpose here is to prove the time invariance of the framework and that it can handle both situations one after the other at short time notice. In Fig. 6.10 we can see this operation profile.

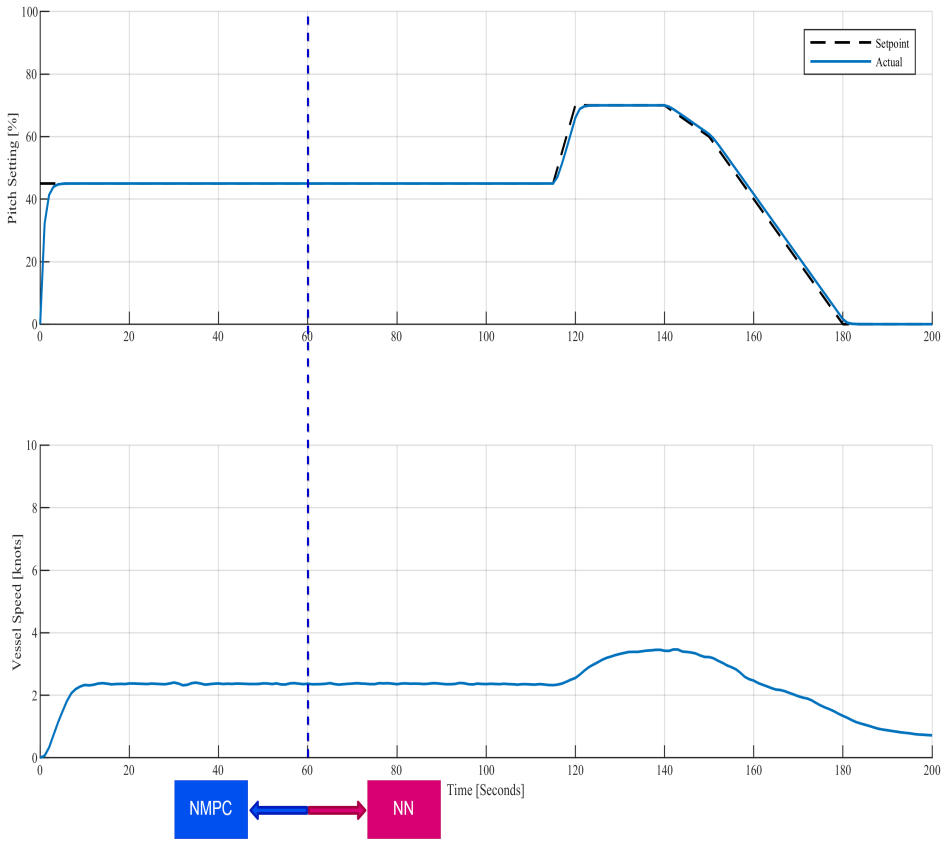


Figure 6.10: Pitch Angle Setting and developed Vessel Speed during simulation

6.4.1 Simulation Results

In the following figures the simulation results for the acceleration and deceleration scenario in which the speed reference is constant and equals 1800 rpm. Fig., 6.12 and 6.13 are attributing to the output torques of the ICE, the EM and the electric brake (load) and the engine speed with the battery's state of charge respectively. Additionally, the results of the predictive phase neural networks are shown in Fig. 6.11.

6.4.2 Simulation Analysis

This scenario starts with mid speed cruising for the first 56 seconds and it's followed by a fast acceleration that commences at 116 seconds and ends at 120 seconds. After that high speed cruising goes on for 20 seconds until 14 and then performs a similar deceleration, as

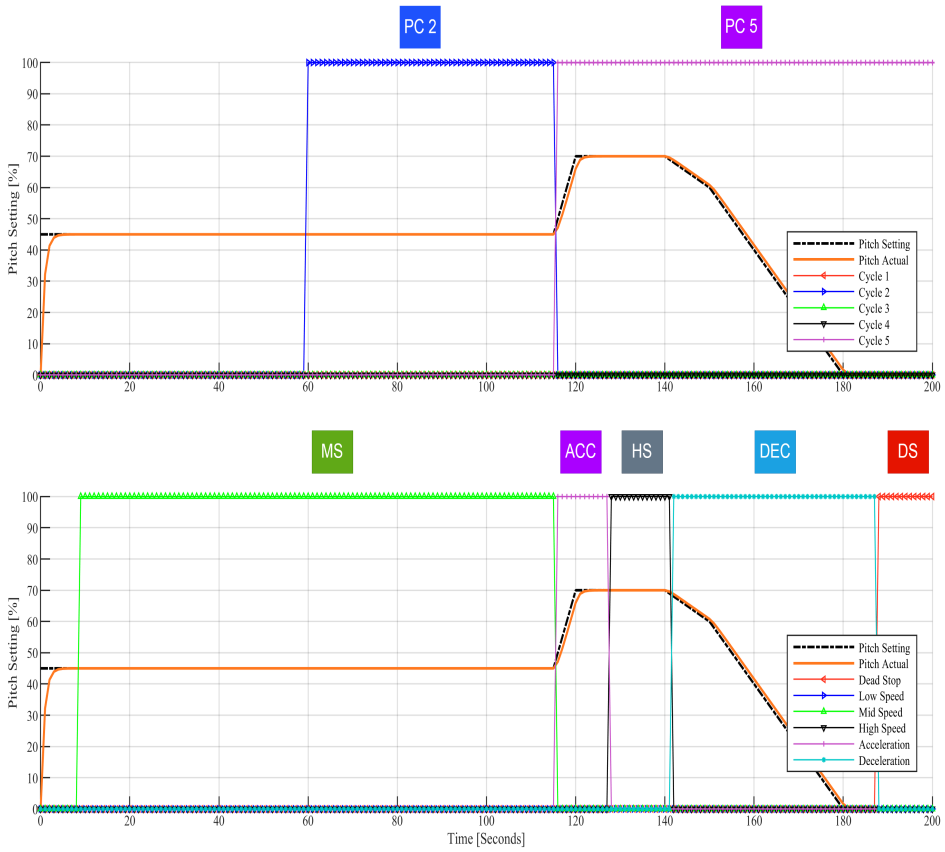


Figure 6.11: Pitch Angle and Cruising Trend Predictions during simulation

in the previous section, to eventually come to a hold at 180 seconds. For the remaining 20 seconds the plant is in idle condition.

For this scheme, the prediction stage neural networks are accurately predicting the situation they are presented with. In the time range of $60 - 116secs$ where the plant remains at 45%, NN-PAC predicts Pitch Cycle 2 which is the only logical choice that it could make and then instantly predicts changes to Pitch Cycle 5 when the acceleration starts. As shown from the acceleration section before the framework favours Pitch Cycle 5 vs 4 in accelerations. For the remaining high speed cruise and deceleration the prediction of cycle 5 is the only logical one. As for the NN-CRT it adequately predicts the Mid speed steady and the High Speed steady afterwards. The same deviation in acceleration and deceleration occurs here as with the discussion we made before. In general a very satisfactory outcome once again by the prediction networks.

The control stage neural networks also perform very well. As we can see the initial torque is handled by the EM when the acceleration starts as the ICE torque slowly increases. After the acceleration the motor is generating with the aim to recharge the battery back to 50% but again it passes this mark by 5% as in the section before, this deviation is again not bad. As for the engine speed it remains for all the simulation very close to the reference value.

Winding up, this fairly transient scenario was handled successfully by The framework as it effectively conducted the power split.

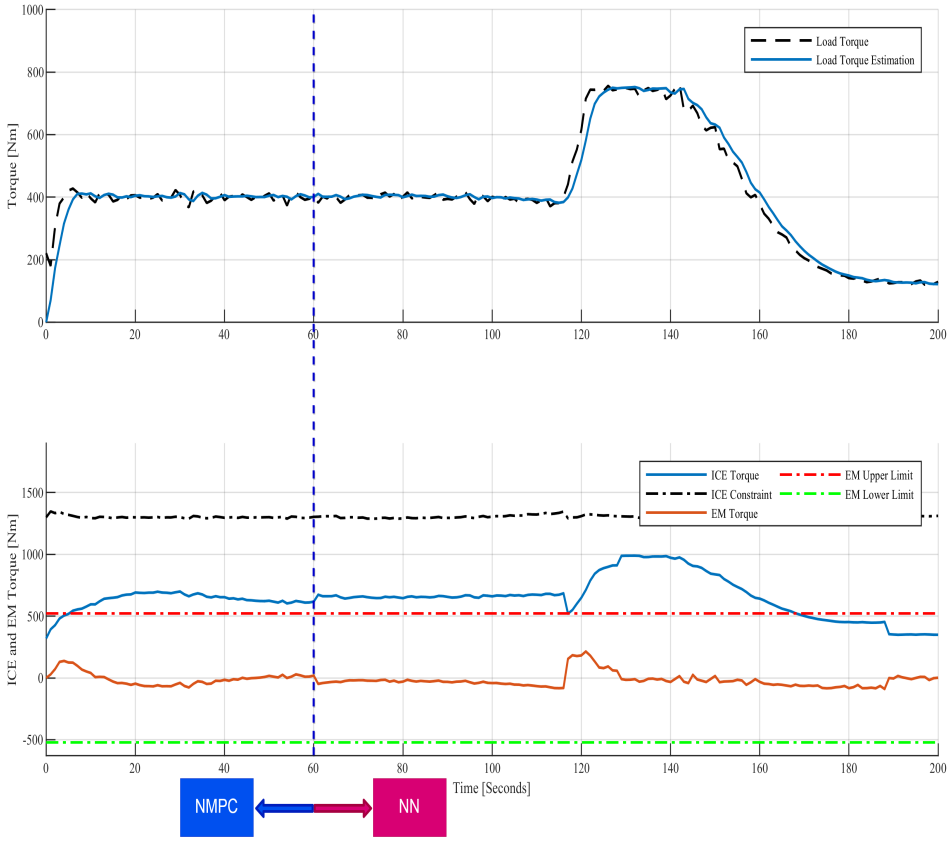


Figure 6.12: Diesel Engine and Electric Motor Torque Control during simulation

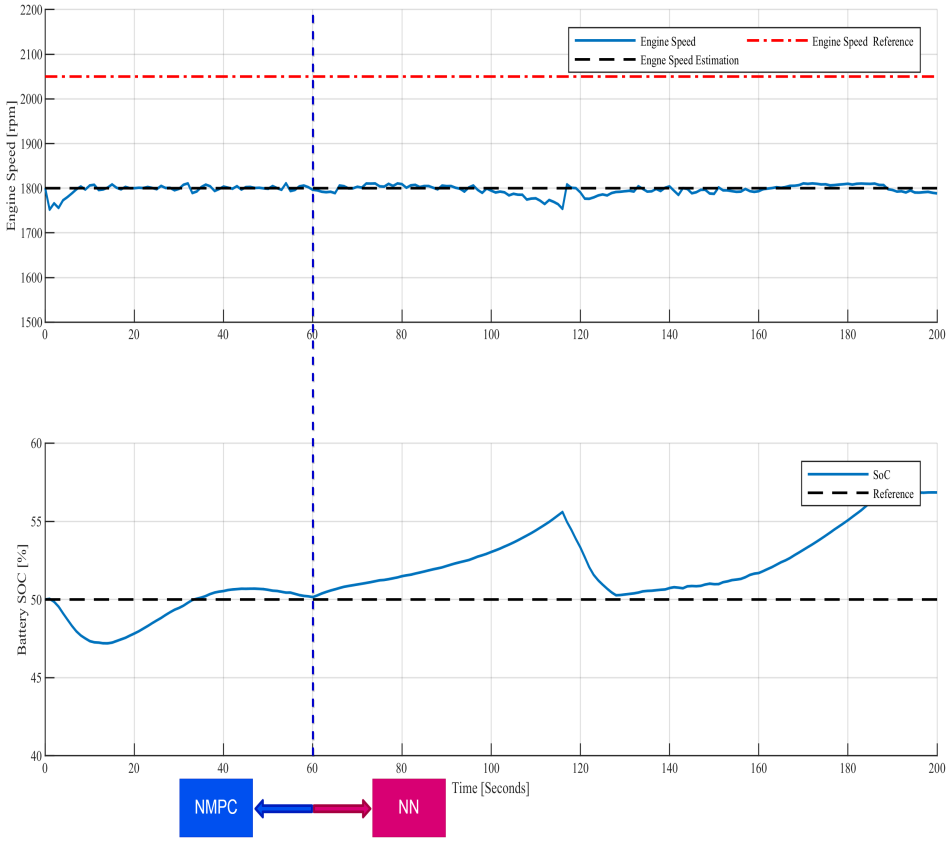


Figure 6.13: Engine Speed and Battery State of Charge during simulation

6.5 Marine Application: Step Acceleration Comparison with NMPC

Finally an all important comparison between the framework and the NMPC is presented for a series of step accelerations that expose the capabilities of the framework as a whole in a simulation environment.

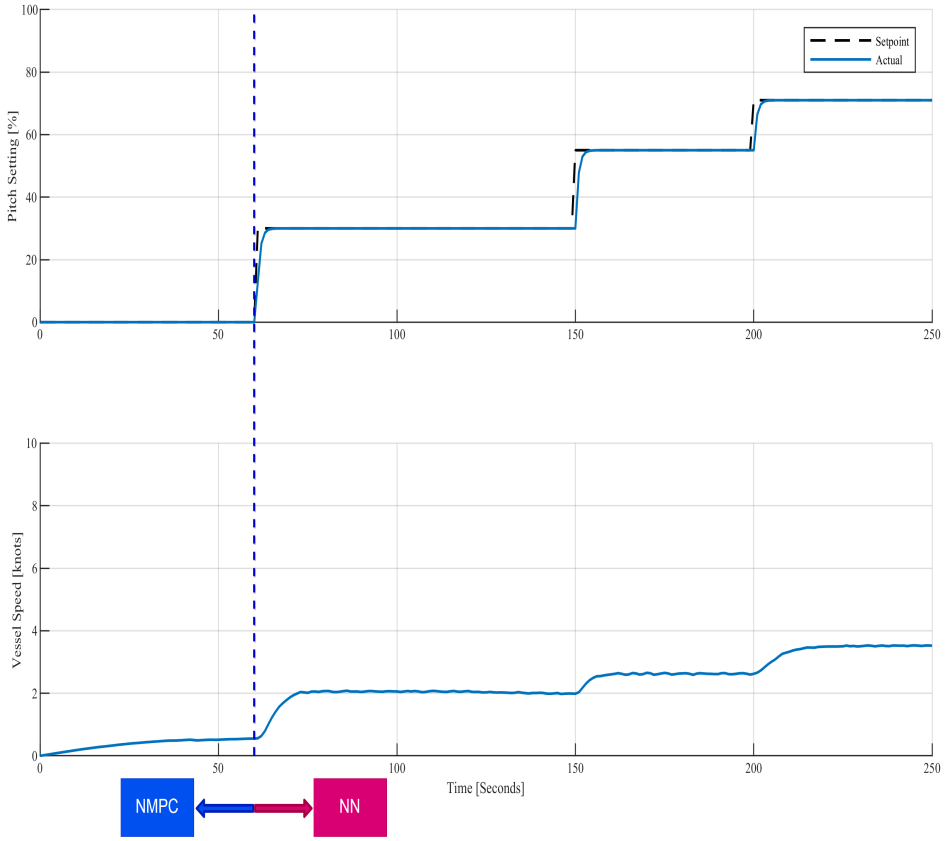


Figure 6.14: Pitch Angle Setting and developed Vessel Speed during step accelerations simulation

6.5.1 Simulation Results

In the following figures the simulation results for the step accelerations scenario are presenter, where the speed reference is constant and equals 1800 rpm. Fig., 6.16 are featuring the comparative outputs for the power split conducted by NMPC and NN-Power framework. Additionally, the results of the predictive phase neural networks are shown in Fig. **??**.

6.5.2 Simulation Analysis

Once again the prediction stage neural networks are accurately predicting the situation they are presented with. At 63 seconds a fast acceleration starts that makes NN-PAC predict cycle 5 and after some stabilization has happened it changes to cycle 3 which is expected as it stays for a long time in mid speed range. After the second acceleration it remains in the

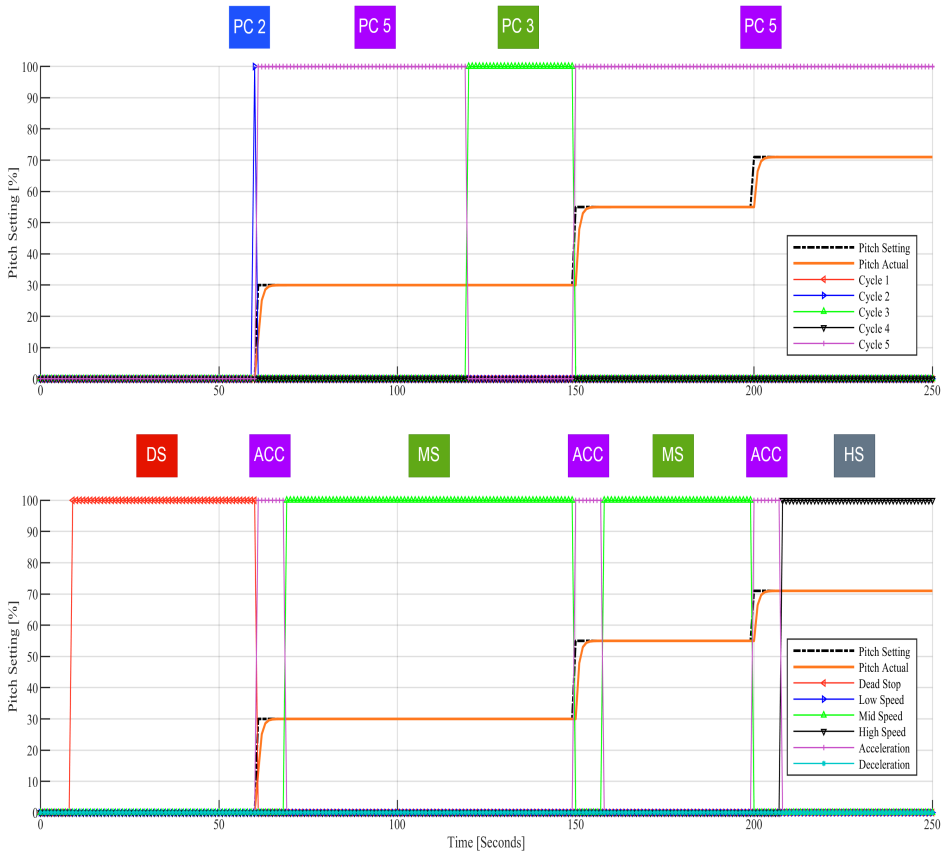


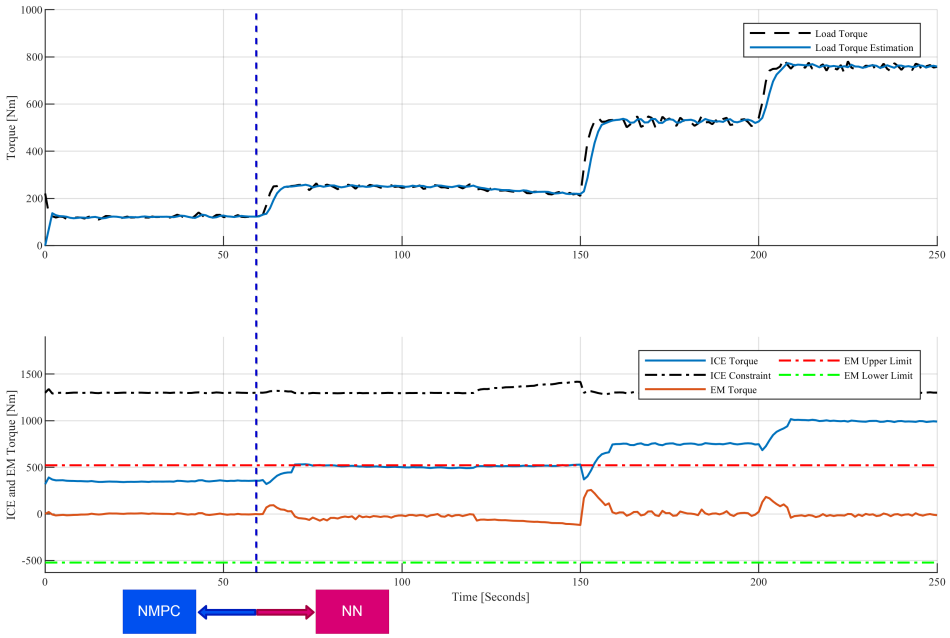
Figure 6.15: Pitch Angle and Cruising Trend Predictions during step accelerationssimulation

range of cycle 5 that we have established as the cycle capable of conducting these transients very well. For the NN-CRT the same issue with acceleration and deceleration occurs here once again. In general a very satisfactory outcome from the prediction networks.

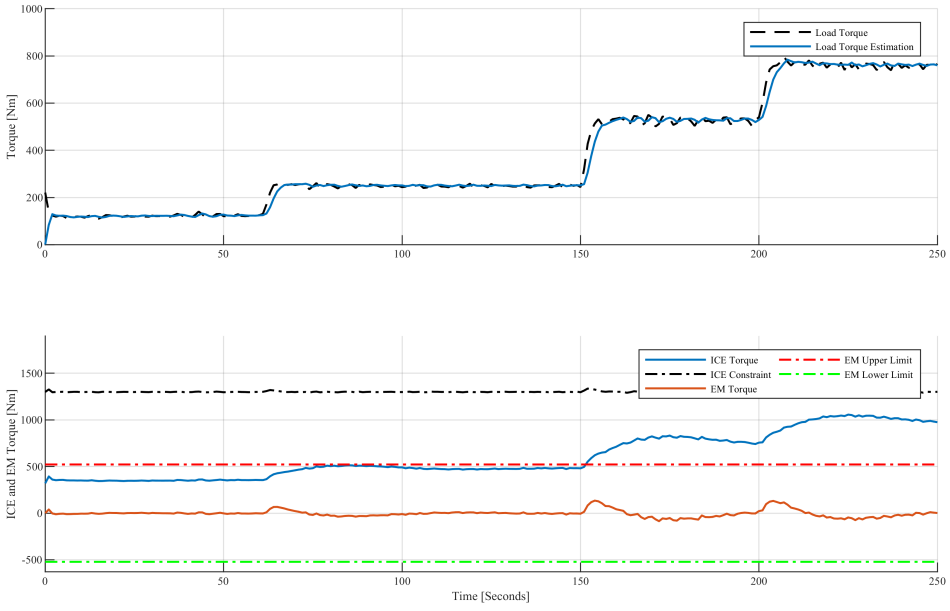
For the control stage neural networks as we can see the initial torque is handled by the EM in every acceleration and the ICE torque slowly rises once again, resembling a steady state condition. After the acceleration the motor is generating to recharge the battery back to 50% and the engine speeds up it's well inside the safe region.

Comparing now these results to the ones of the NMPC we can see that the framework is very closely following the commands of the controller. Nevertheless it is important to make a comment about the obvious deviations. For starters the framework appears to be pushing the motor harder during accelerations. For this specific case the reason why this happened is shown in the SOC diagram. The value of SOC at 150 seconds is at 57.2% where the largest value from training that the network of the 5th cycle has seen is 55.5%. The data that are feed to the network are normalized as we discussed earlier and this results in a value larger than 1 for the SOC feature. This in turn causes the motor to overdrive and subsequently self correct itself back in the accepted region of operation. It is important to note that indeed if this specific framework was left for a longer time at cycle 3 that would cause a problem as the value of SOC would be substantially larger than 1. This proves that although the framework performs very well inside the region that we studied, further development is needed to cover more cases as we will briefly discuss in the next chapter.

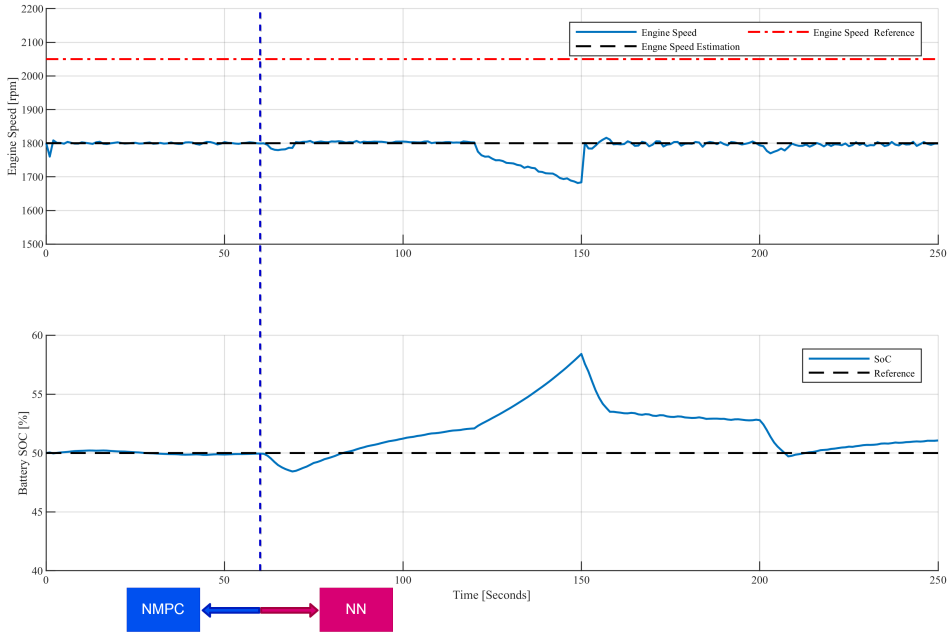
Nevertheless from the results we proved our proof of consent that a framework of neural networks such as LME-NNPower that we studied here can indeed emulate NMPC control for the effective power split of a hybrid marine propulsion plant.



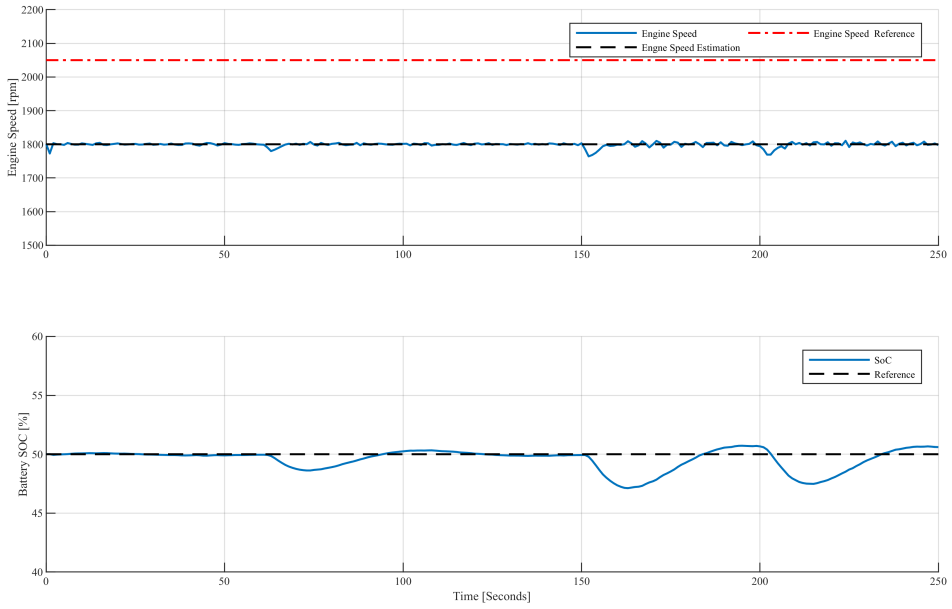
(a) The Diesel Engine Torque Command



(b) The Electric Motor Torque Command



(c) The Engine speed and Battery State of Charge conducted by the NN-POWER



(d) The Engine speed and Battery State of Charge conducted by the NMPC

Figure 6.16: Power split comparison between NMPC and Neural Network

Chapter 7

Conclusions and Recommendations

Conclusions

In this work we studied the development and usage of a machine learning framework based on artificial neural networks for the power split of a hybrid diesel-electric marine power plant. To achieve this we based our efforts on a Nonlinear Model Predictive Control approach that was previously studied.

The framework was divided in two major stages, the prediction stage and the control stage.

For the prediction stage, initially we shaped our available data in an attempt to standardise the operation profile of a tug vessel. This was done by means of knowing the pitch angle of it's CPP propeller blades in a time series fashion. Through this process we created 5 Standard Pitch Angle Cycles that we later used to categorise any operation profile in our range of study.

Afterwards we investigated the notion of cruising trend and it's role in the improvement of the overall performance of the framework. Through this process we created 6 classes of standard cruising trend types in order to be able to classify any trend pattern. With the presented results we concluded that the cruising trend indeed plays an important role towards better overall predictions in all situations.

For both aforementioned problems we employed a similar machine learning approach, i.e. a step-by-step overlapping time frame. In this method we chose a pair of time parameters, the time size of the moving window as well as the time step and according to them we extract useful features from the pitch angle operation profile of the vessel. A wide variety of feedforward neural networks were tested to find the best one for each problem. The selected neural networks of this prediction stage reached a high accuracy range of $88.56\% - 92.81\%$ for the cycle prediction for the case of a $60 - s$ window and 97.56% for the cruising trend prediction for the $9 - s$ window size instance.

For the control stage, a simulation of the NMPC for each cycle created the necessary data and 5 sets of power-split neural networks were developed. Every set was particularly trained for 1 of the standard pitch angle cycles. Again for this problem a wide range of neural network architectures was tested and the general MAE errors of the final networks remained in the acceptable range of less than $5Nm$.

Finally a variety of simulations were conducted in the environment of MATLAB Simulink where the results of the developed framework were convincing enough to conclude that with

further improvement, it's within its capabilities to conduct effective power-spit on a hybrid diesel-electric marine propulsion plant.

Future Work

In this thesis, a machine learning framework using neural networks was developed to emulate nonlinear model predictive control for a set of predefined Pitch Angle Cycles. The considered objective was for the networks to effectively learn and copy the control commands of the controller in order to reduce the engine dynamic response during transient operations via the electric motor.

For future work it can be advised to use real life data to create more operation profiles to expand the range of usability of the developed framework. Moreover we recommend the realization of more neural networks for the prediction stage of the framework as we believe that this will bring better results for the case of online control. Notable features that could be investigated can be the fuel mass consumption, air pressure and temperature. Furthermore features from the turbocharger or other auxiliary machinery can also be investigated for possible improvement.

By the same token, in this thesis the networks that we developed were feedforward neural networks. We strongly encourage the exploration of different neural networks in the supervised learning field, where a strong candidate would be a Long Short Term Memory (LSTM) network or the investigation of reinforcement learning algorithms like Q-learning neural networks that are lately blooming in the field of Machine Learning.

Also the developed GUI program can be improved to further automate the process of training by fully using the latest updates in the Machine Learning field.

Finally we advise the computation wise optimization of the framework in the simulation environment Simulink as well as the integration of an observation mechanism for the battery to take full advantage of the stored energy that is provides. Eventually when everything is in place, a series of real-time online control experiments must also be administered on the HIPPO-2 testbed to verify the consistency of the developed control scheme.

Bibliography

[1] D. Woodyard, *Pounder's Marine Diesel Engines and Gas Turbines 9th Edition*. Elsevier Butterworth Heinemann, 2009.

[2] R. Geertsma, R. Negenborn, K. Visser, and J. Hopman, "Design and control of hybrid power and propulsion systems for smart ships: A review of developments," *Applied Energy*, vol. 194, pp. 30 – 54, 2017.

[3] Y. L. Murphey, J. Park, Z. Chen, M. L. Kuang, M. A. Masrur, and A. M. Phillips, "Intelligent Hybrid Vehicle Power Control - Part I: Machine Learning of Optimal Vehicle Power," *IEEE Transaction of Vehicular Technology*, vol. 61, pp. 30 – 54, 2012.

[4] N. Planakis, G. Papalambrou, and N. Kyrtatos, "Predictive control for a marine hybrid diesel-electric plant during transient operation," pp. 989–994, 04 2018.

[5] H. T. Grimmelius, P. D. Vos, M. Krijgsman, and E. van Deursen, "Control of hybrid ship drive systems," 2011.

[6] V. Karystinos, *Nonlinear Model Predictive Control of a Hybrid Diesel-Electric Marine Propulsion Plant*. Zografou, Athens, Greece: NTUA, 2019.

[7] D. Varsamis, *Load Simulation during ship propulasion and application in experimental marine hybrid Diesel-Electric testbed*. Zografou, Athens, Greece: NTUA, 2019.

[8] S. O. Haykin, *Neural Networks and Learning Machines*. Hamilton, Ontario, Canada: Pearson Education, 2009.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*. Cambridge, UK: Springer, 2006.

[10] W. Waegeman, B. D. Baets, and L. Boullart, "Roc analysis in ordinal regression learning," *Pattern Recognition Letters*, vol. 29, pp. 1 – 9, 2008.

[11] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1 – 58, 1992.

[12] S. Raschka, *Python Machine Learning*. Lively Street 35, Birmingham, UK: Packt Publishing Limited, 2015.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[14] J. Park, Z. Chen, L. Kiliaris, M. L. Kuang, M. A. Masrur, A. M. Phillips, and Y. L. Murphey, "Intelligent Intelligent Vehicle Power Control Based on Machine Learning of Optimal Control Parameters and Prediction of Road Type and Traffic Congestion," *IEEE Transaction of Vehicular Technology*, vol. 58, 2009.

[15] D. Vekios, *Loading Cycle Generation of Marine Engines using Machine Learning Methods.* Zografou, Athens, Greece: NTUA, 2019.

[16] E. Giakoumis, *Diesel Engine Transient Operation.* 04 2009.

Appendices

Appendix A

Cassiopeia GUI

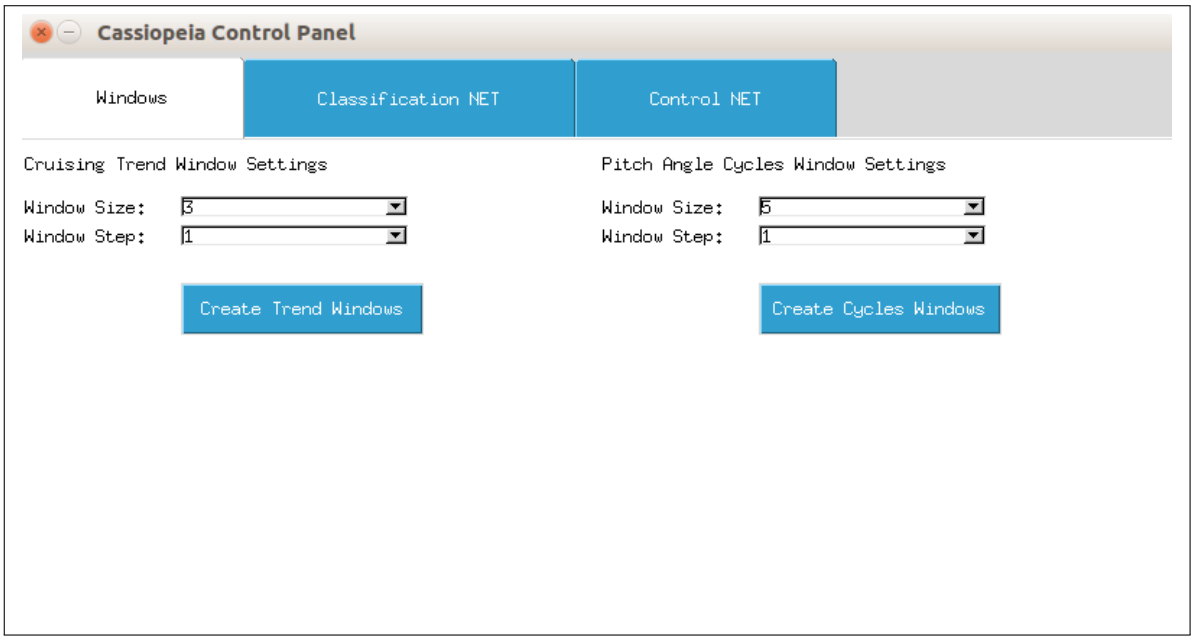
For the needs of this thesis a lite program with a graphical user interface was developed in order to fast forward the repetitive training process, as a lot of networks-models combinations have been tested. Below we present some pictures of the program. In order to make it we used the *Tkinter* module of the programming language *Python*.It has three main frames, each with a different main functionality.

The first frame is used to implement the automatic labelling algorithm so that we can categorise our data according to the problem we are trying to solve (Trend of Cycle Prediction). The user selects the pair or window size and time step that he wants to create data for and then presses the corresponding button. According to the problem that the user has chosen, the program automatically makes a parent folder with this name, i.e. Cycles or Trend. Afterwards the labelling algorithms is executed and the result is a folder with the name $model_size_step$ inside the aforementioned parent folder. Inside this folder there is a *train.csv* that we use for the training process of the networks that refer to this model.

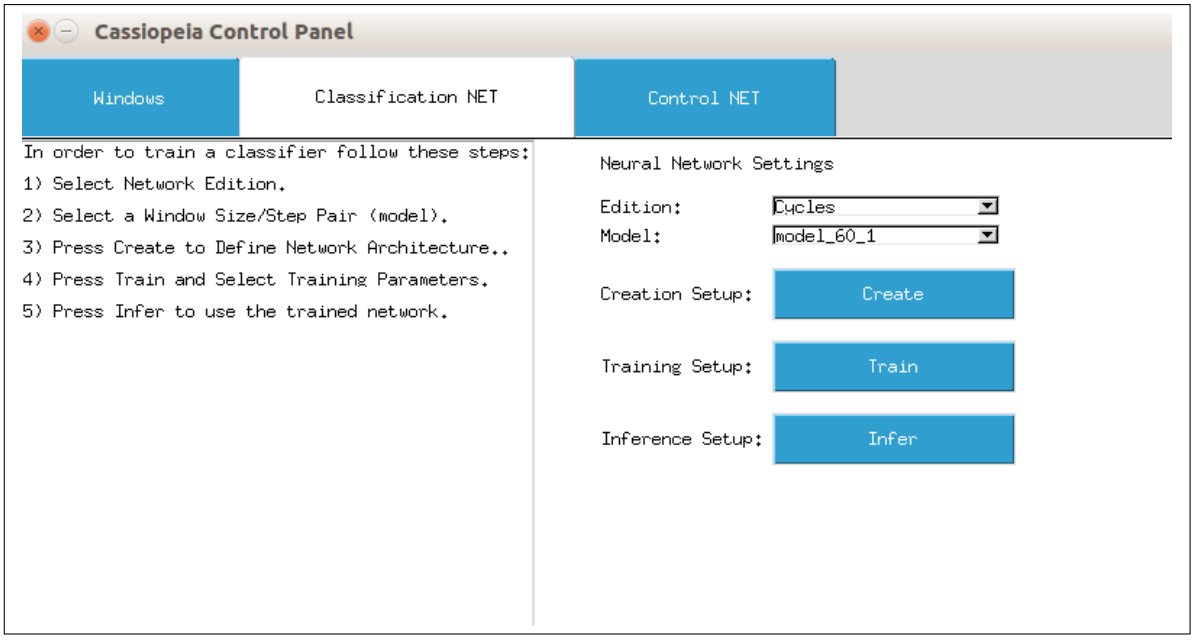
The second frame is the Classification Networks Frame. This frame is used to create a classification network, train it and later use it to infer on unseen data. The user selects the Edition of the network from Cycles or Trend. Afterwards the Model combo-box automatically will only show the existing models inside the Edition parent folder so that the user can only use valid data. To crate a network we press the *Create* button and then the Network Structure window pops up so we can choose the network architecture we want. The result of this process doesn't really create a network but instead makes a folder with a codenamed that hosts a *JSON* file. The program can later load this file when we want to use the network and this is done mainly for two reasons. Firstly to avoid unnecessary computation overhead, in order to keep the program as lightweight as possible and secondly to make multiple architectures possible for the same model. Indeed if we want to create multiple architectures we just change the layers and possibly the activation functions without restarting the program and the program will create a folder with a corresponding codename.

To train a network we press the *Train* button and the training setup window pops up. The first field is the network architecture that we want to train. Note that this pop up window will train a network for the problem we chose in the Edition field from Classification Networks Frame so this information is automatically picked up. Afterwards we chose the training parameters and click the train button. The main outcome from the training is a *Tensorflow* graph that can be used later to continue training if needed and a *MATLAB .mat* file containing all the weights and biases that can be loaded for simulation. The program also gives images of cost and accuracy from the training process.

To make inference with a network we press the *Infer* button and the inference setup window is made available. Here we select the network that we want to test on unseen data and by clicking the button the program outputs a *.csv* file and an image of the inference procedure.

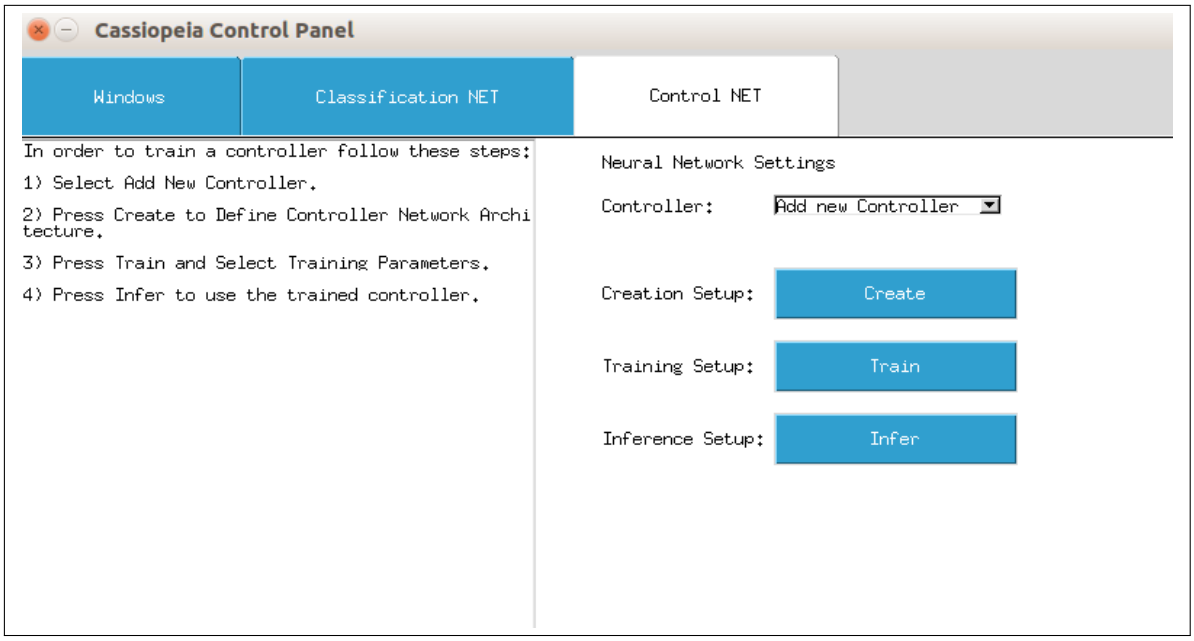


(a) The Automatic Labelling Process Frame

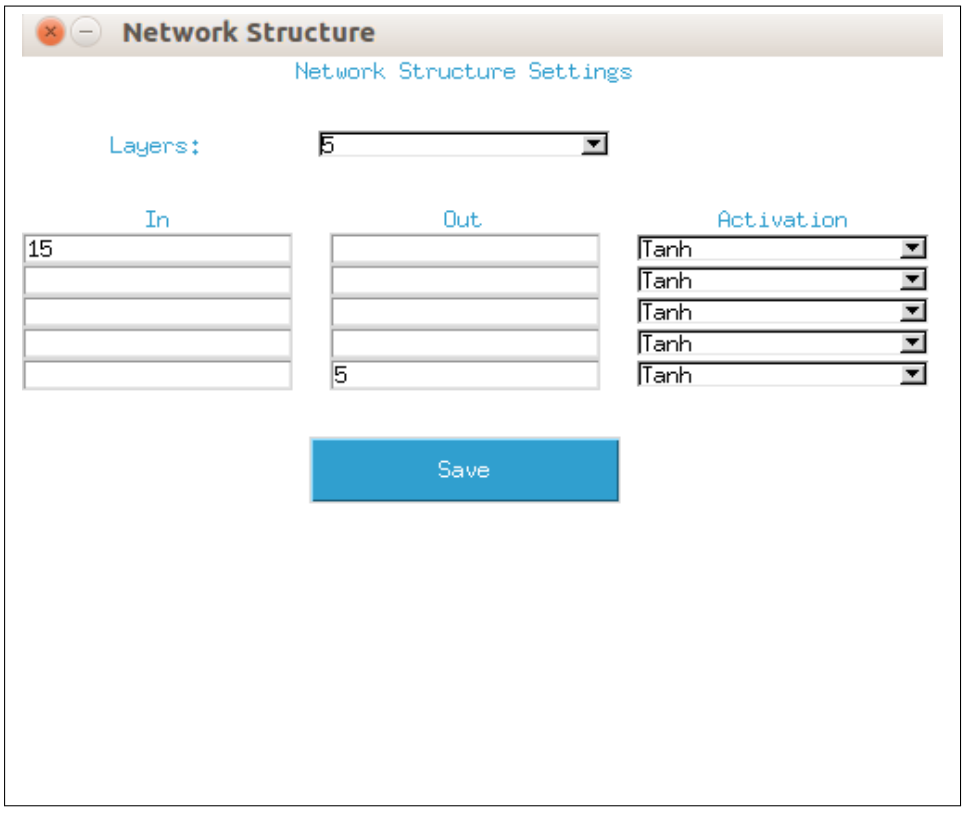


(b) The Classification Networks Frame

Figure A.1: The Cassiopeia GUI Program.

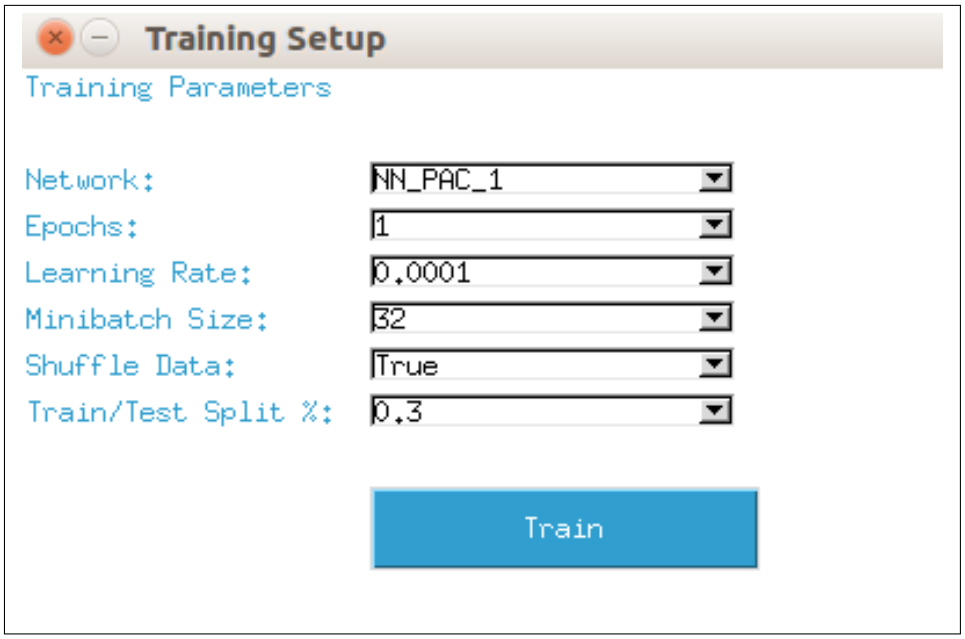


(c) The Frame of the Controllers

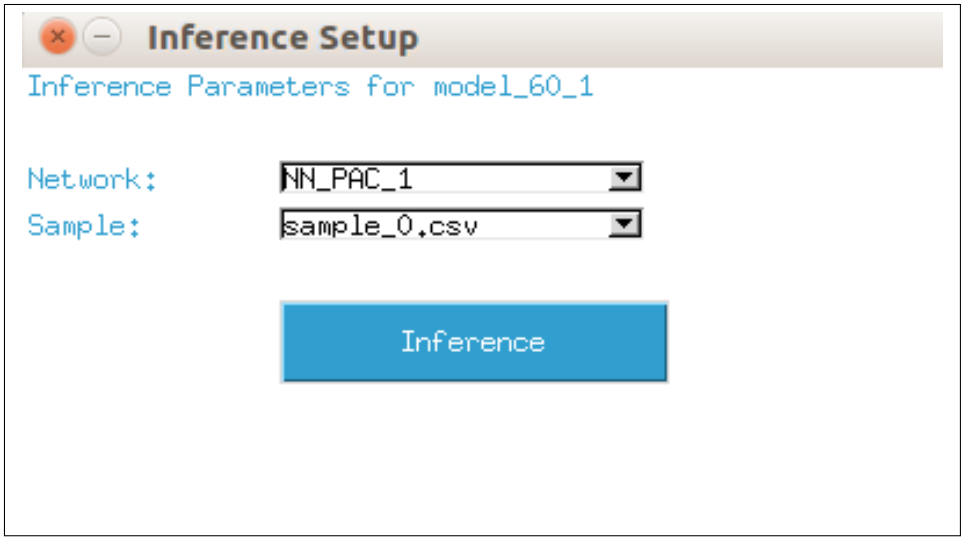


(d) The Networks Creation Process subwindow

Figure A.1: The Cassiopeia GUI Program (cont.)



(e) The Networks Training Process subwindow



(f) The Networks Inference Process subwindow

Figure A.1: The Cassiopeia GUI Program (cont.)