Identification and Control of Marine Vehicles using Artificial Intelligence Techniques

A novel approach

Submitted by: Pepijn Willebrord Justinus van de Ven

For the Award of Doctor of Philosophy

Supervised by:

Dr. C. Flanagan and Dr. D. Toal

Department of Electronic & Computer Engineering

University of Limerick

Submitted to the University of Limerick, November 2005.

I should like to dedicate this thesis to my parents, Helma and Ruud

Abstract

In this thesis a novel approach to the identification of marine craft dynamics using neural networks is described. From a literature review it emerged that augmented controllers, in which a conventional controller is augmented with a neural network, which accounts for unmodelled phenomena and/or unmodelled operation regions, are most likely to be used for future neural controller architectures. Such controllers are appealing, as neural networks can be used to identify the unknown phenomena with a high accuracy. However, at the current time, neural networks are predominantly used to identify unknown phenomena in a lumped way. As a result, it is difficult, or even impossible, to use these neural networks in a conventional controller. A novel approach, involving the use of several neural networks for the identification of individual model parameters, is presented. The new approach is tested, first in simulations and consecutively in an experiment, and found to offer increased accuracy compared to a benchmark least squares identification method. Additionally, it is demonstrated that the obtained model can easily be reformulated in order to be used in a control scheme. In this control scheme, the learning capabilities of neural networks and the robustness and guaranteed stability of more conventional control schemes, can be combined, thus obtaining the advantages of both approaches.

Acknowledgement

After 3 extremely pleasant years working towards a PhD, there are so many people I owe my gratitude. First of all I would like to thank my supervisors Dr. Colin Flanagan and Dr. Daniel Toal for all their support over the last three years. I greatly appreciate the freedom they have given me in choosing the research focus, yet keeping a close eye on my progress and giving the occasional push in the right direction. Their advice during the PhD and their careful proof reading of this thesis have considerably increased its quality.

I spent a total of 6 months in the Marine Technology Center of the Norwegian University of Science and Technology in Trondheim, Norway. As a result my supervisor 'panel' doubled in size. During, and in between my visits to Norway, Prof. Tor Arne Johansen and Prof. Asgeir Sørenson have helped me tremendously. I am highly grateful for their energetic and inspiring supervision. As a result of their efforts, this thesis contains more than a hint of Norwegian flavours. Takk skal dere i Trondheim ha for seks interessante og lærerike måneder!

Both in Ireland and in Norway several other people have had a significant influence on my PhD. Here I would like to thank these people that influenced me in a positive way. Sean Nolan, Levente Molnar, James Riordan, Jamie Hayes, Trevor Love and Edin Omerdic I would like to thank for their help and work on Tethra, the platform used for simulation studies in this thesis. Dino's Virtual Reality world has proved to be of special importance for these simulations. I would like to thank Sinead O'Keeffe for proof reading this thesis thoroughly. Her comments have, undoubtedly, rid this thesis of much of its double Dutch. In Norway I was fortunate enough to have mutual interests with Jon Refsnes and Andrew Ross. Jon's experiments with the Mines-

niper proved an interesting case study for my work, for which Jon willingly provided me with test data. Andrew spent several days with me in the Cyberlab, toying around with Cybership II and III. I still think I'm a better coxswain, Andy.

I would like to thank the Irish Research Council for Science, Engineering and Technology for funding this work. The EU Research Directorates contributed considerably through facilitating two Marie Curie fellowships in the Norwegian University of Technology with a total duration of 6 months. I would like to thank the Marine Institute for their much appreciated financial contributions towards conference costs.

Finally I would like to extend a well meant 'thank you' to all those people that I cannot mention personally for a lack of space. Over the last three years I have thoroughly enjoyed diving, playing volleyball and football, just having a laugh and the occasional pint with you all.

Abbreviations

AC	Augmented Control
ANFIS	Adaptive Network-based Fuzzy Inference System
ARNN	Autoregressive Neural Network
AUV	Autonomous Underwater Vehicle
BP	Back Propagation
BPTT	Back Propagation Through Time
CCL	Combined Control and Learning
CMAC	Cerebral Model Arithmetic Computer
CPM	Control Plant Model
DC	Direct Current
DOF	Degree(s) of Freedom
FALCON	Fuzzy Adaptive Learning Control Network
FBLC	Feedback Linearisation Control
FFW	Feedforward
IMU	Inertial Measurement Unit
LMS	Least Mean Square
LQR	Linear Quadratic Regulator
LS	Least Square
MLP	Multi Layer Perceptron
MPC	Model Predictive Control

MRAC	Model Reference Adaptive Control
NARMA	Nonlinear Autoregressive model with Moving Average
NARX	Nonlinear Autoregressive model with Exogeneous Outputs
NED	North East Down
NF	Neuro-Fuzzy
NN	Neural Network
OWA	Ordered Weighted Average
P	Proportional (controller)
PD	Proportional and Differential (controller)
PPM	Process Plant Model
RBF	Radial Basis Function
Rec.	Recurrent
RMS	Root Mean Square
RNF	Recurrent Neuro-Fuzzy
R NN	Recurrent Neural Network
ROV	Remotely Operated Vehicle
RTRL	Real Time Recurrent Learning
SANFIS	Self Adaptive Neuro-Fuzzy Inference System
SANFON	Self Adaptive Neuro-Fuzzy Optimisation Network
SCL	Separate Control and Learning
SISO	Single Input Single Output
SONCS	Self-Organising Neural-net Control System
SONFON	Self Organising Neuro-Fuzzy Optimisation Network
UUV	Unmanned Underwater Vehicle

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Underwater vehicles, manned and unmanned, are claiming an increasing interest in our society. This is not surprising as 70 % of our planet's surface is covered by water. With improving technology and increasing demand for new resources the unknown depths of the earth's seas and oceans become an ever more appealing target for exploration and mining.

Additionally, man's limitless curiosity makes for a great drive into marine research. After all, our existence stems from below the surface. Additionally, evidence of the rich history of the world's sea fairing nations should often not be looked for in a museum. Due to the harsh environment presented by the sea and the world's weather systems, many a ship never made it to its final destination, leaving an abundance of historical treasures relatively well conserved by salt water, some of which have yet to be discovered.

As a result of these, and other factors, current interest in underwater vehicles can be found in industry, academia, law enforcement and even the recreational sector. Particularly important is that the underwater world is a very hostile environment for human beings. It is thus evident that sending an underwater vehicle, and if possible an unmanned underwater vehicle (UUV), should be preferred to sending a diver on a specific task. It may be a coincidence, but on page 33 of

the March/April 2005's issue of the 'Underwater Contractor International' the two headlines are: 'Too many deaths for diving industry', and 'Notable increase in ROV industry'. In the same issue of this magazine, the loss of the *Autosub*, owned by the UK Natural Environment Research Council, is reported. During its last mission under the Antarctic ice shelf it became trapped and is considered to be irretrievable. Loss of this stg£1.5 million asset illustrates the importance of increased robustness and intelligence for autonomous underwater vehicles.

One of the earliest remotely operated vehicles (ROV) is thought to be a vehicle designed by Giovanni Lupis (Myers, 2005) a Captain in the Austrian navy[1] in the 1860s. This craft, controlled by steering cords from a ship, was designed to carry explosives to an enemy ship. A later version of this craft was equipped with one of the first gyroscopes, thus becoming the first torpedo.

In the 1960s the United States Navy became interested in ROVs for several dangerous underwater tasks. This lead to the development of several ROVs. One of the more successful systems, the CURVE (Cable controlled Underwater Recovery VEhicle), was used for the retrieval of a hydrogen bomb lost off the coast of Spain, near Palomares, in 1966.

In recent years the development of autonomous underwater vehicles (AUV) has attracted increasing interest. See Nolan (2003) for an overview of past and current projects worldwide. Especially for AUVs, but also for ROVs, in order to minimise operator intervention, robust and adaptive low level controllers are important. A low level controller, in this thesis, is defined as a controller that controls the position or velocity of the craft. Higher level controllers, in order of increasing level, could be responsible for point-to-point navigation, trajectory planning or task planning.

In this thesis only low level control of underwater craft will be investigated. State of the art control in this field focusses on the application of linear and non-linear feedback controllers.

[1] Paradoxical as this may sound for a country currently without shoreline, Austria's territory, until the first world war, consisted of a substantial shoreline on the Adriatic sea borders.

Fossen (2002) gives a comprehensive overview of feedback control applied to marine vehicles. Both linear and non-linear feedback controllers introduce certain problems. Linear control theory is well understood and since the 1920's a comprehensive theory has been formulated. However, as most underwater vehicles show non-linear behaviour, linear control cannot be used for all circumstances. Generally, controllers for a limited operating regime are the result. An answer to this problem is the application of non-linear controllers. The theory on these controllers is, unfortunately, far from complete. As a result, non-linear control theory can only be applied under assumptions which again limit the operating range of the controller. Additionally the underwater world presents us with a highly dynamic environment. Due to ocean currents, waves and the relatively high probability of malfunctioning in the propulsion system (e.g. sea weed or chipped propeller blades) a control system will have to be able to work over a wide range of operating points.

A possible solution to these problems is offered by the field of artificial intelligence. Important characteristics of artificial intelligence are its learning capabilities, flexibility and aptitude for dealing with non-linear problems. Good examples of artificial intelligence techniques with these characteristics are artificial neural networks, fuzzy logic and neuro-fuzzy systems. Because of the mentioned characteristics, these artificial intelligence techniques may be suitable candidates to tackle the above mentioned problems.

1.2 Research Objectives

In this thesis the identification and, to a lesser extent, control of a(n) (under-)water craft's velocity using artificial intelligence techniques is investigated. As it was found that a great advantage can be obtained by identifying a forward model of the craft dynamics with artificial neural networks, and consecutively using this forward model for control, the emphasis in this thesis is on such identification.

The use of artificial neural networks for identification is fairly widespread. All through the identification literature, both technical and non-technical, examples of identification using artificial neural networks can be found (Cheron et al., 1996; Gillard and Bollinger, 1996; Matthews and Moschytz, 1994; Boroushaki et al., 2003; W.G. Vieira and Fileti, 2005; A. Jovicevic Bekic and Jovicevic, 2004; Morabito and Versaci, 2003; Pai and Lin, 2005). However, in most cases, the trained artificial neural networks can only be used as a forward model (unless an inverse model was explicitly identified). As a result these artificial neural networks are less suitable for control purposes, other than in control schemes in which a forward model of the plant is explicitly used, for example model reference control.

In this thesis a modelling approach yielding the possibility of a versatile use of the resulting artificial neural networks is pursued. In this approach the structure of a representative model is assumed to be known. The neural networks are used to identify and, if necessary, represent the various parameters in the model. As detailed in chapter 4, the parameters identified by the neural networks can be both constant as well as being functions of time varying quantities, such as the process states.

1.3 Overview of the thesis

Chapter 2 gives a short review of the artificial intelligence techniques used in this thesis, and their application to model identification. Notably artificial neural networks, fuzzy logic and neuro-fuzzy networks are discussed. For the interested reader a derivation of the back propagation algorithm is provided in appendix A.

A review of the application of artificial intelligence to the identification and control of autonomous underwater vehicles is presented in chapter 3. First an overview of models used in the simulation studies, found in the literature is given. This overview is based on a generic model proposed by Fossen (1994), which is briefly covered. The review presents a comprehensive overview of identification of underwater vehicle dynamics using artificial intelligence techniques from the early 1990s (when artificial neural networks were first applied to identification problems) to the present. Additionally, the strategies found in the literature, are classified according to a newly proposed framework. The benefit of this framework is that it enables a clear analysis of the advantages and disadvantages of the approaches found. This analysis, in turn, leads to conclusions regarding the merits of each class and is used to decide on the direction for further research.

Based on the conclusions from chapter 3 it was decided that a forward model, rather than an inverse model, should be identified in such a way that it can be used (*i*) in parallel to conventional control schemes or (*ii*) as part of a conventional control schemes or (*iii*) fully separate from the latter. In order to facilitate the above demands, in chapter 4, a novel approach is presented, in which artificial neural networks model certain parameters in a known model structure. As, after the identification has been performed, both the model structure and approximations of the previously unknown parameters are known, the results can now be used in any of the three

approaches mentioned above.

The newly proposed identification method is adopted in a simulation study in chapter 5. The simulation study is based on simulations with the University of Limerick in-house built underwater vehicle Tethra (Molnar et al., 2004). With data gathered during manual control in an environment created with the Matlab Virtual Reality toolbox, an identification is performed and the model obtained is verified with a feedforward controller based on the identified parameters. No feedback controller was used to clearly show the quality of the identified parameters; it will be shown that for noise free signals, the parameters can be identified accurately. For noisy signals the identification is acceptable when online learning is applied to constantly update the artificial neural networks.

In chapter 6 the newly proposed identification method is compared to a standard off line least squares algorithm. Simulations show that, with noisy measurements, the newly proposed identification system is superior to the least squares algorithm, although this is achieved at the cost of a considerably higher computational load.

Testing of the proposed algorithm on real data is discussed in chapter 7. A three month period was spent in CyberMar, the Marie Curie research centre in the 'Marine Technology Center' of the Norwegian University of Science and Technology in Trondheim, Norway. With measurement data the damping of the Minesniper, a newly developed defence system, manufactured by Kongsberg, is identified. The results show a considerably improved identification, compared with previous models for the damping.

Finally, in chapter 8 the work presented in this thesis, is briefly summarised. Then overall conclusions and suggestions for future work are given.

Chapter 2

A review of artificial intelligence techniques

In this chapter the use of neural networks for control is discussed. The term neural network originates from the efforts of neuroscientists to model the human brain using relatively simple models. These models consist of neurons that interact in some way to perform certain mappings from an input layer to an output layer. In this thesis there is no intention to try to model or replicate the functioning of the brain. As thus no confusion between artificial neural networks and biological neural networks is possible, the adjective artificial will be omitted from this point on.

In section 2.1, the most important network architectures and training algorithms are introduced. Then, in section 2.2, the application of neural networks to identification and control is discussed. Finally, in section 2.3, conclusions are drawn and the choice of the type of neural networks to be used in this work is explained.

2.1 Neural networks

In this section a concise overview of the most important neural networks used for identification and control is given. Several networks using supervised learning, such as back propagation neural networks, radial basis function networks and recurrent neural networks will be discussed briefly. Additionally the combination of fuzzy logic and neural networks, neuro-fuzzy networks, will be discussed.

2.1.1 Feedforward neural networks

Within the field of neural networks there exist a multitude of different networks all meant to learn from presented data. We can divide the learning strategies into two groups: supervised learning and unsupervised learning / reinforcement learning. Supervised learning is based on presenting the network with input data and output data. The network can thus make changes to its free parameters such as to minimise a given loss function over the set of given examples. In unsupervised learning these input-output examples are not available. Rather, the network learns to classify the input data based on certain (statistical) characteristics present in the data. A scalar loss function or other measure of performance may be used as some kind of feedback. In the latter case one speaks of reinforcement learning.

Unsupervised learning and reinforcement learning are appealing from one point of view as they may resemble the way a human brain learns. However in practice it is often the case that input-output data of high quality is available *a priori*, and that the desired end of training is to achieve a mapping of high quality or accuracy. In such a case, supervised training is preferred. In this thesis only networks using supervised learning are used and discussed as the neural networks are used for the mappings mentioned, and as input-output relationships can be obtained through

experiments.

2.1.2 Neural networks using back propagation

Shifting the focus to the group of neural networks using supervised learning, the best known neural network is the feedforward multi-layer neural network trained with the back propagation algorithm. As this network is an extension of Rosenblatt's perceptron (Rosenblatt, 1962) including several layers of neurons, it is often called a multi layer perceptron. In the following the abbreviation MLP will be used to denote these networks.

In a MLP data is processed in a forward direction from input to output. On its way from input to output the data passes through one or more layers of neurons. In each neuron input data from the previous layer is collected. A certain, often non-linear, function is applied to the accumulated inputs and the result of this computation is sent to neurons of the next layer. The connections between all these neurons are thus only from a previous layer to the next layer. No connections within one layer or to previous layers are present. By changing the strength, or weights, of the connections between the neurons, the MLP can learn to perform the desired mapping. This is done by first processing the data through the network from input to output. At the output layer the network output is evaluated and compared to the desired output. Based on the difference between desired and actual output, the layer weights are changed.

The simplest and most straightforward way of changing the weights is based on the gradient of the error function with respect to the weights. At each point in time the gradient will point in the direction of steepest ascent on the error surface. Changing the weights such that a path in the opposite direction is followed would thus decrease the error. This is the basis of the back propagation algorithm of which a derivation is presented in appendix A.

During the last twenty years the back propagation algorithm has proved to be the most popular

training algorithm. A multitude of variations on back propagation have been suggested and are being used for the training of MLPs. Appealing features of MLPs are (Nelles, 2001):

- Interpolation tends to be monotonic
- The accuracy of the mapping is very high.
- Smoothness of the mapping is very high.
- Sensitivity to noise is very low.

However, of course some drawbacks exist:

- Parameter optimisation is performed using non-linear optimisation techniques and is thus slow.
- As a result of using non-linear optimisation techniques, finding a global optimum cannot be guaranteed.
- Interpretation of the free parameters is hardly possible.
- Incorporation of prior knowledge is thus almost impossible.

2.1.3 Radial basis function networks

As in MLPs, radial basis function networks (RBFs) consist of an input layer and output layer and a hidden layer in between. They can be looked upon as networks performing a curve fitting in a high dimensional space[2]. In this high dimensional hidden space the neurons form a basis with which the input vector can be represented.

[2]RBF networks can also be seen as support vector machines with Gaussian kernels (Haykin, 1999)[p. 333]

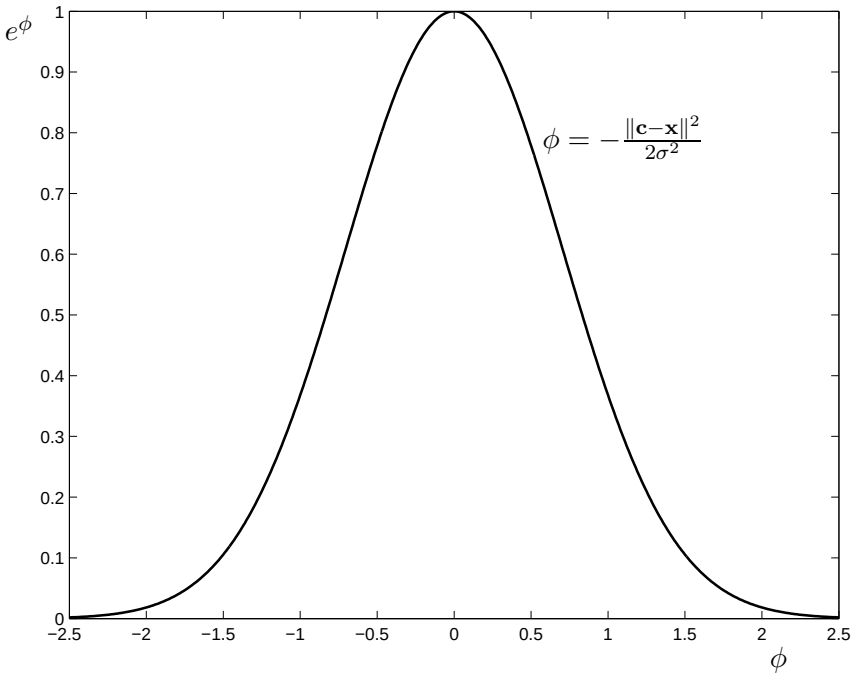


Figure 2.1: Gaussian radial basis function

Figure 2.1 shows a hidden layer RBF neuron for a one-dimensional example (i.e. the input to the RBF network is a scalar). The RBF is chosen to be of the Gaussian form:

$$G = e^{-\frac{\|\mathbf{c}-\mathbf{x}\|^2}{2\sigma^2}}. \tag{2.1}$$

At the centre of the cloche, the RBF reaches its maximum value of 1. Here, the value of the input vector $\mathbf{x}$ is identical to the centre of the RBF neuron, $\mathbf{c}$. From equation 2.1 it can be seen that the value of the RBF will decrease towards zero for deviating inputs to either side. For n inputs the RBF neuron will calculate the Euclidian distance between its centre and the n-dimensional input (the 2-norm) and apply the function from figure 2.1 to the result. To change the sensitivity of the RBF neuron, the standard deviation, σ, can be changed. In some training

algorithms this is done according to the spread of the centres of the RBF neurons:

$$\sigma = \frac{d_{max}}{\sqrt{2M}}, \tag{2.2}$$

where d_{max} represents the maximum distance, or spread, between all centres of the RBF neurons and M the amount of neurons in the RBF network. A high spread will yield a relatively high influence of all training vectors. A low spread will result in the network effectively only yielding an output for those input vectors that are very close to the centres of the RBF neurons.

In this work a relatively simple algorithm for training the RBF network is chosen. Initially the network consists of zero neurons. Then, for every iteration, one neuron is added to the network. The centre of this neuron is chosen to coincide with the training vector that yields the highest error in output prediction. Before commencing, a fixed standard deviation is chosen ($\sigma = \sqrt{5}$).

As the size of the hidden layer is a function of the amount of input training vectors that need to be represented accurately, it is often of considerable size. As can be seen from figure 2.1 it performs a nonlinear transformation on the input data. The reason for this is that a pattern classification problem cast in a high dimensional space is more likely to be linearly separable than in a low dimensional space (Haykin, 1999). An important difference with MLPs, is that RBFs, although a universal approximator, are local approximators whereas MLPs construct global approximations. This is due to the fact that the radial basis functions have a local character. In response to receiving an input, the neurons in the radial basis function network calculate the Euclidian distance between the input vector and the centre of each neuron. If this distance is small enough the input vector falls within the receptive field of that neuron and a significant output will result. However, for input vectors further away from the centre of a specific neuron, the influence on this neuron, and thus its output will be negligible. The neuron therefore influences the input

data locally. This is in contrast to MLPs in which each neuron in a hidden layer computes the inner product of the input vector, hence influencing all input vectors.

Appealing properties of RBF networks are:

- The sensitivity to noise is low as normally basis functions are chosen to operate in regions of high data density (Poggio and Girosi, 1990).
- Interpretation of the basis function parameters is possible due to their local characteristics.
- Incorporation of prior knowledge is possible (Kadirkamanathan et al., 1995; Niyogi et al., 1998).

Reasons for not choosing radial basis function networks for the identification of model parameters are:

- Accuracy of the RBF networks is typically lower than the accuracy of a MLP for the same amount of neurons (Haykin, 1999).
- The 'curse of dimensionality' is medium to very high. Due to the RBF network being a local approximator an increase in dimensionality will generally necessitate an increase in hidden nodes (Nelles, 2001).

2.1.4 Recurrent neural networks

The two previous neural networks, feedforward MLPs, so far simply called MLPs, and RBFs have one important thing in common: information is always processed from input to output. While processing input data, there is no data flow to previous layers in the network. In recurrent neural networks this information flow to previous layers is deliberately made possible in order

to give the neural network a 'notion of time'. By allowing 'old' information to reenter the net, states can be created in the net describing time dependent processes. Hence a recurrent multilayer perceptron[3] can be used to model time dependent phenomena. Although this can be a great advantage over MLPs, there are drawbacks. Most notably the training of recurrent MLPs is computationally a highly complex task. Due to the fact that the network is ideally updated to optimise all past states, the recurrent MLP will have to be 'unfolded in time'. This can be done using a learning algorithm called back propagation through time (BPTT). BPTT (Rumelhart et al., 1986)[Ch. 8] comprises of making N copies of the MLP in series for all N data samples used for training. A MLP with $N \cdot L$ layers, where L is the amount of layers of the recurrent MLP, is thus obtained. For a dataset of several thousand data points this leads to an excruciatingly high computational burden. To lessen this burden the following compromise can be made. Rather than using all N data points in the computation of the network updates, only the last K are used where $1 < K < N$ (Williams and Peng, 1990). For $K = 1$ the update algorithm reduces to standard back propagation. In reducing the depth of the algorithm, the accuracy of the calculated derivatives suffers. However, in general the influence of past derivatives is smaller, and at a certain point in time negligible. Truncation is thus acceptable.

To overcome the shortcomings of BPTT, an algorithm called real time recurrent learning (RTRL) can be used (Williams and Zipser, 1989). In RTRL a recursive approximation of the derivatives at time T is formulated based on all previous derivatives. In doing so RTRL is considerably less computationally expensive than BPTT. However, compared to normal back propagation it still is highly computationally demanding. As a result the application of recurrent MLPs can only be justified if there is a specific need for recurrent connections, *i.e.* if time dependent states need to be represented in the MLP.

[3]The term MLP will be used for feedforward multilayer perceptrons. If recurrent connections are used this will be mentioned explicitly.

2.1.5 Neuro-fuzzy networks

In neuro-fuzzy networks, two artificial intelligence techniques with different characteristics and advantages are combined. The structure of neural networks is used for their learning capability and distributed processing whereas the fuzzy logic input is a higher level If-Then structure which enables the use of *a priori* (expert) knowledge to be used in the networks. Moreover after training, the If-Then structure enables extraction of learned knowledge.

In fuzzy logic, knowledge is represented using membership functions and If-Then rules. The membership functions determine if and to what extent an input belongs to a certain class. The If-Then rules are then used to obtain responses given the class information. The responses of all If-Then rules are then collected and the appropriate output is determined.

The combination of neural networks and fuzzy logic can roughly be divided in two groups. Using neural networks in a fuzzy logic network setting yields neural fuzzy systems and using fuzzy logic in a neural network yields a fuzzy neural system (Lin and Lee, 1996).

Neural fuzzy systems comprise all fuzzy systems that apply neural networks to perform one or more of the actions in a fuzzy logic system. A neural network can be used to represent a membership function. Using sigmoidal neurons, an arbitrary real membership function can be represented by the neural network with the advantage that this membership function can be obtained through normal neural network training. Neural networks can also be used to perform fuzzy operations. In Yager (1992) the ‘ordered weighted average (OWA) neuron’ is proposed with which operations such as a fuzzy AND or a fuzzy OR can be implemented. Fuzzy inference can be improved by the introduction of neural networks (Keller and Tahani, 1992) as the neural network’s ability of parallel computing can speed up the inference of fuzzy rules.

Fuzzy neural systems are all those neural networks of which a part has been ‘fuzzified’.

Examples of parts that can be fuzzified are: the neuron, its weights and input and output data (Lin and Lee, 1996). Examples of fuzzified neurons can be found in Gupta and Qi (1991); Hayashi et al. (1992). Additionally the training algorithm can be fuzzified as shown in Keller and Hunt (1985). In Keller and Hunt (1985) the importance of a certain input-output pair for the weight update is controlled using fuzzy membership functions. In this way insignificant data pairs, that are possibly harmful for correct classification, can be ignored in the weight updates.

2.2 Neural identification and control

In the previous sections artificial intelligence techniques used for identification and control purposes and their learning algorithms were discussed. In this section the architectures encountered in identification using neural networks and neural control (or neurocontrol) will be discussed.

2.2.1 Identification

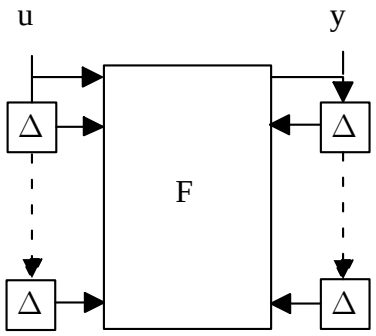


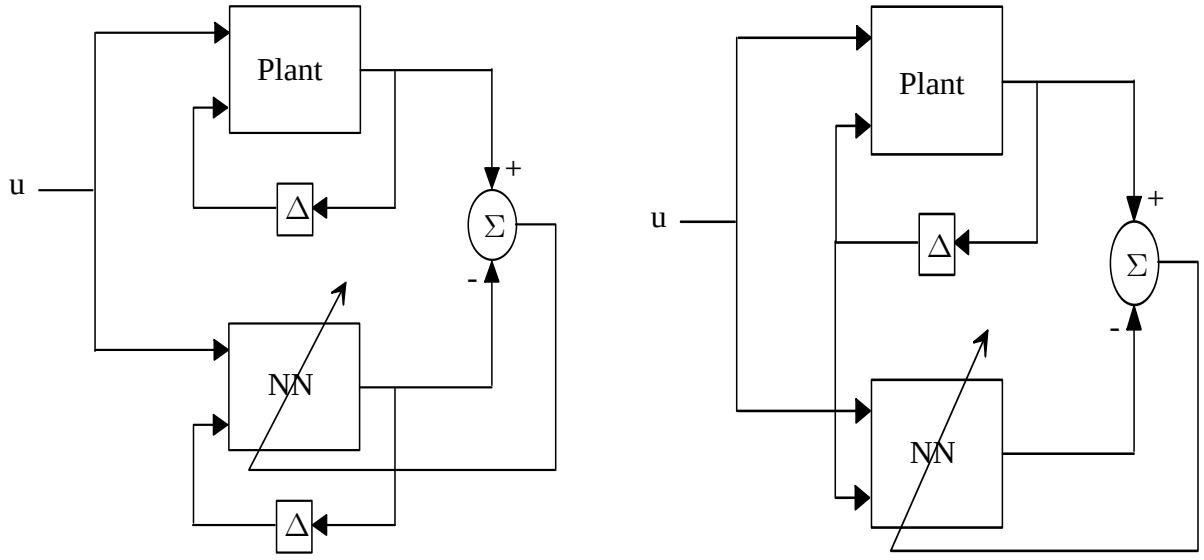
Figure 2.2: Non-linear Autoregressive model with eXogeneous inputs (NARX)

An appropriate identification model for most processes encountered in control problems is the model depicted in figure 2.2 which was presented in Narendra (1996) for neurocontrol purposes. In Narendra (1996) this model is called a NARMA model. However, according to the termi-

nology in current literature on model identification (established by Ljung (1987)), the model is considered to be a NARX model (non linear autoregressive model with exogeneous inputs). In this thesis the terminology introduced by Ljung (1987) will be followed. As can be seen from figure 2.2 the NARX model computes the new output based on past values of both the input and the output. Mathematically the model can be described as:

$$y(k+1) = \mathrm{F}[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-n+1)]. \quad (2.3)$$

Use of neural networks to represent the NARX model is extremely interesting as neural networks can exactly do what is suggested in equation 2.3, namely to approximate an unknown function of past inputs and outputs. The only assumptions that have to be made are that the function to be approximated is real and continuous.



(a) Parallel model

(b) Series-parallel model

Figure 2.3: Training configurations for forward model identification

The neural model can be trained in two ways, as shown in figure 2.3. In figure 2.3a the neural model is running in parallel to the real plant and only receives the real inputs. The new system output is calculated based on this current input and on estimates of the output made in the past. As for an untrained neural network the latter estimates will normally be unrealistic, the neural network in effect learns the wrong mapping. As a result, convergence to the correct mapping may be slow. More appropriate, and therefore usually applied, is the approach in figure 2.3b. In this case the neural network learns to make a mapping from the real process inputs *and* the real process outputs to the new process output. In both cases back propagation algorithms can be used for training as both the input and the output data is available.

Neuro-fuzzy modelling can be applied to an unknown non-linear plant as described by Horikawa et al. (1992). Fuzzy logic is capable of describing non-linear plants linguistically. However, finding the right rules and tuning those rules is cumbersome. Horikawa et al. (1992) proposed a method making use of neural networks to represent the fuzzy rules while the neural network weights function as the parameters in these rules. In this system the process of finding the correct fuzzy rules and the tuning of these rules is done through the application of the back propagation algorithm, thus alleviating the difficulties encountered in finding and tuning the rules.

2.2.2 Neurocontrol

Controllers making use of neural networks will now briefly be discussed. In chapter 3 a more extensive overview of the use of neurocontrol (applied to underwater vehicles) will be given.

A neurocontroller is defined as any controller that uses a neural network somewhere in the control loop. The application of the neural network is normally either as a model of the process to be controlled or as the controller itself, possibly in parallel to a conventional controller. The main reasons for applying a neurocontroller are twofold (Narendra, 1997):

1. A (complete) mathematical model of the process is not available. The available input-output data may be too noisy to be used on linear identification models or the latter are simply incapable of representing the dynamics of the process.

2. The process can be controlled satisfactorily under nominal conditions, but a controller is sought that can increase the robustness of the control scheme for changing circumstances.

Neurocontrollers can offer a solution for both mentioned problems because of their ability to learn the required representation using input and output data. As in this learning process mostly nonlinear algorithms are used, and the theoretical tools for the analysis of these nonlinear algorithms are still immature, use of neurocontrollers is currently more an art than a science. This phenomenon is demonstrated by the difficulties an inexperienced user will encounter when having to choose the value of learning parameters, the number of hidden layers and the number of nodes per hidden layer.

A special class of neuro-controllers consists of the neuro-fuzzy controllers. These controllers use a mix of fuzzy logic and neural networks. An example of such a neuro-fuzzy controller is the Fuzzy Adaptive Learning Control Network (FALCON) proposed by Lin and Lee (1991). FALCON is a neural network with fuzzy functions performed in different layers of the network. The inputs of the network are processed by a layer implementing the membership functions. The next layer then applies the fuzzy rules after which the outputs of the various rules are combined to form the outputs in again the next layer.

Another neuro-fuzzy controller showing a layered functionality is the Adaptive Network-based Fuzzy Inference System (ANFIS) introduced by Jang (1992). In the ANFIS the task of rule inference is performed in 6 layers. The first layer is the input layer, passing its data to the second layer where membership functions are applied to the incoming data. The resulting values

are sent to the third layer where the firing strength per rule is determined. In layer 4 all firing strengths are normalised after which in layer 5 the output of each rule is determined. In layer 6 all outputs are summed yielding the output of the ANFIS. The appealing aspect of this approach is that the system can be trained using back propagation.

2.3 Conclusions

The network topologies and algorithms presented in this section all have their advantages and disadvantages. In order to choose the algorithm that is most applicable for the work undertaken in this thesis, it should first be noted that accurate input-output data is available. As mentioned in section 2.1.1, supervised training is thus preferred to unsupervised training.

As will be discussed in the following chapters, all parameters that will be identified with neural networks are time independent. The use of recurrent neural networks consequently is not necessary. This results in the choice of a suitable network being limited to three candidates: back propagation neural networks, radial basis function networks or neuro-fuzzy networks. The last might well be an interesting option. However, the fact that two disciplines are combined in this approach, means that the number of tuning parameters increases drastically, thus increasing the problems for a structured approach. For this reason, the research in this thesis will be limited to the use of back propagation neural networks and radial basis functions.

Chapter 3

Review of neural/fuzzy modelling and control for underwater vehicles

The state of the art in applying neural networks and fuzzy logic for the modelling and control of AUVs is established in this chapter. In order to create an overview, a framework of control strategies using neural networks (in combination with fuzzy logic) is suggested and advantages and disadvantages for the several classes are identified. First, in section 3.1, an introduction to the hydrodynamics governing the movement of underwater craft is given and a general model describing the hydrodynamics is introduced. Then in section 3.2 the proposed classification framework is presented and the classes are introduced. In section 3.3 several examples from each class are presented and results are reported as found in the literature. These results are interpreted in section 3.4 and conclusions are distilled. Lastly, in section 3.5 the classes are compared and final conclusions are presented.

3.1 AUV models

Undoubtedly of great importance when performing simulations is the model used as a representation of the underwater vehicle. Models range from single input single output (SISO) transfer functions, based on the intuitive estimates of vehicle properties, to sophisticated models based on the identification of real systems with several degrees of freedom.

3.1.1 Equations of Motion

Unfortunately, often little attention is paid to the description of the model used. In general however, it is based on a fundamental model that can be derived from Newtonian formalisms[4] (Fossen, 2002). Inspired by Newton's second law, $m\dot{\boldsymbol{\nu}} = \mathbf{f}_c$, relating mass m, force $\mathbf{f}_c$ and acceleration $\dot{\boldsymbol{\nu}}$, Euler suggested a formulation later to be known as Euler's First and Second Axioms:

$$\left[\begin{array}{rcl} \dot{\mathbf{p}}_C & = & \mathbf{f}_C \\ \mathbf{p}_C & = & m\boldsymbol{\nu}_C \end{array}\right. \tag{3.1}$$

$$\left[\begin{array}{rcl} \dot{\mathbf{h}}_C & = & \mathbf{m}_C \\ \mathbf{h}_C & = & \mathbf{I}_C\omega \end{array}\right. \tag{3.2}$$

where $\mathbf{f}_C$ and $\mathbf{m}_C$ are the forces and moments working on the body, $\mathbf{p}_C$ represents the linear momentum of the body and $\mathbf{h}_C$ its angular momentum. ω is the angular velocity and $\mathbf{I}_C$ is an inertia matrix. The index C indicates reference of the vectors to the body's center of gravity.

[4] In this derivation bold face characters will be used for vectors

Using equations 3.1 and 3.2 a vehicle's six degrees of freedom rigid-body equations can be expressed as:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau} \tag{3.3}$$

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu},$$

with

$$\begin{aligned}
\boldsymbol{\eta} &= \text{position and orientation of the vehicle in earth fixed frame} \\
\boldsymbol{\nu} &= \text{linear and angular velocity of the vehicle in body fixed frame} \\
\mathbf{M} &= \text{inertia matrix} \\
\mathbf{C}(\boldsymbol{\nu}) &= \text{matrix consisting of Coriolis and centripetal terms} \\
\mathbf{D}(\boldsymbol{\nu}) &= \text{matrix consisting of damping or drag terms} \\
\mathbf{g}(\boldsymbol{\eta}) &= \text{vector of forces and moments due to gravitation} \\
\boldsymbol{\tau} &= \text{vector of control inputs}
\end{aligned}$$

The matrix $\mathbf{J}(\boldsymbol{\eta})$ converts velocity in a body fixed frame, $\boldsymbol{\nu}$, to velocity in an earth fixed frame, $\dot{\boldsymbol{\eta}}$. For the body-fixed velocity, $\boldsymbol{\nu} = [u\ v\ w\ p\ q\ r]^T$, the coordinate system is chosen as in figure 3.1 with the surge u pointing towards the bow of the craft, the sway v to starboard, and the heave w pointing down. The rotations around the surge, sway and heave axes, respectively roll p, pitch q and yaw r, obey the rules for a right hand coordinate system. The Earth-fixed system has elements respectively pointing north, east and towards the centre of the earth and

again three rotations about these axes.

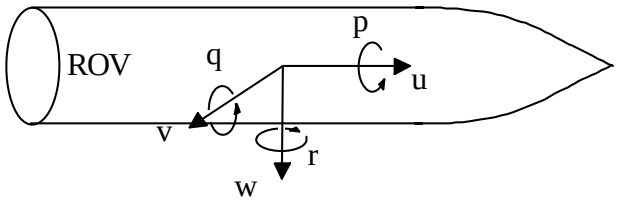


Figure 3.1: Definition of body-fixed coordinate system

A detailed derivation of these non-linear equations of motion can be found in Fossen (2002). Below a small summary of the modelled phenomena is given.

Mass and Inertia

In matrix $\mathbf{M}$ two inertial components are accounted for,

$$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_{A}. \tag{3.4}$$

The rigid body inertial matrix, $\mathbf{M}_{RB}$, represents the mass and inertia terms due to the mass and other physical characteristics of the craft. In a dense medium, such as water however, a considerable contribution to the mass originates from the medium. This so called 'virtual' or 'added' mass is accounted for by the matrix $\mathbf{M}_{A}$. Rather than interpreting this phenomenon as originating from fluid connected to the vehicle and thus increasing its mass, the added mass should be seen as hydrodynamic pressure induced forces and moments on the wetted surface proportional to the vehicle's acceleration. Operating in the wave zone, that is a water depth of typically less than half of the wave length, the vehicle will be affected by the (sinusoidal) wave loads. In this case the added mass matrix is a function of the wave frequency. For deeply submerged vehicles asymptotic values for low frequency ($\omega \rightarrow 0$) can be used (Fossen, 2002, p.

67) and hence the added mass (and thus the total mass), $\mathbf{M}_A$ can be assumed to be constant.

Coriolis and Centripetal forces

For matrix $\mathbf{C}(\boldsymbol{\nu})$, a similar discourse can be held. Both the Coriolis and centripetal forces are forces that are proportional to mass and inertia. Hence, the matrix can again be shown to consist of two matrices:

$$\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C}_{RB}(\boldsymbol{\nu}) + \mathbf{C}_A(\boldsymbol{\nu}). \tag{3.5}$$

While $\mathbf{C}_{RB}(\boldsymbol{\nu})$ represents forces and moments due to the mass and physical characteristics of the craft, $\mathbf{C}_A(\boldsymbol{\nu})$ incorporates the terms originating from the added mass.

Damping terms

In the damping matrix, $\mathbf{D}(\boldsymbol{\nu})$, four terms are combined:

$$\mathbf{D}(\boldsymbol{\nu}) = \mathbf{D}_P + \mathbf{D}_S(\boldsymbol{\nu}) + \mathbf{D}_W + \mathbf{D}_M(\boldsymbol{\nu}), \tag{3.6}$$

where:

$$\begin{aligned} \mathbf{D}_P &= \text{potential damping} \\ \mathbf{D}_S(\boldsymbol{\nu}) &= \text{linear and quadratic skin friction} \\ \mathbf{D}_W &= \text{wave drift damping} \\ \mathbf{D}_M(\boldsymbol{\nu}) &= \text{damping due to vortex shedding} \end{aligned}$$

and the subscript r denotes that drag is a function of relative velocity[5].

Similar to added mass, potential damping is introduced due to forces on the body when the latter is forced to oscillate. Contrary to the case of surface vessels, this term is of minor importance in underwater vehicles operating at great depth and is thus often neglected.

Skin friction effects can be shown to constitute of both a linear term and a quadratic term. Low frequency friction will induce a linear while high frequency effects will add a quadratic term. Skin friction will be higher for faster moving vehicles. This can be conveniently taken into account by expressing skin friction as a function of the relative velocity, $\boldsymbol{\nu}_r$

Like potential damping, wave drift damping only plays a major role at the surface where it can be interpreted as added resistance due to incoming waves.

Damping due to vortex shedding is a result of the non-conservative nature of a moving system in water with respect to energy. The viscous damping force due to this phenomenon is a function of the relative speed of the craft, its physical dimensions, hull shape and appendices (e.g. fins, rudder) and the density and viscosity of the surrounding medium.

Accurate calculation of these phenomena is difficult. Hence, often the damping is approximated by a diagonal matrix containing the linear and quadratic damping terms according to:

$$\begin{aligned} \mathbf{D}(\nu) &= -diag\{X_u,\, Y_v,\, Z_w,\, K_p,\, M_q,\, N_r\} \\ &\quad - diag\{X_{u|u|}|u|,\, Y_{v|v|}|v|,\, Z_{w|w|}|w|,\, K_{p|p|}|p|,\, M_{q|q|}|q|,\, N_{r|r|}|r|\}, \end{aligned} \tag{3.7}$$

where X_u, Y_v, Z_w, K_p, M_q and N_r are the linear terms, and $X_{u|u|}$, $Y_{v|v|}$, $Z_{w|w|}$, $K_{p|p|}$, $M_{q|q|}$, $N_{r|r|}$ the quadratic terms of the damping in six degrees of freedom.

However, it can be shown that the linear part of the drag matrix exhibits the same symmetry

[5]Velocity of the vehicle relative to the water current

as found in the rigid body mass matrix (Fossen, 2002)[p. 102] (for symmetry considerations in the rigid body mass matrix see Fossen (2002)[p. 99]), thus showing that the use of (3.7) can introduce significant errors in the drag estimation.

Gravitation and Buoyancy

The last term on the left-hand side of equation 3.3, $\mathbf{g}(\boldsymbol{\eta})$, models the so-called restoring forces which result from gravitation and buoyancy. While the gravity force is a vector working along a line through the craft's center of gravity, the buoyancy term is a force working along a line through the craft's center of buoyancy. As, in general, those two points do not coincide, the restoring forces will introduce both forces and moments respectively along and about the three body axes.

3.1.2 Practical implementations

As mentioned before, most research endeavours apply a plant based on the equations of motion from equation 3.3. Whether or not all the discussed phenomena are modelled is highly dependent on factors such as availability of a physical craft, measurements performed on such craft and the amount of modelled degrees of freedom. For example, in the case of a 1-dimensional model, equation 3.3 can be written as:

$$m\dot{\nu} + D(\nu)\nu + mg - B = \tau \tag{3.8}$$

in which m is the mass of the vehicle, D represents drag, g is the gravitational constant and B represents the buoyancy of the craft[6].

[6]For clarity the equivalent of $g(\boldsymbol{\eta})$ has been split up in gravitation working in the positive z-direction and buoyancy working in the negative z-direction

Models in this form have been used in Labonte (2002); Yuh (1994); Tsukamoto et al. (1997). As those models, obviously, do not take into account any cross coupling, i.e. influence due to control actions in other degrees of freedom, their use is limited. They do yield insight in the ability of the controller under investigation to control the (non-)linear problem. However, for real situations those controllers may not suffice unless cross coupling can be guaranteed to be negligibly small. Only one of the three cited articles utilising 1-dimensional models, reports practical results. In Tsukamoto et al. (1997) the position of a thruster, fixed in such a way that only movement in the x-direction is possible, is controlled.

A logical step towards a more complete model is to consider movement in three degrees of freedom (Seube, 1991; Wang et al., 1999b; Yuh and Lakshmi, 1993; Yuh, 1990; Wang et al., 2000; Pollini et al., 1997).

Choosing a model with three degrees of freedom is a logical choice as this enables control in a horizontal plane (surge, sway and yaw/heading) (Yuh, 1990; Yuh and Lakshmi, 1993; Pollini et al., 1997). After all, it is highly likely that a neutrally buoyant craft will remain in a horizontal plane if only control actions in this horizontal plane are performed. Controllers for these 'horizontal plane' models give promising and insightful results. Due to taking into account three degrees of freedom rather than one, the controller's ability to counteract cross coupling can be investigated. If, for example, separate thrusters are used for control of surge and sway, this will generally result in cross coupling between the various controllable degrees of freedom. As a result the controllers are not only tested for tracking, but also for their ability to cancel out the influence of control actions in other degrees of freedom. In a particular case (Wang et al., 1999b, 2000) the authors report that, due to the physical structure of their craft, ODIN (Omni-Directional Intelligent Navigator), the moments about the axes (pitch, roll and yaw) are all negligibly small, thus justifying a three degrees of freedom model.

Apart from the extension in modelled degrees of freedom, several of the cited models are based on measured parameters of real craft, thus making simulations more realistic. Measurements on ODIN (Wang et al., 1999b), Dolphin (Yuh, 1990) and Trailblazer (Pollini et al., 1997) are reported in the literature.

With respect to the degrees of freedom, the most challenging model to control is obviously one with six degrees of freedom. In numerous papers use of a model incorporating all six degrees of freedom is reported (Comoglio and Pandya, 1992; Akkizidis and Roberts, 1998; Wang et al., 1999a; Campa et al., 2000; Kim and Yuh, 2001; Li et al., 2002; Wang and Lee, 2002). Interesting to note is that the models in Wang et al. (1999a); Kim and Yuh (2001) and Wang and Lee (2002) are based on the aforementioned ODIN craft. The previously neglected moments around the axes (Wang et al., 1999b, 2000) have thus been taken into account in the model.

Typically the model is based on identification of real craft parameters for which a variety of craft are used: ODIN from the Autonomous Systems Laboratory of the University of Hawaii (Wang et al., 1999a; Kim and Yuh, 2001; Wang and Lee, 2002), the Deep Submergence Rescue Vehicle (DSRV) from the Naval Coastal Systems Center (Comoglio and Pandya, 1992), GARBI developed at the University of Barcelona and Girona (Akkizidis and Roberts, 1998) and the Semi-Autonomous Underwater Vehicle (SAUV) from the Korea Research Institute of Ships & Ocean Engineering (Li et al., 2002).

3.2 Control Strategies

There are several strategies to apply neural networks (NNs) to the control of a system. The dynamics of the system can be controlled by training a neural network to represent the inverse dynamics of the system. A neural network can learn a model of the forward dynamics and based

on this model a controller is then constructed and consecutively used for control of the system. Alternatively, a neural network can be used to replace the conventional system identifier in model based control schemes. Regardless of the chosen strategy, a neural network will normally consist of at least one hidden layer with non-linear activation functions as the use of such hidden layers with non-linear activation functions enables the neural network to solve non-linear classification problems. As underwater craft are highly non-linear systems (Akkizidis and Roberts, 1998; Li et al., 2002), it is envisaged that neural networks will prove beneficial.

Fuzzy-neural networks provide an interesting mix of techniques. These networks combine the qualities of both neural networks and fuzzy logic in order to make maximum profit from both techniques. Fuzzy logic is known to be adept at the translation of human expertise into a set of (If-Then-Else) rules (Tsoukalas and Uhrig, 1997, Ch. 5). This translation of human expertise offers a way to equip the controller with initial, even though possibly very limited, knowledge of the system. By combining fuzzy logic with the learning capabilities of neural networks, the lack in flexibility normally encountered in fuzzy logic networks, is obviated (Akkizidis and Roberts, 1998).

In the following the techniques and strategies, as briefly discussed in the above, are analyzed and classified. A classification into three major control strategies is proposed: *(i) Combined Control and Learning (CCL), (ii) Separate Control and Learning (SCL)* and *(iii) Augmented Control (AC)*. As will become clear, at first sight CCL and SCL appear to be closely related to Direct Control and Indirect Control respectively, as found in the literature (Narendra and Parthasarathy, 1990). According to Narendra and Parthasarathy (1990) in direct control the parameters of the controller are directly changed such as to minimise a predefined error function of the output. In indirect control, first a model of the plant is identified, after which a controller is constructed based on the identified plant model. Although for CCL only control architectures using direct

control have been found, this classification cannot be used. This is due to the fact that time is the limiting factor in the identification schemes. As will be explained in section 3.2.1, in CCL there is relatively little time available to perform learning and thus identification due to the fact that learning is performed in an online fashion. In SCL there is an abundance of time as training is performed offline. Direct control and indirect control do not distinguish between the offline learning performed in SCL and the online learning performed in CCL and are thus not suited to reflect the differences in available time for training found in CCL and SCL.

3.2.1 Combined Control and Learning (CCL)

In the field of (semi-)autonomous control of underwater vehicles numerous articles can be found presenting both theoretical and practical studies applying neural networks that perform the tasks of control and online learning in the same loop. Control of the craft and training of the neural network are performed at consecutive moments in time (Akkizidis and Roberts, 1998; Farrell et al., 1990; Guo et al., 1995; Kim and Yuh, 2001; Labonte, 2002; Seube, 1991; Venugopal et al., 1992; Wang et al., 1999a,b, 2000; Wang and Lee, 2002; Yuh, 1990; Yuh and Lakshmi, 1993; Yuh, 1994). Time sharing of the two actions is symbolized by the single feedback loop in figure 3.2: the setpoint or desired system output is sent to the controller. Based on its representation of the craft, the controller determines the necessary control signal that is then sent to the actuators. The output of the craft is then monitored and fed back to the NN controller. Based on the fed back output and the desired output, the neural network learns. As training is performed online and continuously, the neural network adapts to changing dynamics.

Initially, while the neural network is learning the proper behaviour, a simple conventional controller can improve the performance of the controller. The conventional controller is over-ruled once the neural network is properly trained. This initial controller can also act as a teacher

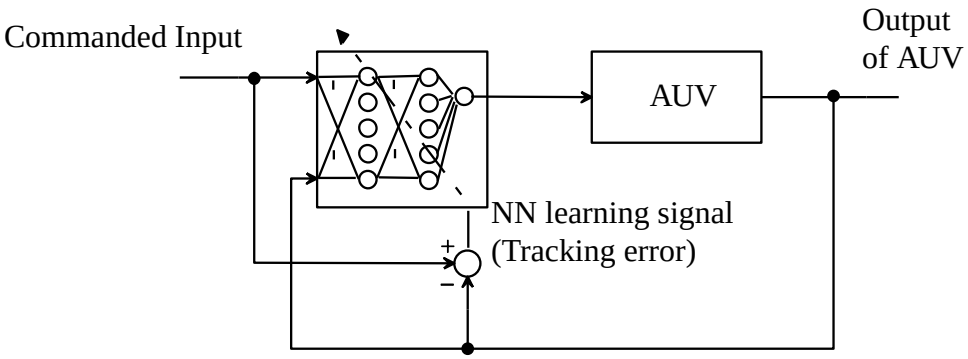


Figure 3.2: Block level representation of combined control and learning

(Guo et al., 1995; Wang and Lee, 2002). Other architectures, mainly neuro-fuzzy controllers, use expert knowledge programmed into the fuzzy controller to perform initial control. Again learning is performed based on the error between the output of the system and the desired output (Akkizidis and Roberts, 1998; Kim and Yuh, 2001). A third possibility, which clearly shows the learning capabilities of the craft, is to have the neural net control the craft without any initial information regarding the proper control actions. The system, without any *a priori* knowledge, learns the proper control actions first by applying a set of inputs to the neural net and using the outputs to teach the NN controller to perform the desired control action until a satisfactory performance is obtained (Farrell et al., 1990; Seube, 1991; Venugopal et al., 1992; Yuh, 1990; Yuh and Lakshmi, 1993; Yuh, 1994; Labonte, 2002). Of course in this strategy the system response to the initial control actions, can at best be estimated roughly. Due caution should be taken that this initial learning does not cause harmful situations.

The popularity of this approach can be explained by the following:

- It is often possible to use a similar control architecture as in conventional control (the conventional controller is replaced by a neural network).

- The complexity of the NN controller is limited as normally an inverse model of the craft is identified. Inverse model identification can be done with one neural network.

- Learning of, and adaptation to new system dynamics, is performed based on the latest data.

An apparent drawback of this approach is that, due to the fact that learning is performed in between control actions, the time for learning is restricted by the desired update rate of the control action. As a result the computational complexity of the learning algorithm used in this approach is restricted.

3.2.2 Separate Control and Learning (SCL)

In contrast to Combined Control and Learning, only a few examples of Separate Control and Learning (see figure 3.3) can be found in the literature.

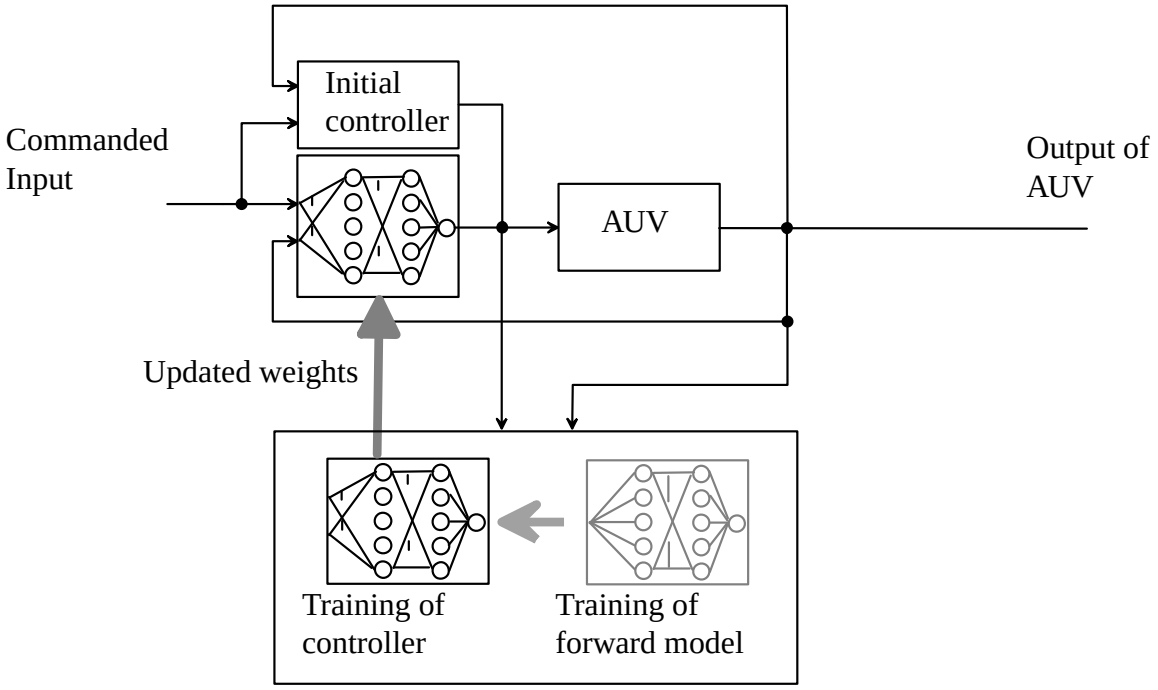


Figure 3.3: Block level representation of separate control and learning

In SCL initial control is performed by a conventional controller or fuzzy network with *a priori* knowledge of the craft. During this period of initial control, which is sometimes referred to as the infancy of the craft, the neural networks are trained. The first step in this training

process is to train a neural network to represent the forward dynamics of the craft. This is done by presenting the neural network with control actions and the resulting craft's behaviour. Once a neural network has identified the forward dynamics, this neural network is used to train a NN controller. After completion of this second training step the latter neural network (or a copy thereof) can be used for control of the real craft. In the above described case, the controller network is trained to represent the inverse dynamics using a NN model of the forward dynamics. It is however also possible directly (i.e. without the use of a forward network) to train a neural network to represent the inverse dynamics. In figure 3.3 this choice is indicated with the shaded forward model. In any case, the neural network seizes control of the craft once it is satisfactorily trained. Training of a new forward model and consecutively new NN controller is then started outside the control loop.

A reason for the relative scarcity of applications using SCL might be that this strategy inherently needs a more extensive architecture, which is illustrated by Fujii and Ura (1990); Ishii et al. (1994, 1995); Ishii and Ura (2000); Ura et al. (1990). There are however several reasons to advocate this approach. Probably the most apparent difference between SCL and CCL is that in SCL learning and control are separate tasks that can be performed simultaneously. As a result, more time can be taken to train the neural network and thus more extensive, and possibly more appropriate, network architectures can be applied. Recurrent networks, for example, can be used to represent memory effects in the craft and are appropriate for filtering of noisy input data (Polycarpou and Ioannou, 1993). The fact that recurrent networks need training algorithms with increased complexity compared to, for example, normal back propagation networks, does not have to pose a problem due to the relaxed time constraints in SCL.

3.2.3 Augmented Control (AC)

Augmented control relies on the assumption that a conventional controller can perform a significant portion of the control task under all circumstances. To aid the conventional controller, a neural network is added, often in feed forward or feed backward configuration as demonstrated by figure 3.4. It adds the necessary response to unmodelled or poorly modelled dynamics, disturbances and other uncertainties, to the total control action. The most apparent advantage of AC is that it is normally possible to use an old architecture with only slight changes (Campa et al., 2000).

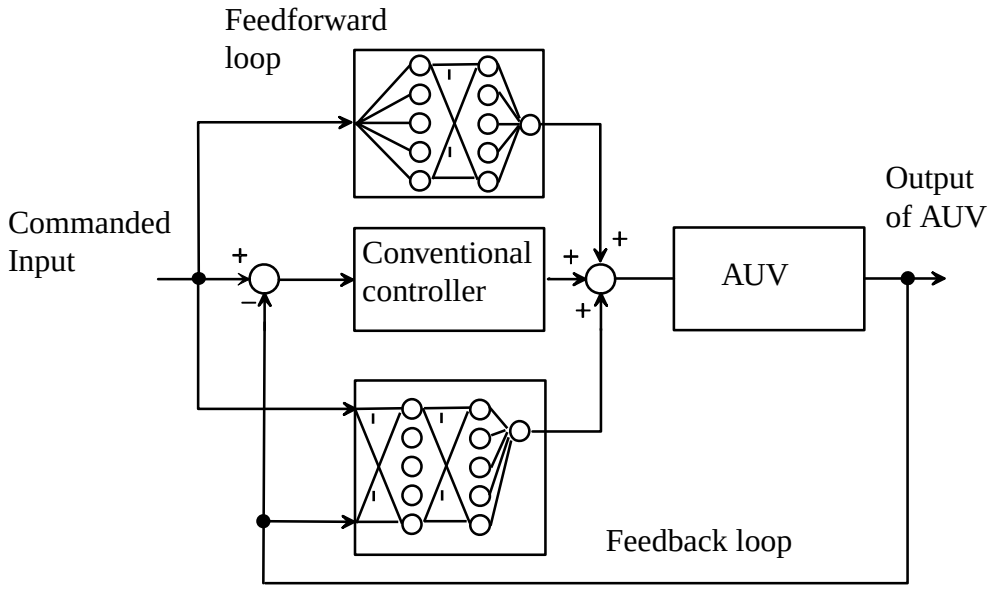


Figure 3.4: Block level representation of augmented control

The neural network can be trained outside the control loop (after all the conventional controller is assumed to yield some means of control). Therefore, from a learning point of view there is no need for very fast processors. Although most examples of augmented control use feed forward networks combined with back propagation algorithms (Li et al., 2002; Yamamoto, 1995; Pollini et al., 1997), there are examples of architectures that employ other topologies such

as recurrent networks (Kodogiannis et al., 1996).

3.3 Examples

In this section, examples of the three classes will be given. To aid the interested reader in making a more extensive comparison, table 3.8 gives an overview of characteristics from the references used throughout this chapter.

3.3.1 Combined Control and Learning: Example A

Guo et al. (1995) present an architecture that can be classified as CCL. A neural network with 2 inputs, 5 hidden nodes and 1 output node, controls motion in the horizontal plane while a compass provides feedback of the control parameters. The heading angle and its derivative are the inputs to the neural net. The latter computes a thrust force ΔT such that a left thruster force $T_0 + \Delta T$ and a right thruster force $T_0 - \Delta T$ give the desired resultant force vector. Back propagation is used to update the neural network in two phases. In the first phase, the initialisation phase, the neural network is trained off-line with a linear controller as its automated teacher. After off-line training the obtained weights are used as initial weights in the controller. During actual deployment of the craft the weights are further updated by minimising a scalar objective function incorporating tracking error and control rate requirement:

$$J(t_k) = \frac{1}{2}[\alpha(\psi_d(t_{k+1}) - \psi(t_{k+1}))^2 + \beta(\Delta\dot{T}(t_k))^2] \tag{3.9}$$

with $\psi_d(t_{k+1})$ and $\psi(t_{k+1})$ respectively the desired tracking and the actual tracking.[7] $\Delta\dot{T}(t_k)$ is the derivative of the control action. The parameters α and β make a trade off between tracking

[7]Due to discretisation, the control output $\Delta T(t_k)$ affects the craft dynamics at $t = t_{k+1}$

performance and control effort. The learning rate is set to 0.001 with 10 learning cycles per sampling period of 0.05 seconds.

The described controller is used to perform simulations and pool tests on a craft developed in the Department of Naval Architecture and Ocean Engineering of the National Taiwan University. The authors claim a good agreement between simulations and pool tests. The observed performance is shown in table 3.1. Unless explicitly specified otherwise, the $0-100\%$ risetime is used.[8]

Table 3.1: Control of Yaw in Guo et al. (1995)

Overshoot	22%
Risetime	6.3s
Stepsize	30°

In Guo et al. (1995) it can be seen that the control action to follow a desired step is an attenuated oscillation in the simulation. In the pool test however, the controller keeps on generating an oscillating control signal, although the response of the system is very close to the desired value. This is undesirable as the actuators (i.e. thrusters) are needlessly switched on and off, causing extra wear or possibly even increasing oscillations. Other tests, probing the aptitude of the controller to compensate for static and dynamic disturbances also show a controller being able to provide reasonable tracking. Of special interest is that the authors show that the neural networks control the craft properly for significantly different forward speeds. These differences in forward speed generally lead to significant changes in the craft's dynamics which would cause considerable problems for conventional linear control methods.

[8] For underdamped systems, i.e. systems showing overshoot in the step response $0-100\%$ risetime is a useful index. For overdamped systems it is more appropriate to use the $10-90\%$ risetime (Dorf and Bishop, 1995)[pp. 222]

3.3.2 Combined Control and Learning: Example B

The controller described by Venugopal et al. (1992) is shown in figure 3.5.

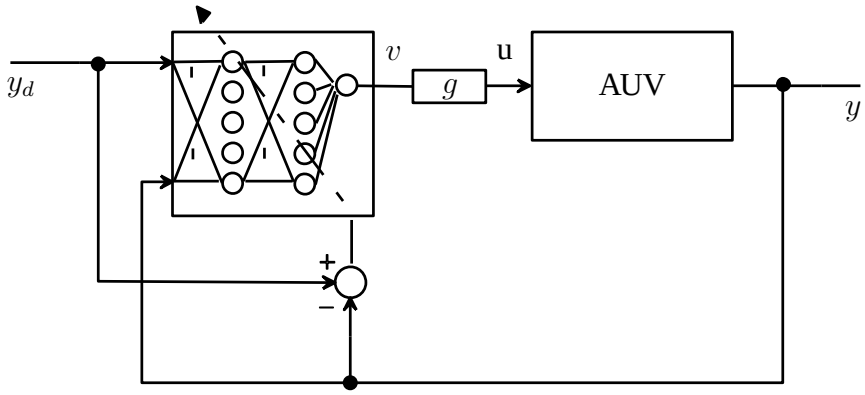


Figure 3.5: Block level representation of the controller presented by Venugopal et al. (1992)

It is consecutively used for the control of three degrees of freedom. The NN controller, with 2 inputs, 20 1^{st} hidden layer and 10 2^{nd} layer hidden nodes and 1 output neuron, is presented with the desired output, y_d, and the actual system output, y. The neural network then computes an output, v, which, after multiplication by a gain factor, g, forms the input to the craft in the form of a rudder angle. The gain is introduced for the following reason. Standard back propagation learning would use an error signal consisting of the output of the neural network minus the desired output of the neural network. However, as the system dynamics are assumed to be unknown, the desired output of the neural network is unknown. A method to circumvent this problem was proposed by Yuh and Lakshmi (1993), who use an estimate of the desired controller output based on the system output error and an approximation of the craft's inertia matrix. As Venugopal et al. (1992) claim that the convergence is slow in this case, a gain factor, which is proportional to the inverse of the Jacobian, is introduced. This gain factor is constantly updated using the following rule:

$$g(k) = g(k-1) - \alpha \frac{\delta e_T(k)}{\delta g(k)} \tag{3.10}$$

with $u(k) = g(k) \cdot v(k)$ and $e_T(k) = \frac{1}{2}(y_d(k) - y(k))^2$, 3.10 can be written as:

$$g(k) = g(k-1) + \alpha[y_d(k) - y(k)] \frac{\delta y(k)}{\delta u(k)} \cdot v(k) \tag{3.11}$$

The neural network is trained solely online using the system output error. The back propagation learning rate, η, is varied between 0.001 and 0.3 while the learning rate, α, used for updating $g(k)$ is set to 0.001.

Simulation results, using a model of the 'ocean voyager' are presented. In these simulations pitch, yaw and heave are subject to control. As only for pitch a step input was used, only this measurement will be used for comparison. Table 3.2 lists the results for a learning rate of $\eta = 0.005$.

Table 3.2: Control of pitch in Venugopal et al. (1992)

Overshoot	18%
Risetime	54s
Stepsize	8.75°

It is found that overshoot, rise time and adaptivity are a function of the back propagation learning rate. Higher learning rates decrease the rise time and increase adaptivity in case of changing dynamics but also increase the overshoot of a step response. In table 3.3 the relation between learning rate and controller performance is shown.

The two numbers for a learning rate of 0.005 indicate two different measurements. It is interesting to see that the same controller under exactly the same circumstances can result in different responses. This is due to the random initialisation of the weights in the neural network.

Table 3.3: Relation between learning rate and controller performance in Venugopal et al. (1992)

Learning rate	Overshoot [%]	Risetime [s]
0.3	82	23
0.05	68	32
0.005	17, 26	63, 73
0.002	0	136 ($10-90\%$)

Also the yaw is controlled. Instead of a step input, a trapezoidal input is used. The obtained tracking error is about 2°. The learning rate used in this simulation is 0.002. For this learning rate it is observed that the commanded rudder angle oscillates around a mean value of zero. Increased learning rates introduce an increase of those oscillations. Especially for a repeated trapezoidal input, it is clear that, although the oscillations, initially apparent on the controller yaw angle, decrease in time, there is a considerable oscillation on the commanded rudder angle signal. Robustness of the controller is investigated by commanding the craft to increase its forward speed in a sinusoidal manner. The forward thruster revolution speed is sinusoidally varied between $0.26m/s$ and $2.8m/s$ with a period of $1000s$. In general it can be said that this change in forward speed changes the craft's response from linear for low speed to non-linear for high speed. The authors show that, with these disturbances present, the tracking of yaw has degraded but an increasing performance over time can be seen. The neural networks are learning to cope with this time-varying disturbance.

3.3.3 Separate Control and Learning: Example A

A good example of this strategy is the controller called SONCS for Self-Organising Neural-net Control System (Fujii and Ura, 1990; Ishii et al., 1994, 1995; Ishii and Ura, 2000; Ura et al., 1990) that uses a so-called real-world and imaginary-world system. In the imaginary-world system, which is a copy of the real-world system (including the craft's dynamics), a controller

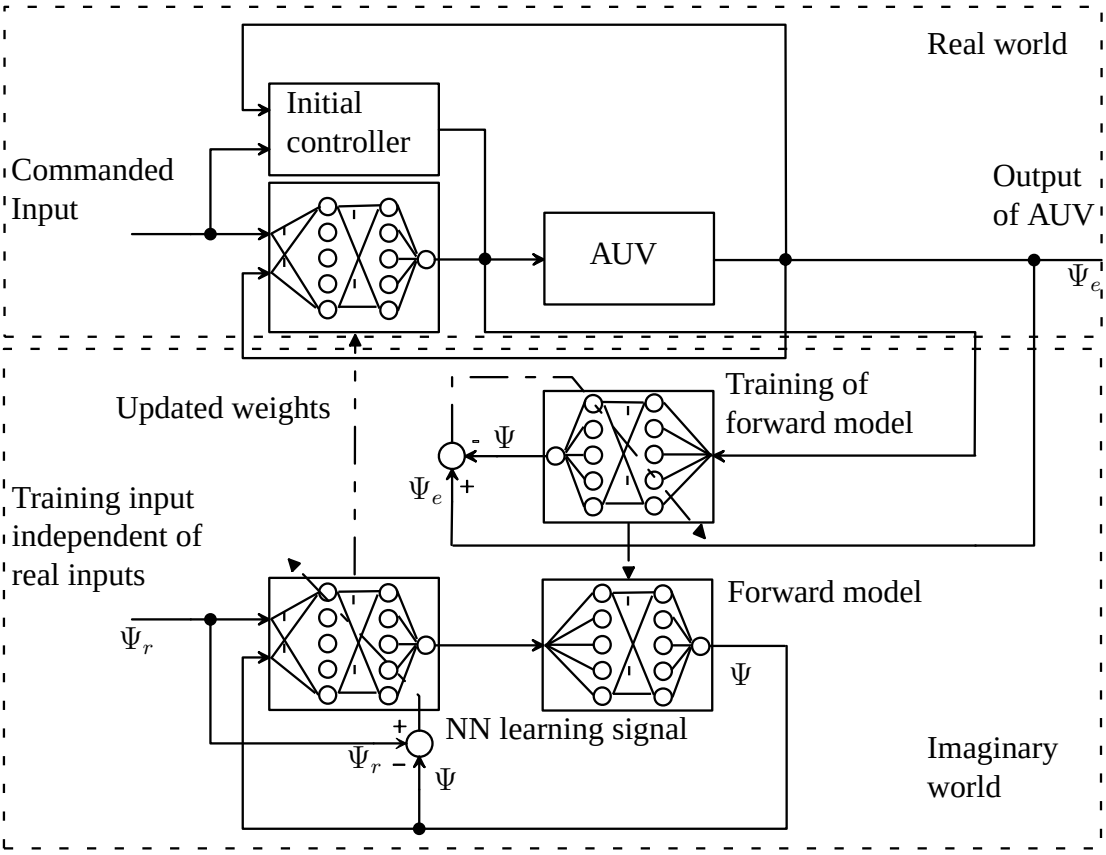


Figure 3.6: Block level representation of the controller presented by Fujii et al. (1993)

is constantly trained using an online updated recurrent forward model of the craft dynamics. This model, in turn, is regularly updated using inputs and outputs from the actual craft. At specified intervals the real-world controller updates its weights with the learned imaginary-world controller weights. Learning is thus performed in a loop totally independent from the control loop. To control the craft during its infancy, a fuzzy controller is used that implements some rudimentary behaviours and is overruled once the NN controller is trained.

Figure 3.6 shows the architecture with the real-world and imaginary-world controllers. Back propagation is used to minimize:

$$Ef = \frac{1}{2}\sum_{k}\left[(\psi_e(k) - \psi(k))^2 + (\Delta\psi_e(k) - \Delta\psi(k))^2\right] \tag{3.12}$$

$$Ec = \frac{1}{2}\sum_{k}\left[(\psi_r(k) - \psi(k))^2 + (\Delta\psi_r(k) - \Delta\psi(k))^2\right] \tag{3.13}$$

for the forward model and the imaginary world controller respectively. The interpretation of Ψ, Ψ_e and Ψ_r is shown in figure 3.6. Δ denotes a change in the variable from the last to the current time point.

Ishii and Ura (2000) report experimental results using SONCS. Tests are performed using the Twin-Burger test bed (Fujii et al., 1993). In these tests the Twin-Burger is required to follow a periodically changing reference signal ($+0.5$ rad to -0.5 rad and vice versa every 10 s) while a constant disturbance of 2.0 Nm is applied to the craft. The used learning and momentum parameters are 0.1 and 0.2 respectively. The obtained performance is shown in table 3.4.

Table 3.4: Control of yaw in Ishii and Ura (2000)

Overshoot	10%
Risetime	5.1s
Stepsize	$57.3°$

It is interesting to note that in the imaginary world model, the overshoot and risetime were estimated to be 0% and 6.4 s. As the controller is overdamped in this case, the $10-90\%$ risetime was taken. The difference in forward model and true dynamics of the vehicle thus result in a slightly underdamped real world controller. In another experiment the behaviour of the controller with a periodically varying disturbance was tested; a sinusoidal signal with an amplitude of 2.0 Nm and a frequency of 0.2 Hz was applied as disturbance while the Twin-Burger follows the same periodically varying heading angle. Again the controller introduced proper reverse-phase forces to counteract the disturbances. In a path-following experiment in a tank test, the Twin-Burger was programmed to follow a 3m $\times$ 3m square path at constant depth with a current ($0.08m/s$) applied as a disturbance. Again the Twin-Burger more or less follows the prescribed

path. It can be clearly seen that tracking of the path improves over time. This shows that the NN controller is superior to the initial fuzzy controller.

3.3.4 Separate Control and Learning: Example B

In Comoglio and Pandya (1992) a Cerebral Model Arithmetic Computer (CMAC) is used to learn the required control command given a desired system output.

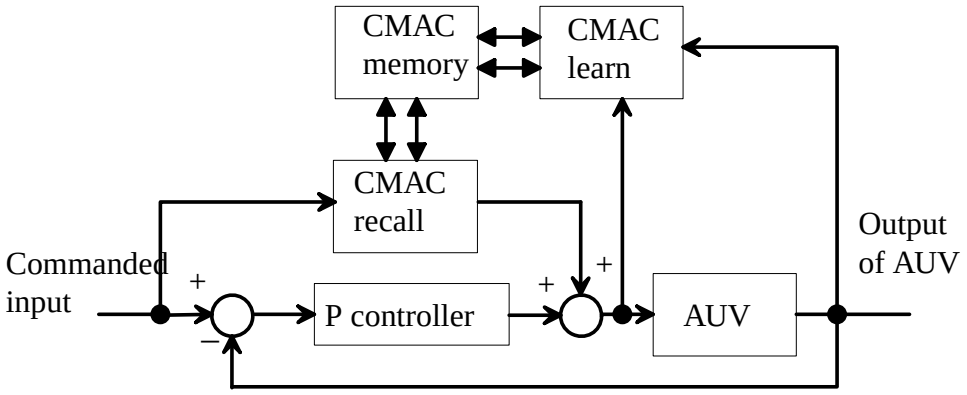


Figure 3.7: Block level representation of the CMAC controller

As shown in figure 3.7 the CMAC consists of a controller (CMAC recall) and a learning loop (CMAC learn). While the CMAC recall is controlling the craft, the CMAC learn loop is constantly updating an inverse model of the craft which is stored in a memory. A constant gain controller provides simple initial control. For a properly trained CMAC controller the input, and thus output, of the constant gain controller will be zero. The CMAC controller thus fully controls the craft. In case the CMAC controller does not represent the inverse dynamics of the craft properly, the constant gain controller will contribute to the total control signal. The constant gain controller thus provides a second control loop (parallel to the CMAC recall control loop) when updating of the CMAC recall block is necessary. Classification of this controller as AC would be unjust as a simple constant gain could easily be modelled by the CMAC controller

itself.

A lookup table stores the inverse model of the craft as follows: an output vector is recalled given a certain input vector. Direct mapping of the non-linear input space on a memory space might ask for more memory than available. Therefore a hash coding technique follows the mapping to limit the latter to the available memory. Nearby input vectors will yield the same output while distant input vectors will yield different outputs; generalisation is thus performed on the set of input vectors.

Simulations with the CMAC were performed using a model developed at the Naval Coastal Systems Center (Humphreys, 1976). In those simulations only the pitch angle is controlled. First the CMAC gains knowledge regarding the craft's dynamics through off-line training. Subsequently the controller is used to follow two commanded pitch angle sequences. The simulation results are given in table 3.5.

Table 3.5: Control of pitch in Comoglio and Pandya (1992)

Overshoot	16%
Risetime	6.5s
Stepsize	11.6°

3.3.5 Augmented Control: Example A

In Li et al. (2002) a two layer neural network is used to augment a linear feedback controller. While a PD controller controls the linear part of the dynamics, the neural network is designed to control the nonlinear uncertainties of the craft's dynamics. The reader is referred to Li et al. (2002) for the derivation of the control law:

$$\mathbf{u} \;=\; \mathbf{M_{RB}}[\ddot{\mathbf{q}}_\mathbf{d} - \mathbf{\Lambda}(\dot{\mathbf{q}} - \dot{\mathbf{q}}_\mathbf{d})] - \mathbf{u_{PD}} - \hat{\mathbf{W}}^\mathbf{T}\phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}), \tag{3.14}$$

with M_{RB} the craft's rigid-body inertia matrix, $\mathbf{q}$ and $\mathbf{q}_d$ respectively the actual and desired position and orientation vector, Λ a design matrix, u_{PD} the linear controller output, $\hat{W}$ the weight matrix and $\phi(x)$ the so called basis function vector (input vector) of the two layer neural network. This control law is implemented as shown in figure 3.8.

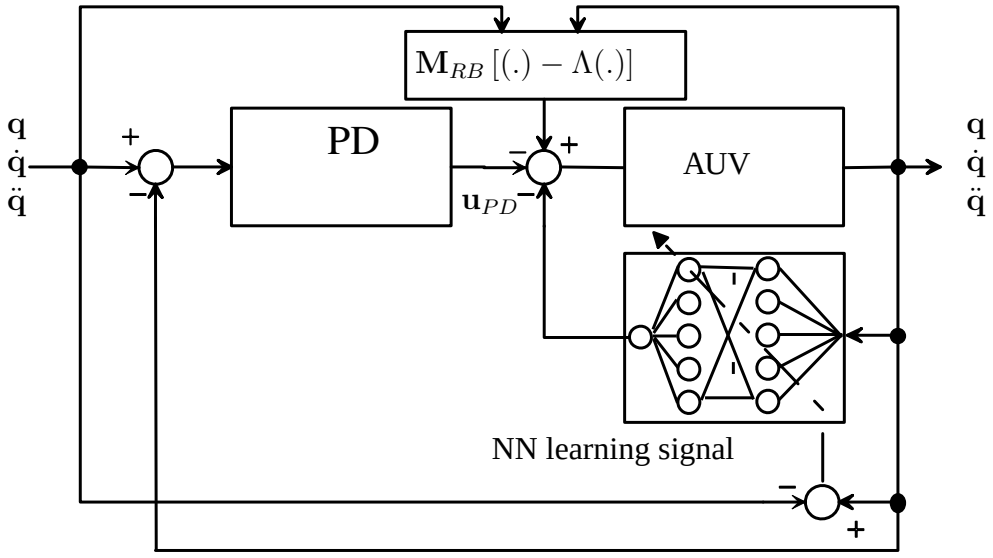


Figure 3.8: Principle of operation of the controller presented by Li et al. (2002)

The used neural network is shown in figure 3.9. The block named 'Input Signal Preprocessing', preprocesses the input vector $\mathbf{q}$ and its first and second order derivatives yielding the following basis function vector:

$$\phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \;=\; [{\boldsymbol{\phi}_1}^T,\ {\boldsymbol{\phi}_2}^T,\ {\boldsymbol{\phi}_3}^T]^T \;=\; [\ddot{\mathbf{q}}^T,\ \dot{\mathbf{q}}^T\,(\dot{\mathbf{q}} \otimes \dot{\mathbf{q}})^T,\ \mathbf{G}^T\,(\mathbf{G} \otimes \mathbf{G})^T]^T, \tag{3.15}$$

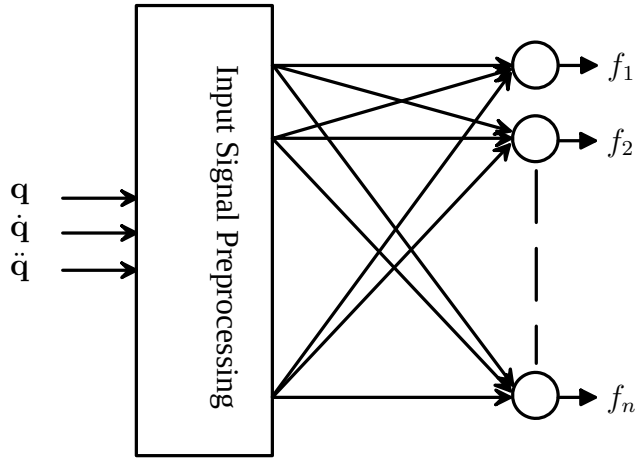


Figure 3.9: Neural network with preprocessing layer

with $\mathbf{G} = [\sin(\phi)\ \sin(\theta)\ \cos(\phi)\ \cos(\theta)]^T$ and $\otimes$ the Kronecker product. The vector on the right hand side of equation 3.15 is then fed to the output layer yielding the neural net control signals in six degrees of freedom.

A model of the SAUV (Hong, 2000) (Semi-Autonomous Underwater Vehicle) developed in the Korea Research Institute of Ships & Ocean Engineering (KRISO) was used for simulations with the proposed controller. In the simulations the performance of the augmented controller was compared to the performance of a controller consisting of linear feedback control combined with a sliding mode controller. The simulation results show that the augmented controller approximates the nonlinear uncertainties in about ten seconds. Comparison with the linear feedback / sliding mode controller shows that after the initial ten seconds, the augmented controller shows far better tracking than the conventional controller.

3.3.6 Augmented Control: Example B

In Kodogiannis et al. (1996) an augmented control system making use of a NN predictive model is presented. In this article, control of the depth is performed on a craft with one degree of

freedom. Future work will focus on control of a craft in six degrees of freedom.

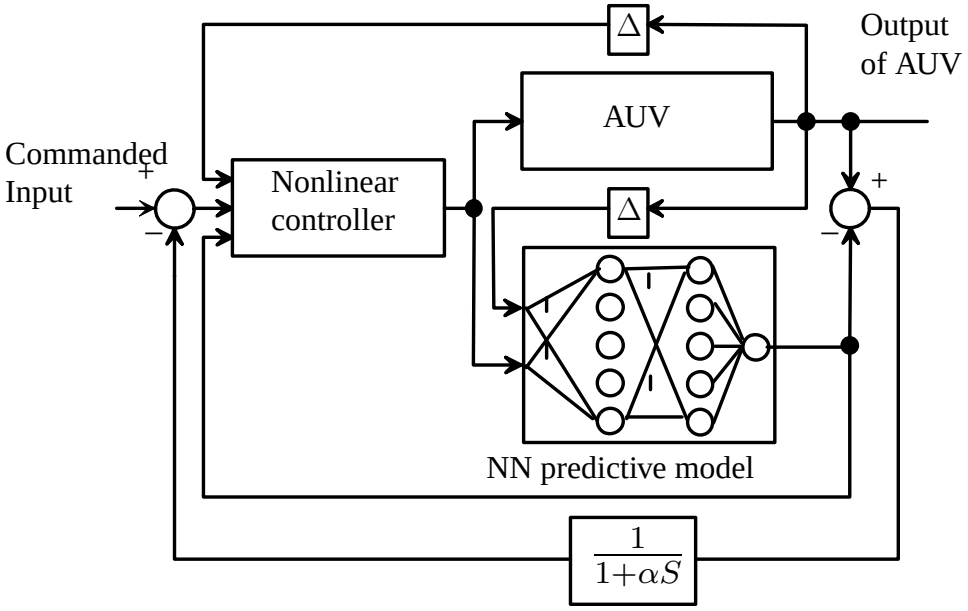


Figure 3.10: Block schematic of the model predictive control scheme

In model predictive control (see figure 3.10) with neural networks, the neural network models the forward dynamics of the system and performs predictions of future outputs. Based on this knowledge of future behaviour, a control output is determined such that the difference between the predicted output and the desired output is minimised. The presented NN prediction model predicts five consecutive future outputs of the system of which only the first one is used for determination of the new control command. Two different neural networks are tested for use as the forward model: a modified Elman network (Haykin, 1999, Ch. 15) and a newly proposed architecture called Autoregressive Recurrent neural network (ARNN). Both networks use recurrent connections to capture memory effects of the system.

The ARNN was designed to contain memory while training can still be performed in a fast way using general back propagation. For faster convergence the only recurrent connections are connections feeding back to the same neuron. As can be seen in figure 3.11 the network consists

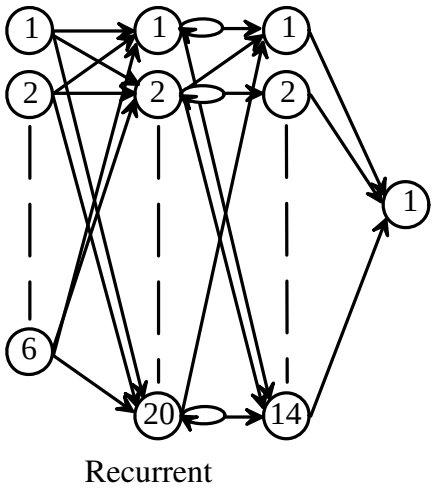


Figure 3.11: ARNN architecture

of two hidden layers (of which only the first one has recurrent connections) with sigmoidal activation functions and a linear output node. To use general back propagation the algorithm is extended to include self-feedback, i.e. for the first hidden layer the output is determined by the output of the input layer and the output of the first hidden layer itself at previous points in time. As five future outputs are to be predicted, the recurrent layer has a memory of five previous points. The network with 6 input nodes, 20 1^{st} hidden layer nodes, 14 2^{nd} hidden layer nodes and 1 output node uses one past control signal and five past depth indications as inputs and predicts the future depth as output.

The modified Elman network, as depicted in figure 3.12, can be trained with a standard back propagation algorithm as the recurrent connections have fixed weights. To extend the capabilities of the Elman network to learn time-variant system knowledge, the nodes in the feedback loop are equipped with self-feedback. Three inputs, consisting of the current control signal and current and previous system states, are fed through the two hidden layers of 12 and 14 nodes respectively and result in the predicted system output.

Pool tests, in which the MPC scheme controlled a craft called the ‘Aquacube’, show that both

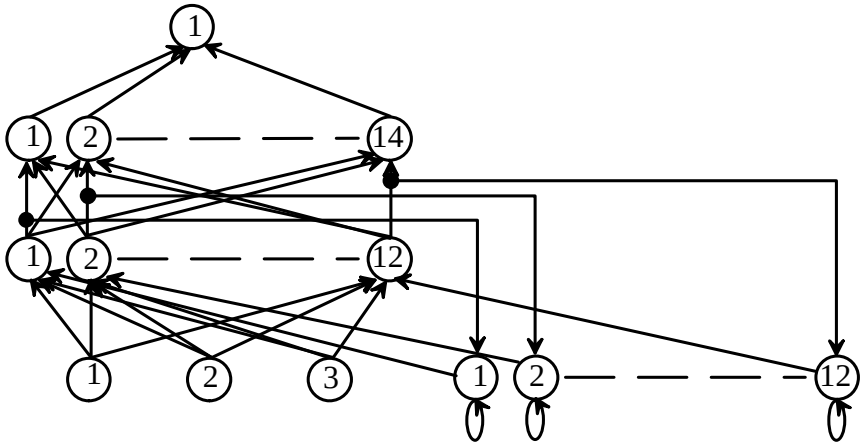


Figure 3.12: Modified Elman architecture

neural networks result in a good tracking of the desired path. For the ARNN the mean square error in prediction of the system output is $5.16 \cdot 10^{-4}m$, while the mean squared tracking error is $5.23 \cdot 10^{-2}m$. The Elman network performs slightly less well with a mean squared error in system output prediction of $7.48 \cdot 10^{-4}m$ and a mean squared tracking error of $5.55 \cdot 10^{-2}m$. The authors claim that 'the improvements obtained with recurrent networks are due to the fact that a minimum control effort was used to achieve the specific performance'. This results in a smoother response to the demanded steps in the input. Results of a step response using the ARNN are shown in table 3.6.

Table 3.6: Control of heave in Kodogiannis et al. (1996)

Overshoot	14%
Risetime	8s
Stepsize	$57.3°$

3.4 Discussion

3.4.1 Combined Control and Learning

In both examples the neural network attempts to model the inverse dynamics of the craft. The main difference between the two strategies is the way the NN controller is trained. In Venugopal et al. (1992) the neural network is trained with the error between system output and desired output. An adaptive gain between the neural network and the craft eases convergence. In Guo et al. (1995) weight updates are performed using an objective function with the output error as well. However, in order to keep the actuator signals within limits, the derivative of the actuator control signal is also input to the objective function. One issue, related to the learning algorithm used in both CCL examples, becomes apparent from the low learning rate. As the weight update of the neural network is based on the last system output, the neural network approximation abilities will only be improved for that particular input-output pair. In other words, online learning results in an improved local estimate of the (inverse) dynamics of the craft. For higher learning parameters this would result in the neural network generalising for the last input-output pair only. Batch learning with a sequence of previous input-output pairs can obviate this problem. The obvious limiting factor is, however, the time available between two control commands. Hence the simpler solution, to use a small learning rate, is normally chosen. Prime advantage of online learning is that disturbances are shown to be counteracted successfully.

Generally, in CCL, a considerable oscillation is observed in the response to a step input. For example in Venugopal et al. (1992) overshoots as high as 82% were observed. This oscillation is not as pronounced in SCL, which can be explained as follows. In CCL, the neural network can be regarded as a normal feedback controller with adaptive weights. The error signal is

fed back to the neural network and is used to update the weights, thus changing the controller transfer function. The moment a step input is presented to the input of the controller, the error increases dramatically as the craft's output is still at or near the old desired value. The step is thus interpreted as a drastic change in the dynamics and the neural network will try to learn the 'new' dynamics with considerable effort. While getting closer to the setpoint, learning will be slower due to the decreasing error. However, when the neural network reaches the desired setpoint, it has been trained such that its output will always drive the craft past its setpoint. Overshoot and a damping oscillation will thus occur. In cases of high learning rates, the closed loop system can become unstable and increasing oscillations will occur.

Furthermore it should be noted that only one degree of freedom is controlled. Although the literature does report CCL using neural networks of more than one degree of freedom (Yuh and Lakshmi, 1993; Wang et al., 1999b), it might prove difficult to identify an inverse model of the craft. Modelling of the dynamics in the various degrees of freedom with separate neural networks may alleviate this problem.

An interesting claim by Guo et al. is that direct control should be preferred to indirect control for the reason that 'the network architecture of forward model for plant dynamics is very problem specific. It usually requires large network with recurrent connections'. In CCL adaptation to changing dynamics or circumstances can be performed quickly and there might thus be no need for networks that can represent the craft's memory effects. In SCL however, as will be discussed in the following, a forward model of the craft is used to train a controller for a prolonged period of time. It will thus need to represent memory effects for which, for example recurrent connections in the neural network are used. For those networks generally more complex and more time consuming training algorithms are used.

3.4.2 Separate Control and Learning

Example A and B show two different strategies within SCL. The SONCS algorithm is based on the identification of a NN craft model and consecutively training of a NN controller with the identified model. In Example B, rather than identifying a forward model, an inverse model is identified. This model is then copied into the controller. As the controller is, ideally, the inverse of the craft, the total transfer function will be unity. The craft output will thus be equal to the desired output.

From figure 3.6, it becomes apparent straight away that this strategy leads to a complex architecture. Due to the possibility of offline training, architectures and algorithms that typically need more time to converge to a solution, can be used. This yields several advantages:

- Memory exhibited by the system can be modelled in the neural network, for example, by using recurrent connections.
- The forward model and the controller can be trained until the desired performance is obtained.
- In case a forward model is identified for training of a controller, the forward model can also be used for, for example, self-diagnosis and path planning.

But there are also inherent disadvantages:

- The forward model represents a model of the past which might prove problematic in case of changing dynamics or environmental influences.
- Without human supervision it is hard to assure convergence of the forward model.
- Considerable computational power is required to train the networks.

Especially the fact that the forward model represents a model of the past is of concern in the underwater world. Proper modelling of a marine craft's dynamics is highly dependent on, for example the craft's velocity and currents in the water, both of which can change suddenly and drastically.

Studying typical step responses of SCL and comparing these to CCL step responses, highlights a rather appealing advantage of SCL. The examples, described in section 3.3, detail results on both yaw and pitch control. For control of yaw, tables 3.1 and 3.4 in example CCL A and SCL A respectively, show that SCL combines smaller overshoot with shorter rise times. The same can be seen from tables 3.2 and 3.5 in examples CCL B and SCL B respectively, for the control of pitch. In the latter two tables almost the same overshoot is reported. But in SCL example B the rise time is more than a factor 8 shorter than in CCL example B. For the same rate of response the overshoot in example CCL B would thus be higher. Generally, SCL shows a far smoother response without the oscillations observed in CCL step responses. The reasons for the improved performance are as detailed in section 3.4.1.

The CMAC controller presented in Comoglio and Pandya (1992) demonstrates the need for the new classification. Although this controller applies indirect control, the inverse model of the system is learned outside the control loop. Hence, rather than classifying this controller as CCL because of the applied indirect control, it should be classified as SCL.

3.4.3 Augmented Control

The two examples for AC show that the original controller can be augmented using neural networks in several ways. The controller itself can be augmented, as in Example A with a feedback NN controller. Alternatively, augmentation can take place by a neural network modelling the dynamics. In Example B, the process plant model is fully represented by a neural network.

However, for increased robustness a neural network can be used in parallel to a simple model of the plant dynamics. This would improve the performance while the NN plant model is still learning to represent the dynamics properly.

In Example A, a 6 DOF model is presented. However, the article reports results of the controller performance for only 3 degrees of freedom. Additionally, as presented in Li et al. (2002), the craft model uses only diagonal matrices, thus neglecting the cross coupling between degrees of freedom normally present in marine craft. In effect, Li et al. (2002) implements a separate controller for each degree of freedom.

In both cases the original architecture is simply expanded with the neural network. There is no need for a totally new design. This can highly reduce development time, and thus cost. The disadvantage of such an approach is that the neural network will possibly have to compensate for improper modelling rather than for nonlinearities. Assume for example that a linear controller is augmented with a neural network. Once the operating point drifts away from the setpoint, for which the linear controller was designed, chances are that the system consisting of nonlinear craft and linear controller becomes unstable. This would obviously lead to an increased 'work load' for the NN controller. Finally it should be kept in mind that the advantage of being able to yield results quickly may be undone by the fact that the results will always be dependent on the original design. As for the original design, often the use of neural networks was never intended, it may not be able to profit from the capabilities of neural networks fully.

3.4.4 Overall Discussion

Table 3.7 gives an overview of characteristics as distilled from the literature. From this table it can be concluded that all three classes offer certain advantages in certain circumstances which will be clarified below.

CCL offers a fairly simple way to control the craft if the craft is designed for low speed deployment or if the tracking demands are relaxed. The oscillations occurring for big changes in the setpoint (i.e. desired velocity or attitude) show that CCL controllers are less useful for agile craft that operate at both high and low speeds. On the other hand, even though with considerable overshoots, CCL is capable of dealing with changing dynamics. This is mainly due to the fact that the changing dynamics are instantly detected by the neural network and acted upon. This is a considerable advantage compared to more conventional controllers.

Although in CCL one does need fast processors, the neural networks used for control are constrained to relatively simple networks. As the control architecture is also relatively simple this strategy would be more interesting for commercial applications in which one is not, *per se*, interested in good forward models of the craft.

In SCL this problem is avoided as more appropriate networks can be applied. As a result SCL would generally be a more flexible approach, offering proper control for a higher range of velocities and other dynamic parameters. An environment with quickly changing conditions (e.g. weather conditions) however, may prove too challenging for SCL as model and controller updating requires more time than in CCL. Also, SCL is (in terms of hardware and development costs) a more demanding approach which may prove to be an issue in industrial applications. But as it allows for elaborate NN architectures, SCL offers a good test bed for research focussing on the application of exotic structures and research endeavours to model the craft using neural networks.

AC combines advantages of the two other approaches, namely the simple, straightforward control architecture from CCL and the separate control and learning loops from SCL, but a major point of concern is whether the used conventional controller contributes a considerable portion of the total control action under all circumstances, as assumed. For operation close to the operating

point of the conventional controller, however, AC might well be preferred by industry to both CCL and SCL. AC, after all being a combination of neural network and conventional techniques, diminishes the chances of total controller malfunctioning. As robustness is a major issue in industrial applications, this advantage may well be the dominant factor for manufacturers.

Compared to control methods not using neural networks, the most important difference and advantage of neural network based control schemes, is their ability to adapt for changing dynamics. Where simpler, and often linear, control schemes will show degraded performance or even divergent behaviour, all three neural network based approaches offer a means of counteracting the changes successfully.

Table 3.7 Comparison of the three approaches

	Advantages	**Disadvantages**	**Applications**
CCL	• Old controller can often be replaced • Learning with newest data / circumstances • No FWD model necessary • The complexity of the NN controller is limited • Direct control is less demanding (Guo et al., 1995)	• Learning is restricted by the control step size • Batch learning hard to apply • Finding inverse model for 6 DOF difficult • Possibly high overshoot in step responses	• Low but accurate velocity operation • Dynamic station keeping • Coarse control at high velocity

Continued on next page

Table 3.7 Comparison of the three approaches

	Advantages	Disadvantages	Applications
SCL	• Less demanding computationally • Memory can be modelled • Forward model and controller can be trained until desired performance is obtained • Forward model can be used for, for example self-diagnosis and path planning	• Possibly complex architecture • Controller is trained with model of the past • Without human supervision it is hard to ensure convergence of the forward model • Considerable computational power is required to train the networks	• Underwater vehicles for (NN) research purposes • Large velocity range • Relatively calm circumstances (e.g. lakes)
AC	• Simple implementation • Old architecture can be used • Computationally less demanding	• Conventional controller might be totally inadequate if dynamics change drastically • Innovations hindered by 'what was there'	• Industrial applications • Operation relatively close to operating point of conventional controller

3.5 Conclusions

In this section several examples of (semi-) autonomous control of underwater vehicles using neural networks were presented and categorised according to the proposed classification framework consisting of the groups: CCL, SCL and AC. In Combined Control and Learning (CCL), a neural network controls the craft and adapts at the same time to counteract for changing dynamics or circumstances. In the literature only examples of CCL have been found that apply direct control. This means that the neural network identifies an inverse model of the craft's dynamics, hence yielding a unity closed loop response. In Separate Control and Learning, (SCL), learning is performed outside the control loop and is hence independent of the maximum time between two control actions. Both direct control and indirect control, in which a NN model of the craft is identified that is then used for training of a NN controller, can be applied. The third group, Augmented Control (AC) combines the possibilities of CCL, SCL and conventional control methods. A neural network is used to augment the control action of a conventional controller, either using direct or indirect control.

As the chosen NN architecture, learning algorithms and test bed highly influence the presented results, it is far from trivial to draw conclusions as to which strategy, CCL, SCL or AC, should be regarded to be most favourable. However some general characteristics have been found for the defined classes. CCL and SCL both offer the advantage of high adaptability to changing dynamics and circumstances. Of these two, CCL adapts slightly faster, but also shows higher overshoots when following a trajectory. Control using SCL is more accurate, but fast and continuous changes may prove to be too demanding for this class. AC offers the back up of reliable conventional controllers. However, in the examples of AC found, the increased reliability is paid for by a decreased adaptability compared to both CCL and SCL.

3.6 Research objectives

Although the use of neural networks in industrial applications is becoming more popular, there still is a tendency to limit the use of neural networks to non-critical functions or systems, or to provide conventional backup. This is a direct result from the fact that, currently, no generic analytical methods are available to prove convergence of online learning for neural networks. As a result of this reliability issue, it is expected that the near future will show the use of neural network controllers with conventional controllers as a back up. For this reason the work presented in this thesis seeks to optimise the advantages that can be obtained from such an AC controller. In an AC architecture there is no need to identify an inverse model. As earlier it was argued that making an inverse model can be troublesome, in this work the choice was made to identify a forward model. In chapter 4 a new identification method will be presented that allows full flexibility in using the obtained neural model in combination with existing classical models or controllers. This flexibility is obtained by making optimum use of the available knowledge regarding the craft's dynamics. By using neural networks to identify and represent individual parameters in the model, the latter can be easily reformulated to be used in various control schemes.

Table 3.8 Comparison of AUV characteristics

Reference	Network	Learn. Algor.	Control	Sim	Test	Vehicle
			CCL			
(Akkizidis and Roberts, 1998)	NF	Fuzzy Error Prop.	Direct	N	N	Garbi
(Farrell et al., 1990)	FFW NN	BP	MRAC	Y	Y	SeaGrant
(Guo et al., 1995)	FFW NN	BP	Direct	Y	Y	NTU
(Kim and Yuh, 2001)	NF	SANFIS	Direct	Y	N	ODIN
(Labonte, 2002)	Adaline	LMS	Direct	Y	N	
(Seube, 1991)	FFW NN	Viability theory	Direct	Y	N	Dolphin
(Venugopal et al., 1992)	FFW NN	BP	Direct	Y	N	Ocean Voyager
(Wang et al., 1999a)	NF	SONFON	Direct	Y	N	ODIN
(Wang et al., 1999b)	NF	SANFON	Direct	Y	N	ODIN
(Wang et al., 2000)	NF	SANFIS	Direct	Y	N	ODIN
(Yuh, 1990)	FFW NN	BP	Direct	Y	N	
(Yuh and Lakshmi, 1993)	FFW NN	BP	Direct	Y	N	
(Yuh, 1994)	FFW NN	BP	Direct	Y	N	ODIN
			SCL			
(Comoglio and Pandya, 1992)	CMAC		Direct	Y	N	DSRV
SONCS	Rec. NN	BP	Indirect	Y	Y	Twin-Burger

Continued on next page

Table 3.8 Comparison of AUV characteristics

Reference	Network	Learn. Algor.	Control	Sim	Test	Vehicle	Conventional Control
				AC			
(Campa et al., 2000)	FFW NN	BP	Direct	Y	N		LQR
(Li et al., 2002)	FFW NN	BP	Direct	Y	N	SAUV	PD
(Kodogiannis et al., 1996)	R NN	BP	Direct	Y	Y	AquaCube	MPC
(Pollini et al., 1997)	FFW NN	BP	Direct	Y	N	Trailblazer 25	FBLC
(Wang and Lee, 2002)	R NF	R-SANFIS	Direct	Y	Y	ODIN	PD

FBLC	Feedback Linearisation Control
FFW / R NN	Feedforward / Recurrent Neural Network
LMS	Least Mean Square algorithm. Also known as ‘delta rule’
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
(R) NF	(Recurrent) Neuro-Fuzzy network
PD	Proportional and Differential (controller)
SONCS	(Fujii and Ura, 1990; Ura et al., 1990; Fujii et al., 1993; Ishii et al., 1994, 1995; Ishii and Ura, 2000)
Adaline	Adaptive Linear Element
SANFIS	Self Adaptive Neuro Fuzzy Inference System
SANFON	Self Organised Neuro Fuzzy Optimisation Network
SONFON	Self Adaptive Neuro Fuzzy Optimisation Network

Chapter 4

A novel identification strategy

From chapter 3 it is clear that there are ample possibilities to use neural networks for the tasks of identification and control of underwater vehicles. However, they all have in common that no *a priori* knowledge regarding the craft's dynamics is used. At best, they are used in parallel to a model of the craft to model the unknown part of the model. In this chapter, a new approach for the identification of marine vehicles, making use of neural networks, is presented. Key aspects of the novel approach are that, using *a priori* knowledge regarding the model, the neural networks identify individual parameters in the model, and that the identified parameters can be used both in a forward model and in controller architectures.

In section 4.1, the state of the art in neural modelling is presented. In section 4.2, an alternative identification approach, in which individual parameters are identified, is suggested. This approach is then thoroughly presented in section 4.3. Finally, in section 4.4, preliminary conclusions are drawn.

4.1 Neural modelling in 'parallel-to-model' fashion

In state of the art control architectures using neural networks the latter are normally used in parallel with conventional models or controllers in a switching or output-blending fashion. To obtain an idea of the function approximated by a neural network in such a parallel-to-model fashion, the dynamical model (see page 23):

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}, \tag{4.1}$$

is expressed in state-space form. First equation (4.1) is re-ordered:

$$\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}\left[\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}(\boldsymbol{\eta})\right]. \tag{4.2}$$

Taking the state vector as $\boldsymbol{\nu}$ and the inputs to the system as: $\boldsymbol{\tau}(t)$ and $\boldsymbol{\eta}(t)$, (4.2) can be written in state-space form:

$$\begin{aligned} \dot{\boldsymbol{\nu}} &= \boldsymbol{\Phi}\left[\boldsymbol{\nu}(t), \boldsymbol{\tau}(t), \boldsymbol{\eta}(t)\right], \\ \mathbf{y} &= \boldsymbol{\Psi}\left[\boldsymbol{\nu}(t)\right] + \mathbf{e}(t), \end{aligned} \tag{4.3}$$

with $\boldsymbol{\Phi}\left[\boldsymbol{\nu}(t), \boldsymbol{\tau}(t), \boldsymbol{\eta}(t)\right]$ the right hand side of (4.2) and $\boldsymbol{\Psi}\left[\boldsymbol{\nu}(t)\right]$ simply $\boldsymbol{\nu}(t)$. The error term $\mathbf{e}(t)$ represents the measurement noise and will, for the moment, be neglected for clarity.

Figure 4.1 shows the corresponding block diagram. Assuming that one has partial knowledge regarding the function $\boldsymbol{\Phi}$, a neural network can be used to model the unknown part of the system in parallel with the known part of the system as depicted in figure 4.2. In this approach one assumes: $\boldsymbol{\Phi} = \boldsymbol{\Phi}_M + \hat{\boldsymbol{\Phi}}$ where $\boldsymbol{\Phi}_M$ corresponds to the known part of $\boldsymbol{\Phi}$ and $\hat{\boldsymbol{\Phi}}$ to the unknown

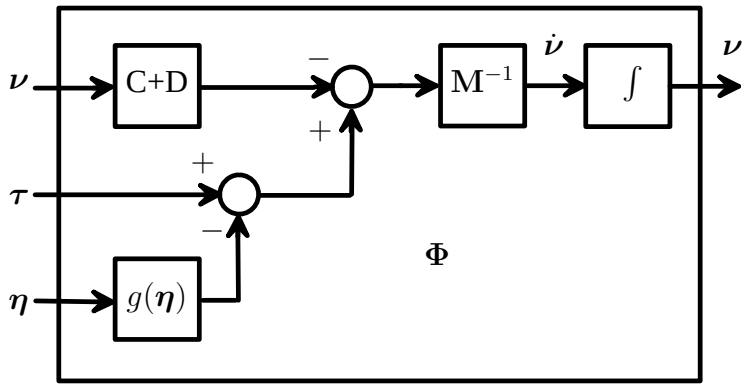


Figure 4.1: State space representation of the non-linear dynamic equations of motion

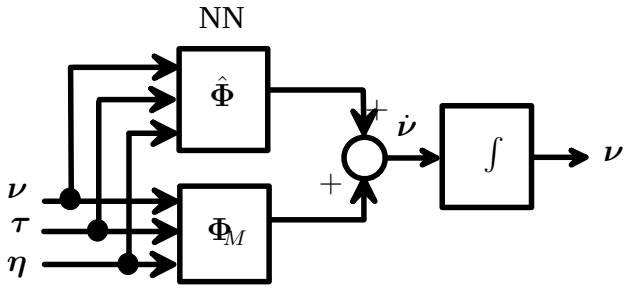


Figure 4.2: State space representation of the non-linear dynamic equations in parallel with a neural network to model unknown parameters

part approximated by the neural network. If the same assumption is made for the matrices $\mathbf{M}^{-1}$, $\mathbf{C}(\boldsymbol{\nu})$, $\mathbf{D}(\boldsymbol{\nu})$ and $\mathbf{g}(\boldsymbol{\eta})$, (4.2) can be written as[9]:

$$\begin{aligned}\dot{\boldsymbol{\nu}} &= \mathbf{M}_M^{-1}\left\{\boldsymbol{\tau} - \mathbf{C}_M(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}_M(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}_M(\boldsymbol{\eta})\right\} + \underline{\mathbf{M}_M^{-1}\left\{-\hat{\mathbf{C}}(\boldsymbol{\nu})\boldsymbol{\nu} - \hat{\mathbf{D}}(\boldsymbol{\nu})\boldsymbol{\nu} - \hat{\mathbf{g}}(\boldsymbol{\eta})\right\}} \\ &+ \underline{\hat{\mathbf{M}}^{-1}\left\{\boldsymbol{\tau} - \mathbf{C}_M(\boldsymbol{\nu})\boldsymbol{\nu} - \hat{\mathbf{C}}(\boldsymbol{\nu})\boldsymbol{\nu} - -\mathbf{D}_M(\boldsymbol{\nu})\boldsymbol{\nu} - \hat{\mathbf{D}}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}_M(\boldsymbol{\eta}) - \hat{\mathbf{g}}(\boldsymbol{\eta})\right\}}. \qquad (4.4)\end{aligned}$$

The underlined terms in (4.4) represent $\hat{\boldsymbol{\Phi}}$. These terms will be estimated by the neural network.

[9] N.B.: the inverse of the mass matrix is modelled as a sum of a known and an unknown matrix. Of course the unknown part of $\mathbf{M}$ is not equal to the inverse of the unknown part of $\hat{\mathbf{M}}^{-1}$: $(\mathbf{M}_M + \hat{\mathbf{M}})^{-1} \neq \mathbf{M}_M^{-1} + \hat{\mathbf{M}}^{-1}$

4.2 Neural modelling of individual parameters

In the previously described modelling approach the neural network makes no, or only partial use of the available *a priori* knowledge. Due to looking at the neural network as a fully unknown non-linear mapping between the plant's input data and output data, knowledge regarding the model structure is lost. As briefly mentioned in chapter 3, on page 59, substantial advantages can be obtained by using neural networks to model individual parameters in the model. This way of using neural networks for modelling purposes is demonstrated in figure 4.3. The neural networks, NN_1, NN_2 and NN_3 in figure 4.3 model:

$$NN_1 \Rightarrow \hat{\mathbf{M}}^{-1};\ NN_2 \Rightarrow \hat{\mathbf{C}} + \hat{\mathbf{D}};\ NN_3 \Rightarrow \hat{\mathbf{g}}(\eta), \tag{4.5}$$

which is a considerably easier task, demonstrating that the parallel-to-model neural network mapping is unnecessarily complicated. Furthermore use can be made of known structural properties during the learning stage. Examples of such structural properties are independence of the rigid body mass matrix on the velocity, $\boldsymbol{\nu}$, and known coupling between certain degrees of freedom. Use of these properties can significantly shorten the training process of the neural networks. Furthermore, as parameters are identified separately, it is straightforward to reformulate the model to derive control signals given desired outputs. In other words: an inverse model of the craft dynamics can easily be obtained. And lastly, if one of the parameters is accurately known before starting the proposed identification process, the proposed model allows use of this parameter, rather than having to identify a possibly inferior model of the known parameter.

The main restriction in the training of the neural networks in this new approach is that the inputs and outputs of the various blocks to be represented by the neural networks have to be

measurable, or deductible from measurements for supervised learning to be applicable (Haykin, 1999).

To benefit from the mentioned advantages, the alternative configuration of the neural networks, as shown in figure 4.3 is proposed.

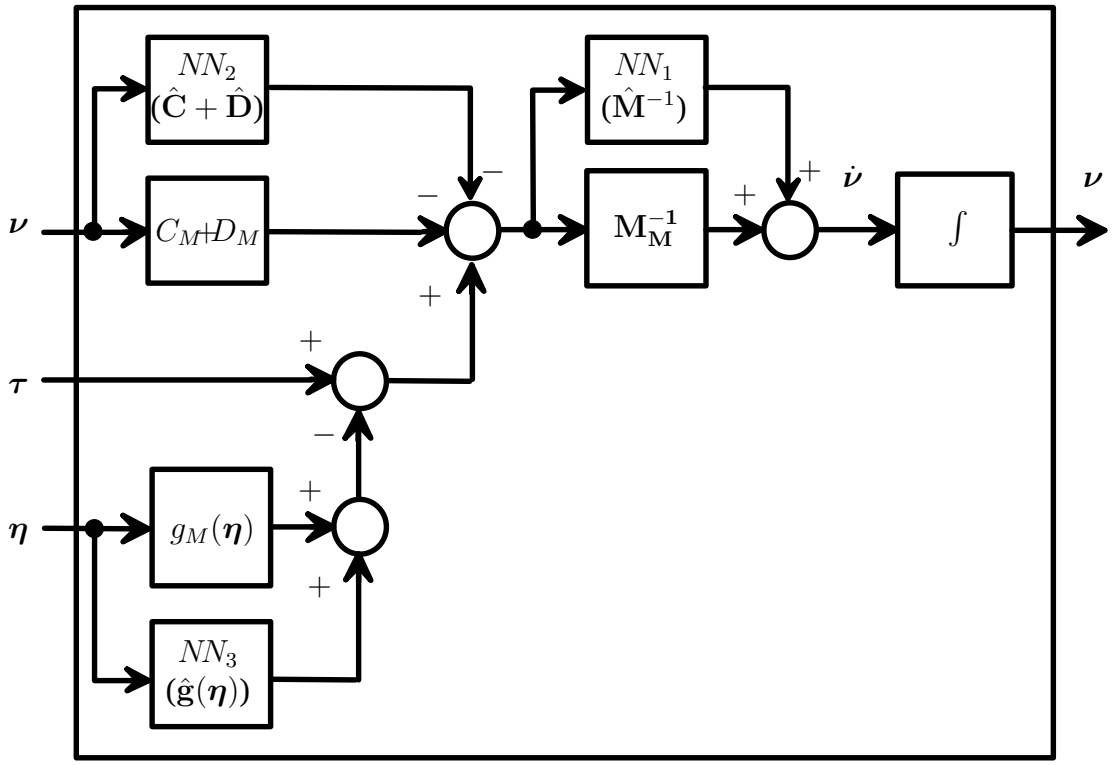


Figure 4.3: State space representation of the non-linear dynamic equations with several neural networks to model unknown parameters

However, as some parameters in the model from figure 4.3 are constant (notably the mass matrix), the use of neural networks to represent these parameters is not necessary. Nevertheless *identification* of these parameters using neural networks can prove beneficial as will be shown in section 4.3.3. The proposed identification model will thus be similar to figure 4.3 with constant (matrix) parameters instead of neural networks, as appropriate.

4.3 A novel identification method using neural networks

4.3.1 Assumptions

Control forces and moments

It is assumed that accurate measurements of the control forces and moments are readily available. If accurate models of the thruster dynamics are available, this is a justifiable assumption. Additionally it is assumed that there are no environmental disturbances.

Restoring forces

In the proposed method it is assumed that the restoring forces can be calculated. This is an acceptable assumption as the restoring forces can be described as:

$$\begin{aligned} \mathbf{g}(\boldsymbol{\eta}) &= -\begin{bmatrix} \mathbf{f}_g^b + \mathbf{f}_b^b \\ \mathbf{r}_g^b \times \mathbf{f}_g^b + \mathbf{r}_b^b \times \mathbf{f}_b^b \end{bmatrix} \\ &= -\begin{bmatrix} \mathbf{R}_b^n(\boldsymbol{\Theta})^{-1}(\mathbf{f}_g^n + \mathbf{f}_b^n) \\ \mathbf{r}_g^b \times \mathbf{R}_b^n(\boldsymbol{\Theta})^{-1}\mathbf{f}_g^n + \mathbf{r}_b^b \times \mathbf{R}_b^n(\boldsymbol{\Theta})^{-1}\mathbf{f}_b^n \end{bmatrix}, \end{aligned} \tag{4.6}$$

with $\mathbf{f}_g^b$, $\mathbf{f}_b^b$, $\mathbf{r}_g^b$ and $\mathbf{r}_b^b$ the gravity and buoyancy forces and the centre of gravity and buoyancy respectively, and $\mathbf{R}_b^n$ the transformation matrix from the body-fixed frame to the NED frame. The restoring forces are a function of relatively easily obtainable parameters. This can be shown by expanding equation 4.6:

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} (W-B)\sin(\theta) & & \\ -(W-B)\cos(\theta)\sin(\phi) & & \\ -(W-B)\cos(\theta)\cos(\phi) & & \\ -(y_gW - y_bB)\cos(\theta)\cos(\phi) & + & (z_gW - z_bB)\cos(\theta)\sin(\phi) \\ (z_gW - z_bB)\sin(\theta) & + & (x_gW - x_bB)\cos(\theta)\cos(\phi) \\ -(x_gW - x_bB)\cos(\theta)\sin(\phi) & - & (y_bW - y_bB)\sin(\theta) \end{bmatrix}. \tag{4.7}$$

In equation 4.7, W and B are the weight and buoyancy forces working on the craft respectively. The parameters x, y and z with indices g and b are the three dimensional coordinates of the craft's centre of gravity and the centre of buoyancy respectively. These parameters can be estimated relatively accurately beforehand or when a change to the craft is made. The roll angle ϕ and pitch angle θ can be obtained from the inertial measurement unit (IMU) and the restoring forces can thus be calculated.

Accelerations

The accelerations of the craft need to be measurable, or derivable, in all six degrees of freedom. As, at present, underwater vehicles normally have an onboard highly accurate inertial measurement unit, the linear accelerations and velocities (in six degrees of freedom) can be obtained without further investments. For the angular accelerations the IMU output will have to be differentiated as an IMU measures angular velocities rather than accelerations.

Constant mass

A standard procedure in control of deeply submerged marine vehicles, is to approximate the mass matrix with its counterpart for zero wave frequency, ω, (Fossen, 1994, p. 50):

$$\mathbf{M} = \lim_{\boldsymbol{\omega} \to 0} \mathbf{M}(\boldsymbol{\omega}) \tag{4.8}$$

This yields a constant mass matrix and a Coriolis matrix linearly dependent on the velocity $\boldsymbol{\nu}$.

4.3.2 Outline of identification process

The craft parameters mass, Coriolis and centripetal terms and damping, are assumed to be unknown and will be identified in an online fashion.

Initially the mass matrix, $\mathbf{M}$, is identified. The Coriolis matrix, $\mathbf{C}$, can then be calculated. The only other unknown parameter: the damping matrix, $\mathbf{D}(\boldsymbol{\nu})$, can subsequently be identified using the identified mass and Coriolis matrices.

As the mass matrix can be assumed to be constant for deeply submerged underwater vehicles[10], an explicit representation is distilled using neural networks merely during the identification process. Hence an explicit representation for the Coriolis and centripetal terms can be derived. In contrast to the mass matrix, exact expressions for the damping matrix are not available; only relatively simple, oftentimes empirical, models with non-linear terms are currently used. Hence, a neural network is used to identify the damping. But moreover, the neural network is used to represent the damping, as neural networks can represent any real and smooth function on a closed set (Hornik et al., 1989). Therefore, the representation of the damping does not hinge

[10] It is tacitly assumed that the mass (distribution) of the craft does not change due to, for example, fuel consumption. Such low frequent changes can however be accounted for easily.

on a heuristic model. The proposed identification model is depicted in figure 4.4.

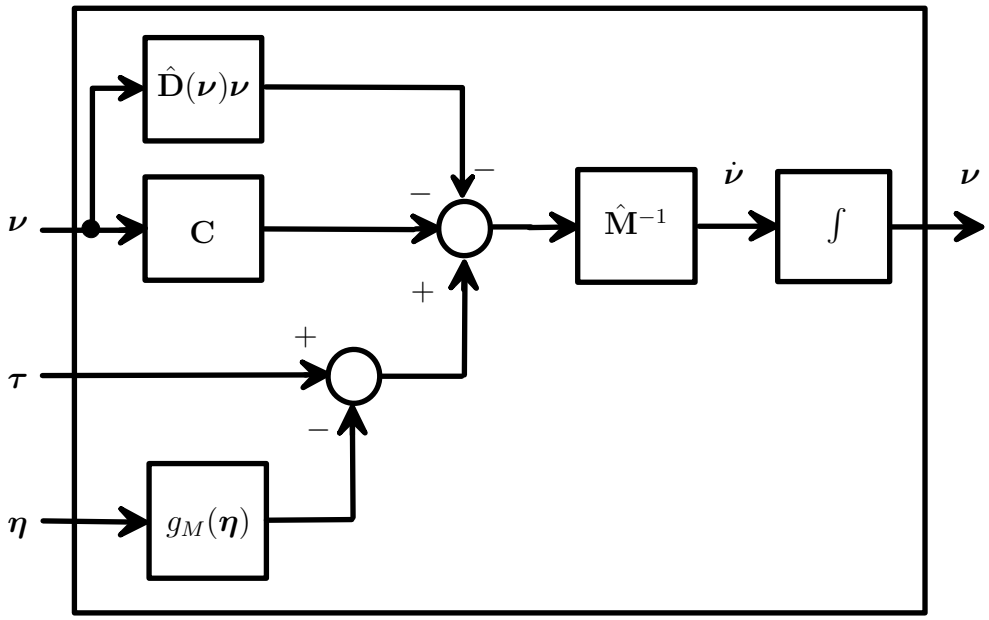


Figure 4.4: State space representation of the proposed identification model

4.3.3 Identification of the mass matrix

Although the mass matrix is constant, it is difficult to obtain the values of its elements straight from measurements. One could take measurements for low speed as then all factors incorporating $\boldsymbol{\nu}$ can be neglected and equation 4.1 simply becomes:

$$\mathrm{M}\dot{\boldsymbol{\nu}} = \boldsymbol{\tau} - \mathrm{g}(\boldsymbol{\eta}) \tag{4.9}$$

In theory one can calculate the mass matrix, M from equation 4.9. However in practice this poses severe problems, as a high acceleration, low velocity regime is seldom encountered.

Neural networks offer an elegant way to use equation 4.9 indirectly. To identify the mass matrix a neural network is trained to represent the full dynamics of the craft (as expressed in 4.1). Its inputs are $\boldsymbol{\tau}$, $\mathrm{g}(\boldsymbol{\eta})$ and $\boldsymbol{\nu}$ from which it will calculate $\dot{\boldsymbol{\nu}}$. When a satisfactory representation

of equation 4.1 has been achieved, the mass matrix is distilled by calculating the response of the neural network for a series of control inputs, $\boldsymbol{\tau}$, while the velocity inputs, $\boldsymbol{\nu}$, are kept close to zero. A convenient value for $\boldsymbol{\nu}$ is its mean value if this is close to zero. The neural network will thus yield the product of $\mathbf{M}^{-1} \cdot (\boldsymbol{\tau} - \mathbf{g}(\boldsymbol{\eta}))$ while the contribution of both $\boldsymbol{C}(\boldsymbol{\nu}) + \boldsymbol{D}(\boldsymbol{\nu})$ is negligible. As $\boldsymbol{\tau}$ and $\mathbf{g}(\boldsymbol{\eta})$ are known, $\mathbf{M}$ can be calculated using a standard least squares fitting algorithm to identify $\mathbf{M}^{-1}$ and invert this matrix consecutively.

This approach cannot be followed with all neural networks. By fixing the velocity inputs of the neural networks with a small value, the influence of the velocity is indeed negligible. However, it is not guaranteed that the data points thus created are in any way close to data points used during training of the neural networks. These new points will only result in realistic predictions if the training data set has enough samples in the region of interest (low or zero values for the velocity combined with a rich variety in values for the control forces) *and* if interpolation behaviour of the neural network is smooth. This is true for a back propagation neural network properly trained with sufficient data.

Care should be taken that independent measurement data is obtained for all six degrees of freedom. If a degree of freedom is not, or is insufficiently, excited the corresponding column in the mass matrix will consist of zeros or the estimates will be inaccurate. It is thus of paramount importance to induce movement in all degrees of freedom.

4.3.4 Identification of the damping

Now only the damping matrix $\mathbf{D}(\boldsymbol{\nu})$ is unknown and hence, initially, not accounted for, the dynamic equation for the reference system or process plant model (ppm) and the initial approximate

or control plant model (cpm) respectively can be written as:

$$\begin{aligned} \dot{\boldsymbol{\nu}}_{ppm} &= \mathbf{M}^{-1}[\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\nu}_{ppm})\boldsymbol{\nu}_{ppm} - \\ &\quad \mathbf{D}(\boldsymbol{\nu}_{ppm})\boldsymbol{\nu}_{ppm} - \mathbf{g}(\boldsymbol{\eta}_{ppm})], \qquad (4.10) \\ \dot{\boldsymbol{\nu}}_{cpm} &= \hat{\mathbf{M}}^{-1}\left[\boldsymbol{\tau} - \hat{\mathbf{C}}(\boldsymbol{\nu}_{cpm})\boldsymbol{\nu}_{cpm} - \mathbf{g}(\boldsymbol{\eta}_{cpm})\right]. \qquad (4.11) \end{aligned}$$

With the ppm, a simulation is performed in which the craft is controlled by a human operator through direct thruster commands. During the simulation one-step-ahead predictions are computed from the cpm. The influence of the damping matrix can then be computed as follows. At every time step one assumes that both systems have the same state vector: $\boldsymbol{\nu}_{ppm} = \boldsymbol{\nu}_{cpm} = \boldsymbol{\nu}$, and that the approximations of the mass and Coriolis matrix are ideal: $\hat{\mathbf{M}} = \mathbf{M}$ and $\hat{\mathbf{C}} = \mathbf{C}$. Substituting $\boldsymbol{\nu}$ for $\boldsymbol{\nu}_{ppm}$ and $\boldsymbol{\nu}_{cpm}$ in equations 4.10 and 4.11, the difference between equations 4.10 and 4.11 becomes the damping matrix $\boldsymbol{D}(\boldsymbol{\nu})$, multiplied by the state vector $\boldsymbol{\nu}$ and the inverse of the mass matrix, $\boldsymbol{M}^{-1}$. The product $\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}$ can thus be calculated as:

$$\mathbf{D}(\boldsymbol{\nu})\,\boldsymbol{\nu} = \mathbf{M}\left[\dot{\boldsymbol{\nu}}_{cpm} - \dot{\boldsymbol{\nu}}_{ppm}\right]. \qquad (4.12)$$

Equation (4.12) will be used for training of the neural networks and hence is the identification model.

4.3.5 Training of neural network models with multiple outputs

To prevent problems during the learning stage, rather than using one neural network to represent the mass and damping, six neural networks are used. It should be noted that the six elements of the vectors $\mathbf{M}\dot{\boldsymbol{\nu}}$ and $\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}$ do not have to be of the same order of magnitude. Especially

the elements pertaining to degrees of freedom that cannot be actuated will in general have a small magnitude. As back propagation learning relies on feeding back an error signal that will generally be larger for larger output values, the speed of learning can drastically change from output element to output element. As a result, parts of the neural network will learn faster than other parts. This again might result in overtraining of certain parts (*i.e.* the neural network might learn to represent the noise in the training data rather than the actual information). On the other hand, parts of the neural network will be 'undertrained' again resulting in poor performance. For offline training the above described problem can be circumvented by normalising the data. However, for online training there is no guarantee that this normalisation still holds.

4.3.6 Identification of underactuated craft

Most marine craft, both surface vessels and underwater craft, are underactuated. This means that it is impossible to excite all degrees of freedom through thruster, or control surface actuation. Even though this is the case, for underwater craft it is often possible to obtain movement in all degrees of freedom due to cross coupling or through operator manoeuvers. An example of cross coupling is the induction of a pitch moment due to a change in surge. [11]

The same issue holds for the measurements of the induced forces when an underwater craft is underactuated. In these cases measurements of the restoring forces, induced due to gravity and buoyancy forces, play an important role. By manoeuvering the underwater vehicle into certain attitudes a zero force in a certain degree of freedom can be prevented due to the restoring forces now having a component in this degree of freedom.

To illustrate this, one can think of an underwater vehicle that cannot be actuated in sway

[11] This can, for example, be seen in speedboats that lift their bow when the forward speed is increased. An example of operator manoeuvers to excite uncontrollable degrees of freedom is to turn to port or starboard while travelling in the forward direction. A velocity in sideways direction (in this case sway is assumed to be uncontrollable) is initiated.

direction. By inducing a roll moment, the sway components of the underwater vehicle's restoring forces will differ from zero, thus inducing a non-zero 'control' input in sway, which would have been zero otherwise.

It should be mentioned that, for this strategy to be applicable, the underwater craft should not be neutrally buoyant. In case the underwater craft is neutrally buoyant it becomes impossible to use gravitation to induce forces in surge, sway and heave direction and as a result the identification cannot be performed for all six degrees of freedom.

However, the model can be rewritten to exclude those degrees of freedom that are not excitable. Assume for example that forces can only be induced in the surge direction, and moments can be induced in roll, pitch and yaw (such manoeuvrability resembles a large class of torpedo shaped rovs). From a control point of view the uncontrollable degrees of freedom (sway and heave) need not be identified and the model can thus be written in terms of the four controllable degrees of freedom.

4.4 Conclusions

In this chapter a novel approach to the identification of model identification of marine craft, making use of neural networks, is presented.

Initially neural networks are trained on a batch of data to model the craft's acceleration resulting from current craft velocity and input forces to the craft. Consecutively distilling data from these neural networks for low velocities, will render the Coriolis and damping matrices negligible, and will thus yield the mass matrix.

With the identified mass matrix, training data to identify the craft's damping parameters, can be prepared and neural networks can be trained to represent the damping as an arbitrary, real

function of the craft's velocity.

Before investigating the proposed identification strategy through simulations and experiments, some conclusions can be drawn.

4.4.1 Dataset

First of all it should be stated that full and extensive actuation of the craft is of prime importance in collecting a training data set. As stated, the damping is a function of the velocity and for a good representation, the neural networks will thus have to be presented with sufficient data samples throughout the velocity range of interest. A second reason for the need of an extensive data set, is that neural networks only generalise if the data set covers the whole region of interest. If little or no data is available in a certain region, a neural network will at best give a smooth interpolation between known points. For large gaps however, it is more likely that the neural network yields a random output (in the case of back propagation neural networks) or an output close to zero (in the case of RBF networks). Of course, neither output is desirable, although a random output is potentially more harmful.

4.4.2 Signal processing

In the proposed identification strategy it is assumed that accelerations of the craft are available. This is typically true for state of the art underwater vehicles that are normally equipped with an inertial measurement unit. However, for low cost underwater vehicles and for surface vessels this is not necessarily true. In these cases the accelerations will have to be obtained from velocity, or even position measurements. The single or, respectively, double differentiation necessary to obtain the acceleration information is potentially a great source of noise. Filtering of the data is then necessary to obtain smooth signals, which can then be differentiated.

4.4.3 Online training

True online training of the neural networks has not been discussed in this work. Although the algorithms for online training are available, no algorithms have been developed or tested to ensure the convergence of online training algorithms. As is generally known, it can be shown that a neural network can approximate any real smooth function on a closed set. However, so far it has not been shown that a training algorithm will always converge to this state given certain boundary conditions. On a more positive note, the fact that the training performed in this work (using the Levenberg Marquardt algorithm) does not get stuck in insufficiently small local minima, offers good prospects for such proofs.

4.4.4 Possible use and benefits

The proposed identification strategy is expected to be beneficial for underwater vehicles for two reasons. First of all the necessary acceleration measurements are normally available from these platforms. And secondly, underwater vehicles are typically subjected to substantial changes in their dynamics due to, for example a change in toolskid. These considerable changes are furthermore accompanied by a need for precise dynamic model knowledge in cases of high precision manoeuvring demands. Application of the proposed identification strategy is expected to result in highly accurate models that will be adapted for changing circumstances. This is a great improvement over state of the art models that are often rough estimates and do not profit from online updates.

Application of the identification strategy to surface vessels is possible if the accelerations in the degrees of freedom of interest are significant. For an oil tanker, for example, accelerations can be very low. It may thus prove difficult to obtain these accelerations from possibly noisy

velocity data. For more agile craft, the use of position or velocity measurements only is feasible.

Chapter 5

Simulations with Tethra

In this chapter application of the identification strategy outlined in the previous chapter is demonstrated through simulations. Within the Mobile & Marine Robotics Group of the University of Limerick an open frame underwater vehicle, named Tethra, is currently being developed (Molnar et al., 2004). Using a model of this underwater vehicle, simulations are performed during which the model parameters are identified.

In section 5.1, formulas for the calculation of the inertia of Tethra, composed of inertias of its simple elements, are presented. Simple elements are structures such as cylinders, spheres and cubes, of which the inertia can easily be calculated.

In section 5.2 these formulas are applied to a structural model of Tethra to obtain a mathematical model of Tethra's dynamics for simulation purposes. Then, in section 5.3 the derived model is used in a simulation study. With the model, data is gathered that is then used to train several neural networks to identify the mass matrix and to identify and represent the damping parameters. The identified parameters are then used in a feedforward controller to test the accuracy of the identification process. The feedforward controller is used to determine the correct input forces and moments in order to follow a predetermined trajectory. The resulting trajectory is then compared to the desired trajectory, yielding a quantitative measure of the accuracy of the identification.

5.1 Inertial properties of simple elements

The movement of an object in 6 degrees of freedom can be described using Newton's second law and its rotational analogue:

$$\begin{aligned} \mathbf{F} &= m\mathbf{a} \\ \mathbf{m} &= \mathbf{I}\boldsymbol{\omega}, \end{aligned} \tag{5.1}$$

where:

- $\mathbf{F}$ is the force working on the object
- m is the mass of the object
- $\mathbf{a}$ is the acceleration of the object resulting from the applied force
- $\mathbf{m}$ is the angular momentum of the object
- $\mathbf{I}$ is the inertia tensor
- $\boldsymbol{\omega}$ is the angular velocity of the object

The inertia tensor, $\mathbf{I}$, in a coordinate frame c is defined as:

$$\mathbf{I}_c = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}, \tag{5.2}$$

and describes the inertial properties of the object as an integral over the inertial properties of all particles in the object:

$$I_{xx} = \int_V (y^2 + z^2)\rho dV \quad I_{yy} = \int_V (x^2 + z^2)\rho dV \quad I_{zz} = \int_V (x^2 + y^2)\rho dV \tag{5.3}$$

$$I_{xy} = I_{yx} = -\int_V (xy)\rho dV \tag{5.4}$$

$$I_{xz} = I_{zx} = -\int_V (xz)\rho dV \tag{5.5}$$

$$I_{yz} = I_{zy} = -\int_V (yz)\rho dV \tag{5.6}$$

$$\tag{5.7}$$

where x, y and z denote the position of the infinitesimal volume element dV with mass ρ. The diagonal terms in equation 5.2 are called the moments of inertia and the off-diagonal elements are called the products of inertia. Choosing the coordinate frame to coincide with the principal axes, the products of inertia in equation 5.2 will vanish. If so required, a transformation to another coordinate frame can be made in two steps. First the inertia tensor is transformed to a coordinate frame with the same origin as the original frame but its axes parallel to the axes of the new reference frame (McKerrow, 1993):

$$\mathbf{I}_c^{rot} = \mathbf{T}\mathbf{I}_c\mathbf{T}^T \tag{5.8}$$

where $\mathbf{T}$ is the rotation matrix:

$$\mathbf{T} = \begin{bmatrix} \cos(\psi)\cos(\theta) & -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta) \\ \sin(\psi)\cos(\theta) & -\cos(\psi)\cos(\phi) + \sin(\psi)\sin(\theta)\sin(\phi) & \cos(\psi)\cos(\phi) + \sin(\psi)\cos(\phi)\sin(\theta) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix}. \tag{5.9}$$

Consecutively, the parallel axes theorem can be used to translate the rotated coordinate frame from the origin c to the origin o:

$$\mathbf{I}_o = \mathbf{I}_c^{rot} - m\mathbf{r} \times (\mathbf{r} \times \mathbf{I}^{3\times 3}), \tag{5.10}$$

where $\mathbf{r}$ denotes the euclidian distance between the points o and c. With equations 5.8 and 5.10, the inertia of several objects in different reference systems can be transformed to one common reference system. If these objects are interconnected rigidly to form (for example) an underwater vehicle, the underwater vehicle's total inertia can be calculated by taking the sum of all transformed inertias of the individual objects. For Tethra, the craft under investigation, this yields a relatively simple modelling strategy, as Tethra is predominantly made of simple objects, notably cylinders. Thus, by deriving the inertia tensor for a cylinder, and applying the result to the several cylindrical objects Tethra is made of, all the individual inertias can be found. Using 5.8 and 5.10 on all individual inertias and performing a sum over the results will yield the total inertia of the underwater craft. This process is detailed in section 5.2. To finalise this section, the inertia of a hollow cylinder will be derived.

Consider a hollow cylinder with outer radius R_o, inner radius R_i, height h and total mass M as depicted in figure 5.1. The total mass has contributions of the 'hollow' (possibly not hollow but with another mass density) inside of the cylinder with mass M_i and the wall of the cylinder

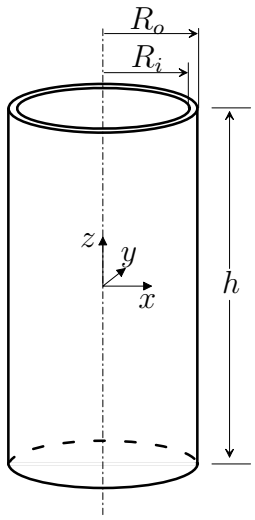


Figure 5.1: Hollow cylinder

with mass M_w. The local coordinate frame is chosen as depicted in figure 5.1, which, based on symmetry, should be in principal axes form. Now all six integrals from equation 5.3 need to be solved. Starting with the moments of inertia, the integrals are calculated in cylinder coordinates:

$$x = r\cos(\phi); \quad y = r\sin(\phi); \quad z = z$$
$$0 < r < R_o; \quad 0 < \phi < 2\pi \tag{5.11}$$

$$
\begin{aligned}
I_{xx} &= \int_V \rho(y^2+z^2)dV = \int_0^{R_o}\int_{-\frac{h}{2}}^{\frac{h}{2}}\int_0^{2\pi} \rho(r^2\sin^2(\phi)+z^2)r\,dr\,dz\,d\phi \\
&= \int_0^{R_o}\int_{-\frac{h}{2}}^{\frac{h}{2}} \rho(\pi r^2+2\pi z^2)r\,dr\,dz = \int_0^{R_o} \rho(\pi r^2 h+\frac{\pi h^3}{6})r\,dr \\
&= \rho\left[\frac{\pi r^4 h}{4}+\frac{\pi h^3 r^2}{12}\right]_{r=0}^{r=R_o} \\
&= \frac{M_i}{\pi R_i^2 h}\left[\frac{\pi r^4 h}{4}+\frac{\pi h^3 r^2}{12}\right]_{r=0}^{r=R_i} + \frac{M_w}{\pi(R_o^2-R_i^2)h}\left[\frac{\pi r^4 h}{4}+\frac{\pi h^3 r^2}{12}\right]_{r=Ri}^{r=R_o} \\
&= \frac{M_i R_i^2}{4}+\frac{M_i h^2}{12}+\frac{M_w(R_o^2-R_i^2)}{4}+\frac{M_w h^2}{12}.
\end{aligned} \tag{5.12}
$$

For symmetry reasons I_{yy} is equal to I_{xx} and I_{zz} can be calculated as follows:

$$
\begin{aligned}
I_{zz} &= \int_V \rho(x^2+y^2)dV = \int_0^{R_o}\int_{-\frac{h}{2}}^{\frac{h}{2}}\int_0^{2\pi} \rho r^3\,dr\,dz\,d\phi \\
&= 2\pi h\int_0^{R_o}\rho r^3 dr = \left[2\pi h\frac{M_i}{\pi R_i^2 h}\frac{r^4}{4}\right]_{r=0}^{r=R_i} + \left[2\pi h\frac{M_w}{\pi(R_o^2-R_i^2)h}\frac{r^4}{4}\right]_{r=R_i}^{r=R_o} \\
&= \frac{M_i R_i^2}{2}+\frac{M_w(R_o^2-R_i^2)}{2}.
\end{aligned} \tag{5.13}
$$

Explicit calculation of the forces of inertia has been omitted as these integrals lead to terms $\int \sin(\phi)d\phi$, $\int \cos(\phi)d\phi$ and $\int \sin(\phi)\cos(\phi)d\phi$, which, for integration from 0 to 2π all yield zero. The assumption that by choosing a coordinate frame leading to the highest degree of symmetry, the principal axes were found, is thus true.

5.2 Modelling Tethra

This section details the modelling of the University of Limerick in-house constructed underwater robot, Tethra[12]. In 2000 a project was started with the aim of realising a test bed for autonomous control in an underwater environment. For this purpose it was decided that an open frame structure with two dry hulls should be constructed. The open frame structure has the benefit of offering an easy way of installing new equipment. Figure 5.2 shows a picture of Tethra.

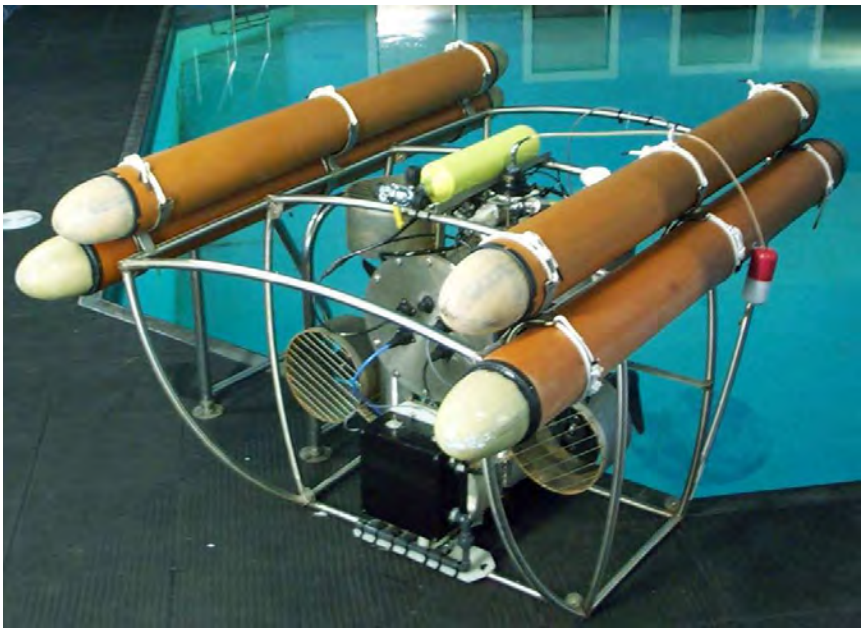


Figure 5.2: The University of Limerick in-house underwater robot Tethra on the side of the test pool

The bluff body of Tethra results in relatively high damping coefficients as well as high added mass coefficients. As software, such as WAMIT (2005), is not capable of calculating accurate values for damping and mass for bluff bodies, Tethra (and with Tethra most other hovering underwater robots) is the ideal craft for showing the benefits of identifying the craft dynamics using the method described in chapter 4 (page 67ff).

[12]Tethra is the name of a Fomorian King and god of the sea in Irish mythology

Table 5.1: Mechanical properties of materials used

Element	Floats	Floats	Lower Hull	Upper Hull	Frame	Endplate	Thruster
L [m]	1.98	1.75	0.61	0.61	0.46-1.56	$5 \cdot 10^{-3}$	0.30
radius [m]	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	0.20	0.20	$1.25 \cdot 10^{-2}$	0.395	$5 \cdot 10^{-2}$
Wall thickness [m]	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$4 \cdot 10^{-3}$	$4 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	-	-
M_i [kg]	3	2.7	20	15	-	-	10
M_w [kg]	7	6.2	20	20	-	-	-
Material	PVC, Marine foam	PVC, Marine foam	Stainless steel	Stainless steel	Stainless Steel	Stainless Steel	Various

In this section the dynamics governing Tethra will be analysed and an approximate model derived. First a representation of Tethra in terms of simple elements is given. Then the construction of Tethra in terms of simple elements will be derived.

5.2.1 Mechanical properties of Tethra

As can be seen from figure 5.2, Tethra mainly consists of cylindrical elements; a frame of stainless steel piping is constructed around two stainless steel cylindrical hulls. Neutral buoyancy is obtained through four cylindrical floats mounted at the top on the port and starboard side of the craft. The elements used, and their estimated characteristics are listed in table 5.1.

Using the equations derived in section 5.1 and the information regarding the position (x,y,z) and attitude (ϕ, θ, ψ) of the various elements, as given in table 5.2, the inertia tensor for the whole craft with respect to the craft's centre of gravity, $\mathbf{I}_t$, can be calculated:

$$\mathbf{I}_t = \begin{bmatrix} 51.6798 & -0.0816 & 0.5526 \\ -0.0816 & 49.7434 & 0 \\ 0.5526 & 0 & 52.6979 \end{bmatrix}. \tag{5.14}$$

Table 5.2 List of simple elements and their characteristics

Element	Length	x [m]	y [m]	z [m]	$\phi\,[^\circ]$	$\theta\,[^\circ]$	$\psi\,[^\circ]$
Frame	0.46	.25	-.502	-.228	0	90	-8
	0.46	-.25	-.502	-.228	0	90	-8
	0.46	.25	.502	-.228	0	90	8
	0.46	-.25	.502	-.228	0	90	8
Frame	0.48	0	0	-.978	0	0	0
	0.48	.25	-.516	-.695	0	90	4
	0.48	-.25	-.516	-.695	0	90	4
	0.48	.25	.516	-.695	0	90	-4
	0.48	-.25	.516	-.695	0	90	-4
	0.48	-.6475	.491	-.701	0	119.6	4
	0.48	-.6475	-.491	-.701	0	119.6	-4
	0.48	.6475	.491	-.701	0	60.4	4
	0.48	.6475	-.491	-.701	0	60.4	-4

Continued on next page

Table 5.2 List of simple elements and their characteristics

Element	Length	x [m]	y [m]	z [m]	ϕ [°]	θ [°]	ψ [°]
Frame	0.50	.78	-.249	-.956	-5	0	90
	0.50	.78	.249	-.956	5	0	90
	0.50	.25	-.249	-.956	-5	0	90
	0.50	.25	.249	-.956	5	0	90
	0.50	-.25	-.249	-.956	-5	0	90
	0.50	-.25	.249	-.956	5	0	90
	0.50	-.78	-.249	-.956	-5	0	90
	0.50	-.78	.249	-.956	5	0	90
	0.50	0	-.47	0	0	0	0
	0.50	0	0	0	0	0	0
	0.50	0	.47	0	0	0	0
Frame	0.60	-.383	.477	-.2335	0	119.6	4
	0.60	-.383	-.477	-.2335	0	119.6	-4
	0.60	.383	.477	-.2335	0	60.4	4
	0.60	.383	-.477	-.2335	0	60.4	-4
Frame	0.94	.25	0	0	0	0	90
	0.94	-.25	0	0	0	0	90
Frame	1.56	0	-.498	-.934	0	0	0
	1.56	0	.498	-.934	0	0	0
Float	1.98	0	.578	-.934	0	0	0
	1.98	0	-.578	-.934	0	0	0

Continued on next page

Table 5.2 List of simple elements and their characteristics

Element	Length	x [m]	y [m]	z [m]	$\phi\,[^\circ]$	$\theta\,[^\circ]$	$\psi\,[^\circ]$
Float	1.75	0	.498	-1.084	0	0	0
	1.75	0	-.498	-1.084	0	0	0
Hull	0.61	0	0	-.692	0	0	0
	0.61	0	0	-.231	0	0	0
Thruster	0.30	0	.35	-.430	0	0	0
	0.30	0	-.35	-.430	0	0	0
	0.30	0	.28	-.730	0	90	0
	0.30	0	-.28	-.730	0	90	0
Endplate	-	.25	0	-.692	0	0	0
	-	-.25	0	-.692	0	0	0
	-	.25	0	-.231	0	0	0
	-	-.25	0	-.231	0	0	0

5.2.2 A simple model of Tethra

With the data presented in section 5.2.1, a simple model of Tethra can be constructed. Figure 5.3 shows this model, which is composed of the simple elements listed in table 5.2.

Rigid body mass and added mass parameters

The rigid body mass matrix, $\mathbf{M}_{RB}$, with respect to the centre of the body fixed coordinate system, $\mathbf{o}$, can be calculated from the inertia tensor from equation 5.14 and the position of the centre of gravity with respect to $\mathbf{o}$, $\mathbf{r}_g$:

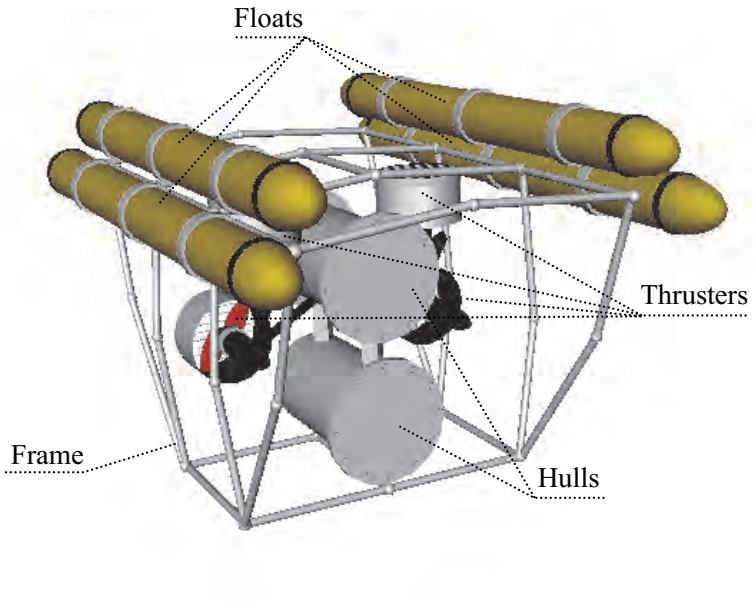


Figure 5.3: Model of Tethra composed of simple elements

$$\mathbf{r}_g = [-0.0052\ 0\ \ -0.2269]^T. \tag{5.15}$$

Now the rigid body mass matrix can be calculated using:

$$\mathbf{M}_{RB} = \begin{bmatrix} m \cdot \mathbf{I}^{3\times 3} & -m \cdot \mathbf{S}(\mathbf{r}_g) \\ m \cdot \mathbf{S}(\mathbf{r}_g) & \mathbf{I}_t \end{bmatrix}, \tag{5.16}$$

where the notation used in Fossen (2002) was adopted and the operator $\mathbf{S}$ is defined as:

$$\mathbf{S}(\mathbf{a})\mathbf{b} = \mathbf{a} \times \mathbf{b}. \tag{5.17}$$

Using equations 5.14 and 5.15 in equation 5.16, yields[13]:

[13]The negligible off diagonal elements in equation 5.14 and the x component of equation 5.15 were omitted in this calculation.

$$\mathbf{M}_{RB} = \begin{bmatrix} 231 & 0 & 0 & 0 & -52 & 0 \\ 0 & 231 & 0 & 52 & 0 & 0 \\ 0 & 0 & 231 & 0 & 0 & 0 \\ 0 & 52 & 0 & 52 & 0 & 0 \\ -52 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 53 \end{bmatrix}. \tag{5.18}$$

Calculation of the added mass contributions is more complicated. Especially as the craft has a bluff shape and consists of several cylindrical elements in close proximity to each other, calculation of the added mass matrix is fairly complicated. As the intention in this thesis is not to obtain a highly accurate model based on analytical and numerical methods, the added mass contribution is simply neglected. This does not have any effect on the proposed identification method.

Restoring forces

As shown in section 4.3.1, the restoring forces can be expressed as:

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} (W-B)\sin(\theta) \\ -(W-B)\cos(\theta)\sin(\phi) \\ -(W-B)\cos(\theta)\cos(\phi) \\ -(y_g W - y_b B)\cos(\theta)\cos(\phi) + (z_g W - z_b B)\cos(\theta)\sin(\phi) \\ (z_g W - z_b B)\sin(\theta) + (x_g W - x_b B)\cos(\theta)\cos(\phi) \\ -(x_g W - x_b B)\cos(\theta)\sin(\phi) - (y_b W - y_b B)\sin(\theta) \end{bmatrix}. \tag{5.19}$$

The values for W and B can be found from tables 5.1 and 5.2 by calculating the mass and volume of each simple element. If the total mass is m, then:

$$W = mg, \tag{5.20}$$

where g is the gravitational constant. The upward forces due to the crafts volume, V, are:

$$B = \rho V g, \tag{5.21}$$

where ρ is the density of the water. The position of the centre of buoyancy and the centre of gravity can be calculated by taking a 'weighted' average of the positions of the simple elements the craft consists of. Both the mass and the buoyancy forces of each element can be modelled as a point force in the centre of the element. If this point force, whether it be from the negative buoyancy forces or from the positive gravity forces[14], is indicated with $\mathbf{F}_i$ for element i, then the centre of gravity and the centre of buoyancy have to obey the following equation:

$$\sum_i \mathbf{F}_i(\mathbf{o}_i - \mathbf{o}_x) = 0, \tag{5.22}$$

where $\mathbf{o}_x$ is the centre of buoyancy or gravity and $\mathbf{o}_i$ is the position of the point force. From equation 5.22, $\mathbf{o}_x$ can be found.

$$\mathbf{o}_x = \frac{\sum_i \mathbf{F}_i \mathbf{o}_i}{\sum_i \mathbf{F}_i}. \tag{5.23}$$

With the angles ϕ, θ and ψ measurable, the restoring forces can now be calculated.

[14] In the body-fixed frame the positive z-axis is pointing down

Damping

The estimation of the damping through analytical methods or numerical methods is difficult and imprecise for bluff bodies. And as identification of the damping through such methods is not the goal of this work, realistic values for the damping have been chosen based on observations during experiments. As noted in chapter 3, the linear part of the damping has the same form as the mass matrix. Oftentimes the quadratic damping is chosen to have a diagonal form. With these boundary conditions the linear and quadratic damping matrices, respectively indicated with D_l and D_q, were chosen such that a realistic behaviour was obtained when comparing simulations to pool trials:

$$\begin{aligned} \mathbf{D_l} &= \begin{bmatrix} 116 & 0 & 0 & 0 & -15 & 0 \\ 0 & 116 & 0 & 15 & 0 & 0 \\ 0 & 0 & 116 & 0 & 0 & 0 \\ 0 & 15 & 0 & 26 & 0 & 0 \\ -15 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26 \end{bmatrix} \\ \mathbf{D_q} &= \text{diag}(100\,100\,100\,30\,30\,30) \cdot |\boldsymbol{\nu}|. \end{aligned} \tag{5.24}$$

5.3 A simulation study

With the model described in the previous sections, simulations were done to perform an identification of the mass matrix and the damping parameters. Consecutively the identified parameters were tested in a feedforward controller.

Initially data was collected for training of the neural networks. As discussed in Nelles (2001)[Sec. 17.7], a mature theory for the choice of excitation signals when collecting data for nonlinear identification experiments has yet to be developed. To ascertain that all degrees of freedom are excited, a human operator was used. This scenario was simulated in Matlab using the virtual-reality toolbox to make a realistic representation of a real-world situation. A human operator can control the vehicle in this virtual world during which data is recorded. Once enough data has been collected to perform identification, the parameter in question is estimated. Enough data is obtained when the regions of operation in all degrees of freedom have been visited. To achieve this, the human operator has to ensure that all (coupled) manoeuvres are performed at least once for several different velocities. No tools for data set evaluation, other than testing the data set simply through training the neural networks and evaluating the obtained identification, were used. Development of such tools would be highly recommendable when automating the identification process.

As discussed in chapter 4 a certain order is used for identification:

1. First a normal matrix representation of the mass matrix is determined through the use of neural networks.

2. Then the Coriolis matrix, $\mathbf{C}$ can be determined from the estimate of the mass matrix, $\mathbf{M}$.

3. Using the estimates of $\mathbf{M}$ and $\mathbf{C}$, the influence of the damping matrix, $\mathbf{D}$ can be determined

as described in section 4.3.4.

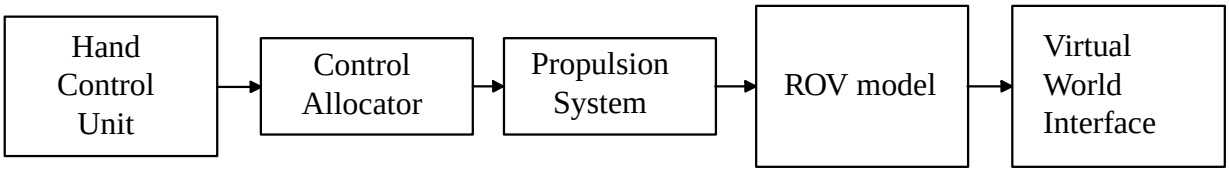


Figure 5.4: Block diagram of the simulation environment in Matlab

Figure 5.4 gives an impression of the virtual-world interface made within the Mobile & Marine Robotics Group in the University of Limerick. Joystick commands are interpreted in the block called 'Hand Control Unit'. These joystick commands, that represent excitations in four degrees of freedom are then converted to thruster commands in the block 'Control Allocator'. Time delays and saturation of the thrusters are described by the block 'Propulsion System' and the vehicle dynamics are described in the block 'ROV model'.

In the Matlab virtual world environment the University of Limerick test pool has been drawn to obtain similar circumstances in simulation and future testing. Figure 5.5 shows the vehicle in the pool at the beginning of a simulation.

During a simulation with a length of 1000 seconds, in which the craft was flown by a human operator, enough information was gathered to ensure sufficient independence of the collected data. A sampling rate of 10 Hz was used. After data collection the first neural network training stage commenced. In this stage, 6 neural networks with 12 input neurons, 5 neurons in one hidden layer with hyperbolic tangent activation function and 1 linear output neuron were used. The input to these neural networks is $[\boldsymbol{\nu}^T \ (\boldsymbol{\tau} - \mathbf{g}(\boldsymbol{\eta}))^T]^T$. Each neural network outputs one element of the vector $\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}[\boldsymbol{\tau} - \mathbf{g}(\boldsymbol{\eta}) - (\mathbf{C}(\boldsymbol{\nu}) + \mathbf{D}(\boldsymbol{\nu}))\boldsymbol{\nu}]$. After 50 epochs of training with the Levenberg-Marquardt algorithm, the process was stopped and extraction of the mass matrix performed. All values for $\boldsymbol{\tau}$ used in the training process were fed into the neural network one by one,

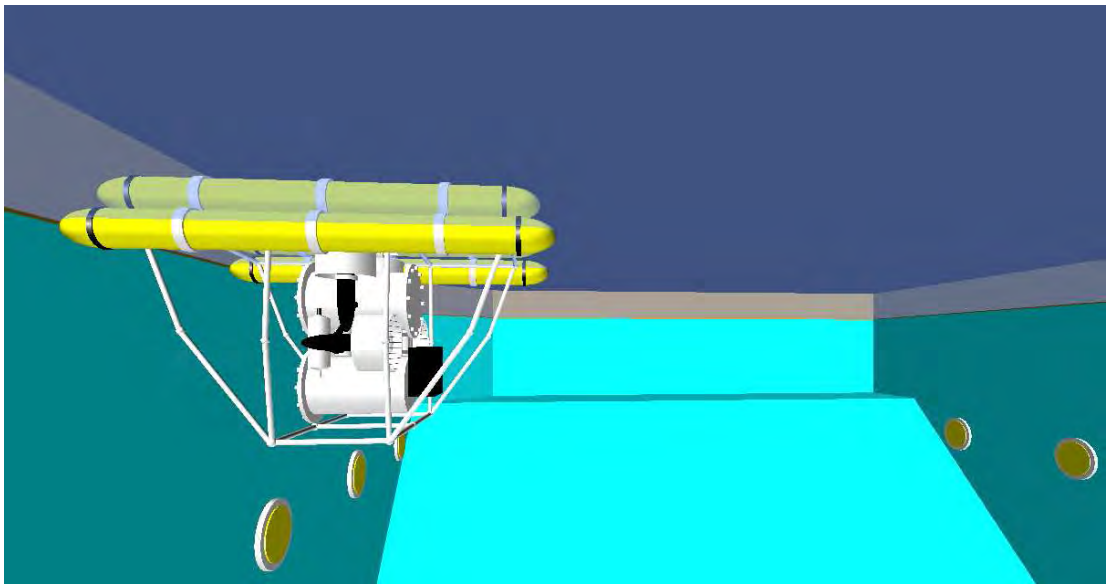


Figure 5.5: The University of Limerick in-house vehicle in the virtual pool at the start of a simulation

while $\boldsymbol{\nu}$ was kept constant at its mean value of $[0.10; 0.016; 0.016; -0.0010; -0.00020; 0.050]$[15]. This is sufficiently small not to be of any influence. From the collected data, which represents $\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}(\boldsymbol{\tau} - \mathrm{g}(\boldsymbol{\eta}))$, $\mathbf{M}^{-1}$ was calculated using a least squares fitting algorithm. Finally matrix inversion of $\mathbf{M}^{-1}$ yields $\mathbf{M}$.

Using the same data, identification of the damping matrix was started using the identified approximation of the mass matrix. Again proper identification will highly benefit from a complete data set. Complete in this sense means that, again, all degrees of freedom must be excited. But, as damping is a non-linear function of velocity, it is also important that training data is acquired for the full range of operation velocities. As for identification of the mass matrix, 6 different networks are used to approximate the six components of $\mathbf{D} \cdot \boldsymbol{\nu}$. The 6 networks, with 12 inputs, 5 neurons in a hidden layer using a hyperbolic tangent activation function and 1 linear output neuron, were trained with the Levenberg-Marquardt training algorithm for 50 cycles. The input

[15]For clarity a semicolon is used for row separation.

to all networks is $[\boldsymbol{\nu}^T \ |\boldsymbol{\nu}|^T]^T$, where the superscript T indicates the vector transpose and again each network approximates one element of the vector $\mathbf{D} \cdot \boldsymbol{\nu}$.

5.3.1 Simulation results

The identified mass matrix is shown in equation 5.25. Comparison with equation 5.18 shows that the identification is accurate. The maximum relative error made in the estimation of the non-zero elements in the mass matrix is 10%, which is generally acceptable for control purposes. Most elements that are zero in the real mass matrix (equation 5.18), are non-zero in the mass matrix identified using neural networks. However, their magnitude is less than 7% of the significant parameters in the same row of the mass matrix[16].

$$\hat{\mathbf{M}} = \begin{bmatrix} 255 & 2 & -3 & 2 & -53 & 0 \\ 0 & 222 & 0 & 47 & 0 & -1 \\ 15 & 1 & 236 & 1 & -14 & 0 \\ 0 & 52 & 0 & 51 & 0 & 1 \\ -54 & -2 & 3 & -2 & 49 & 0 \\ 0 & 1 & 0 & 2 & 0 & 53 \end{bmatrix} \tag{5.25}$$

Case Study I: Drag estimation under ideal conditions

As damping is generally a non-linear function of the velocity of which the exact form is unknown, verification of the identified damping is performed through simulations. In these simulations the underwater vehicle is switched to 'auto pilot'. In this mode a path planner generates way points which can be translated into desired accelerations for the controller. Equation 4.1 shows that,

[16]The error is related to the maximum parameter in the same row as each row of the mass matrix multiplied by the velocity, $\dot{\boldsymbol{\nu}}$, constitutes one of the elements of the force and moments vector working on the craft

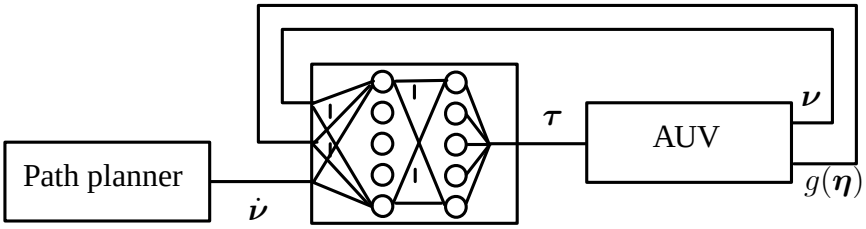


Figure 5.6: Block schematic of craft controlled by feedforward controller

given $\boldsymbol{\nu}$, $\boldsymbol{\eta}$, $\mathrm{g}(\boldsymbol{\eta})$, and the model parameters $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{D}$, the necessary control input, $\boldsymbol{\tau}$, to yield an acceleration, $\dot{\boldsymbol{\nu}}$, can be computed.

$$\boldsymbol{\tau} = \hat{\mathbf{M}}\dot{\boldsymbol{\nu}}_{cpm} + \hat{\mathbf{C}}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) + \hat{\mathbf{D}}(\boldsymbol{\nu})\boldsymbol{\nu}. \tag{5.26}$$

Thus, using equation 5.26 as the control law, a feedforward controller can be made, with the desired control force as its output. Figure 5.6 depicts the resulting architecture.

The accuracy of the NN damping approximation can now be tested by comparing the resulting paths for the ppm and the cpm with the approximated damping. In order to test for coupling between the various degrees of freedom, the auto pilot dictates a spiralling path from a depth of 0.3 m to a depth of 2 m and back up[17] as depicted in figure 5.7.

Figure 5.8 shows the path followed by the feedforward controller with exact knowledge of the craft dynamics (solid line). That is, in the controller the true values of $\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{D}$ and $\boldsymbol{g}(\boldsymbol{\eta})$ are used. To obtain an idea of the influence of errors in the estimation of these parameters, a simple error model is used; the dotted path illustrates the behaviour of the controller if all parameters in the estimates of $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ are increased by 10%.

The plots show that the craft does not follow a perfect circle from the beginning. This is due to the fact, that in the feedforward controller no information is used regarding the non-linear

[17] Limited by the depth of the pool

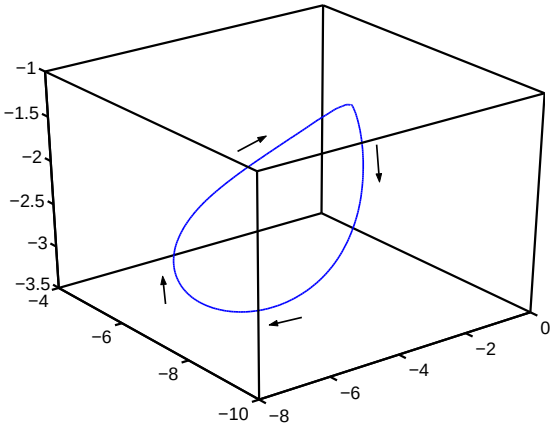


Figure 5.7: Dive profile from 1.5 m to 3.5 m and back up while following circular path in x-y plane

dynamics of the thruster system.

The neural network model performance is shown in figure 5.9 in which the measurement data is assumed to be noise free. The graph shows that the dynamics have been identified satisfactorily. The root mean square tracking error of the 3D position of the craft, defined as:

$$\mathbf{E}^{tr}_{rms} = \sqrt{\sum_{i}^{N} \frac{(\mathbf{p}_{i_{ppm}} - \mathbf{p}_{i_{cpm}})^2}{N}}, \tag{5.27}$$

where $\mathbf{p}$ is the 3D position of the craft, is less than a millimeter in all three degrees of freedom. Also the maximum absolute tracking error, defined as:

$$\mathbf{E}^{tr}_{max} = \max |(\mathbf{p}_{i_{ppm}} - \mathbf{p}_{i_{cpm}})|, \tag{5.28}$$

is smaller than one millimeter in all three degrees of freedom. Of course one should realise that such an accuracy will only be obtained in case no disturbances, such as waves or currents, are present.

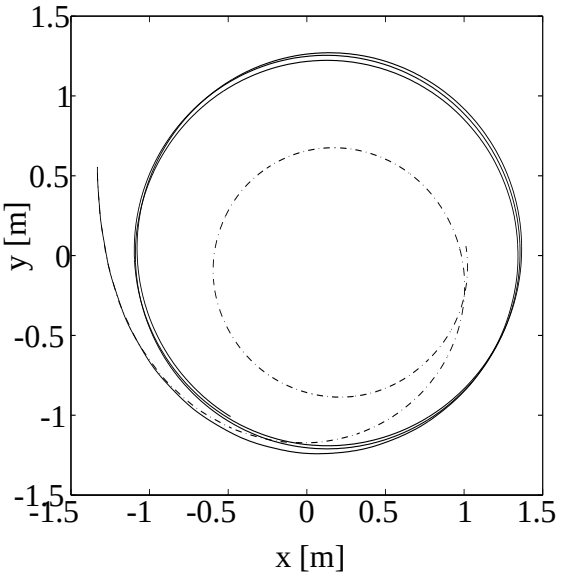


Figure 5.8: Path followed by ideal controller (solid line) and controller with 10% error in parameters (dotted line)

Case Study II: Drag estimation under noisy conditions

To investigate the ability of the neural network to perform estimation of the damping using noisy training signals, typical noise is added to the measurements. For the linear accelerations a zero mean noise sequence with an amplitude of $0.5\,ms^{-2}$ is added. The angular accelerations are summed with a noise component of $0.05\,rads^{-2}$. The velocity measurements are assumed to be impeded with a noise sequence, whose amplitude is equal to $0.1\,\%$ of the magnitude of the velocity. It should be noted that in the simulations no noise prefiltering is assumed. This would make matters considerably easier for the neural network. Performance of the neural networks is shown in figures 5.10a and 5.10b.

Clearly, the added noise affects the prediction abilities of the model. With a root mean square error as shown in table 5.3, the noise is of significant influence on the approximation accuracy of the neural networks.

The prediction of the right control force, $\boldsymbol{\tau}$, can be improved by continuing the learning

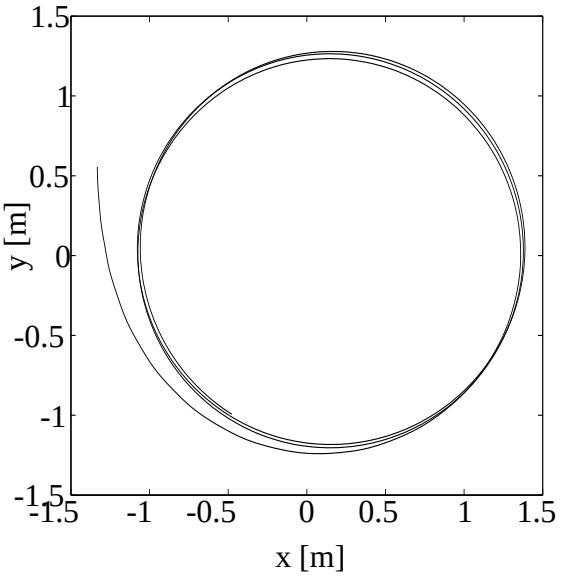


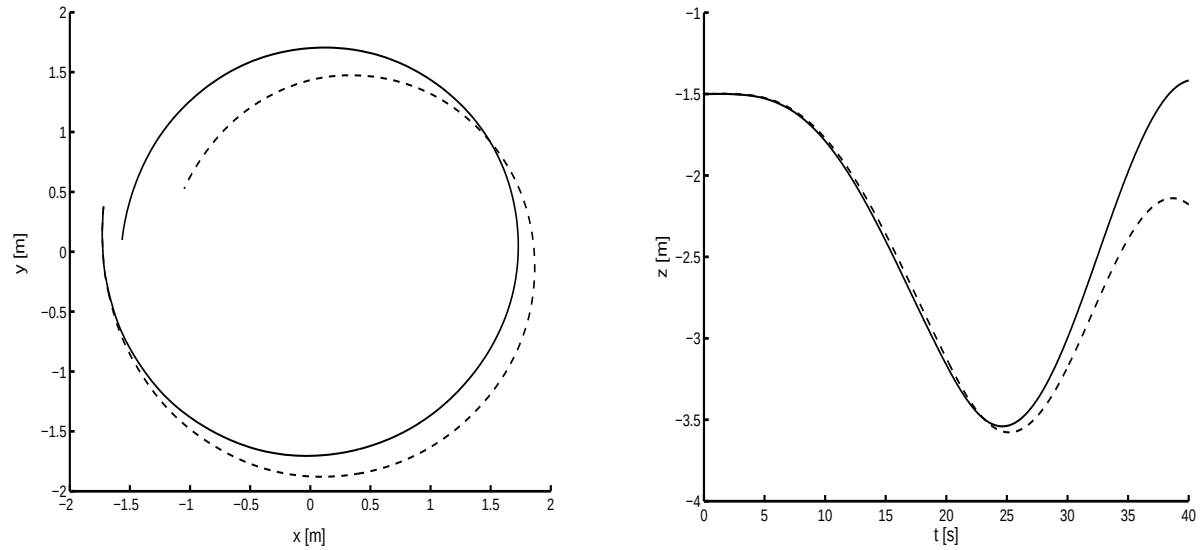
Figure 5.9: Path followed by neural network controller

Table 5.3: Tracking errors due to noise in the measurements

	x	y	z
$E^{tr}_{rms}[m]$	.29	.20	.24
E^{tr}_{max} [m]	.61	.42	.76

process online. The result of this experiment is shown in figures 5.11a and 5.11b, demonstrating that online learning improves the predictions considerably. Both the root mean square tracking error and the maximum absolute tracking error are smaller than 1 millimeter in all three degrees of freedom.

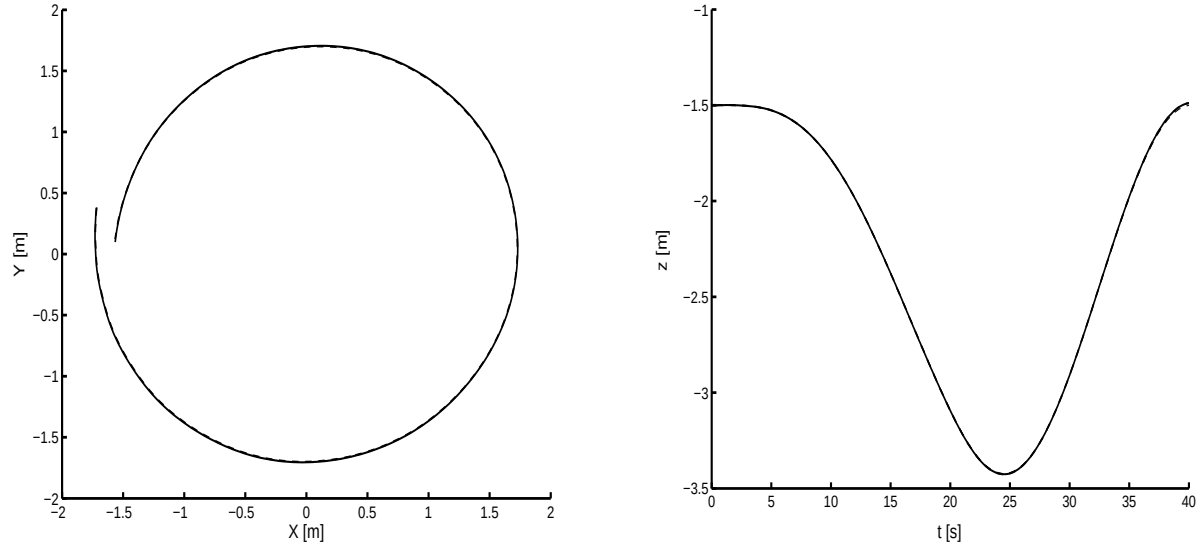
However, in this case the neural network does not necessarily represent the damping any more. Online learning results in a better local estimate of the damping as the neural network is now trained with larger weight on the latest data points. The global estimation accuracy may suffer from this, unless a considerable amount of the state space is visited and thus used in the online learning process.



(a) Horizontal projection

(b) Depth profile versus time

Figure 5.10: Path followed by vehicle with damping approximated from noisy measurements (ppm solid line, cpm dotted line)



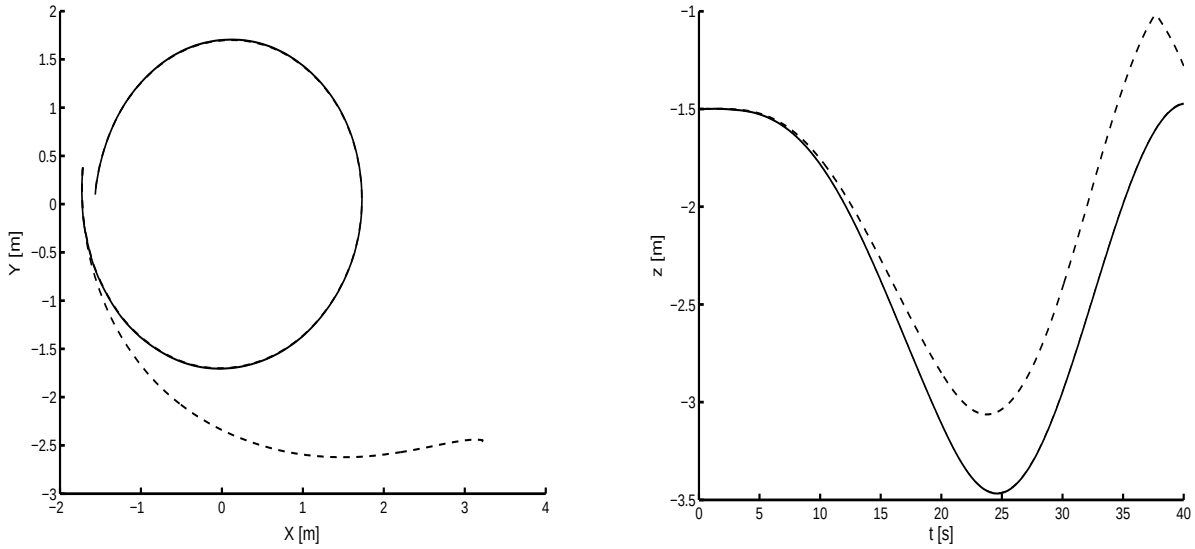
(a) Horizontal projection

(b) Depth profile versus time

Figure 5.11: Path followed by vehicle with damping approximated from noisy measurements and online updating (ppm solid line, cpm dotted line)

Case Study III: Drag estimation for time-varying dynamics

In this simulation the suitability of neural network based identification methods for changing toolskids is demonstrated. The AUV from Case Study II is equipped with a 60 kg heavy torpedo shaped sidescan sonar. As a result several matrices, which were previously assumed to be known, will change drastically. However, changes in the mass matrix, $\mathbf{M}$ (and thus the Coriolis matrix, $\mathbf{C}(\boldsymbol{\nu})$) and the restoring forces, $\mathbf{g}(\boldsymbol{\eta})$, are assumed to be known. Changes in the damping matrix will be assumed to be unknown and online learning is performed to adapt for these changes. Figure 5.12a and 5.12b show the trajectory for the process plant model and the control plant model using the neural network to model damping, but without online learning.



(a) Horizontal projection

(b) Depth profile versus time

Figure 5.12: Path followed by vehicle with damping approximated from noisy measurements and unknown change in damping (ppm solid line, cpm dotted line)

Clearly, the change in damping, which amounts to parameters changing up to a factor 10, has a detrimental effect on the performance of the feedforward controller. The root mean squared

tracking error and maximum absolute tracking error are shown in table 5.4.

Table 5.4: Tracking errors due to a change in the dynamics

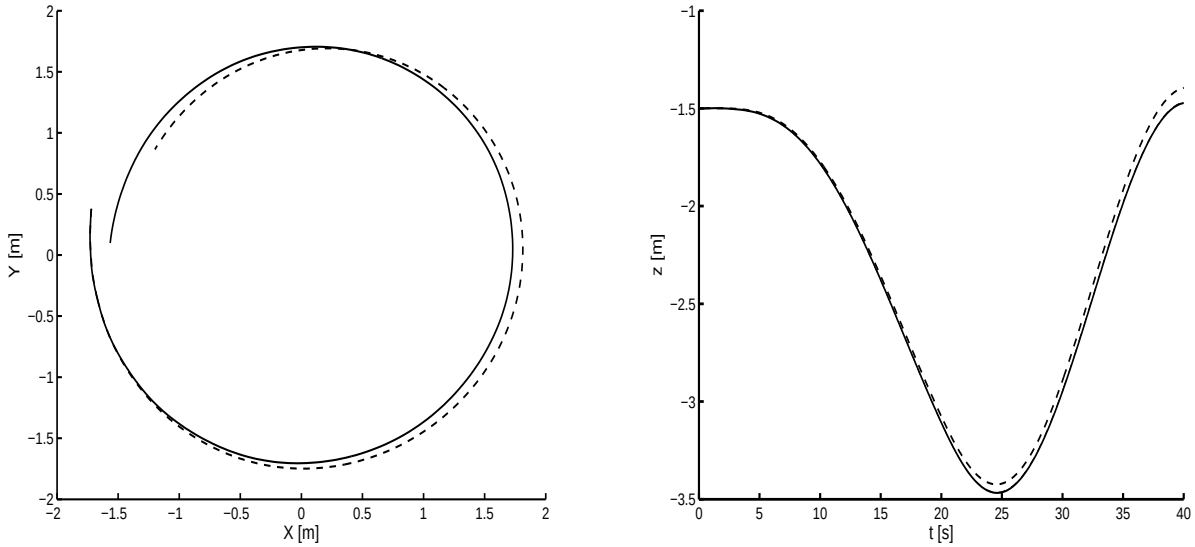
	x	y	z
$E^{tr}_{rms}[m]$	1.96	2.28	.35
E^{tr}_{max} [m]	4.8	4.2	.60

Again online learning can be applied to perform an update of the neural network knowledge regarding the damping. The trajectories resulting from such an online update while controlling the craft, are shown in figures 5.13a and 5.13b. Although the neural network feedforward controller does not follow the reference path perfectly, a considerable increase in performance is observed, which is listed in table 5.5.

Table 5.5: Tracking errors due to changing dynamics with online learning

	x	y	z
$E^{tr}_{rms}[m]$	.31	.25	.042
E^{tr}_{max} [m]	.67	.77	.078

Again the corresponding changes in the neural network estimate of the damping will show local improvements while the approximation might be worse globally. However, while operating the vehicle further the state space visited will gradually become a representative set. If online learning is applied the neural networks will thus learn to represent the new damping eventually.



(a) Horizontal projection

(b) Depth profile versus time

Figure 5.13: Path followed by vehicle with unknown change in damping and online learning (ppm solid line, cpm dotted line)

Chapter 6

A comparison of the novel strategy to least squares identification

This chapter presents a comparison of the proposed identification method, using neural networks, with a standard least squares (LS) method. The LS algorithm, used in the Matlab command 'arx' (Signal Processing Toolbox), calculates the least squares fit of the data to an autoregressive model with exogeneous inputs, using QR factorisation (Gollub and Van Loan, 1996). In section 6.1, this LS algorithm is used to derive a LS model of Tethra using signals obtained from simulations. For the sake of reality the signals are impaired with realistic noise. Consecutively, in section 6.2, the findings from section 6.1 will be compared to the results obtained in section 5.3 with the proposed identification method applied to the Tethra vehicle.

6.1 Identification of craft parameters using a standard least squares algorithm

In order to be able to use a least squares method, the Coriolis and centripetal forces, represented by the matrix C_{RB}, are neglected resulting in the following identification model:

$$\boldsymbol{\tau} = \mathbf{M}_{lms}\dot{\boldsymbol{\nu}} + (\mathbf{D_l} + \text{diag}(\mathbf{D_q})|\boldsymbol{\nu}|)\boldsymbol{\nu}, \tag{6.1}$$

where diag(*arg*) indicates a diagonal matrix with only nonzero elements *arg* on the main diagonal. With the identification model from equation 6.1 and collected data for the input forces and moments, $\boldsymbol{\tau}$ (including the restoring forces), the velocity, $\boldsymbol{\nu}$, and the acceleration of the craft, $\dot{\boldsymbol{\nu}}$, the mass and (linear and quadratic) damping matrices are identified. It should be noted that, to be able to use the least squares algorithm, the damping matrix has to be assumed to be of a certain algebraic form. In this case the damping is assumed to be consisting of a linear plus quadratic term.

With simulated data, gathered during 1000 seconds of manual control of the Tethra craft, an identification was performed. The measurements were impaired with zero mean noise (see section 5.3.1 for the noise parameters), yielding the following parameter estimates:

$$M = \begin{pmatrix} 273 & -17 & 14 & -7 & -60 & -3 \\ 2 & 260 & -8 & 64 & -3 & 22 \\ 8 & 7 & 233 & 6 & -1 & 3 \\ 1 & 61 & -2 & 55 & -1 & 6 \\ -61 & 3 & -2 & 1 & 51 & 1 \\ 0 & -1 & 0 & 0 & 0 & 53 \end{pmatrix}, D_l = \begin{pmatrix} 217 & -54 & -5 & 19 & 61 & 17 \\ -1 & -23 & 0 & -294 & 30 & 28 \\ 3 & -12 & 116 & -10 & -70 & -4 \\ -1 & -17 & 0 & -47 & 5 & 6 \\ -36 & 12 & 1 & -4 & 8 & -3 \\ 0 & 1 & 0 & 2 & 0 & 26 \end{pmatrix},$$

$$D_q = \begin{pmatrix} 18 & 0 & 0 & 0 & 0 & 0 \\ 0 & 105 & 0 & 0 & 0 & 0 \\ 0 & 0 & 96 & 0 & 0 & 0 \\ 0 & 0 & 0 & 247 & 0 & 0 \\ 0 & 0 & 0 & 0 & 131 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{pmatrix}. \tag{6.2}$$

6.2 Comparison of methods

In order to be able to compare the newly proposed identification method with the standard least squares identification, the identified parameters need to be known explicitly. Due to using neural networks to represent the damping, the proposed identification algorithm does not offer explicit values for the damping parameters. Therefore these will have to be extracted from the trained neural networks. This is done by presenting the neural networks with a range of inputs. The resulting output of the neural networks (damping forces and moments) is recorded. Consecutively the least squares algorithm described in section 6.1, can be used to compute a best fit of

Table 6.1: Comparison of identified mass and damping matrices obtained with a standard LS algorithm and the newly proposed method using neural networks

	$\mathbf{M}$	$\mathbf{D}_l$	$\mathbf{D}_q$
Model	$\begin{pmatrix} 231 & 0 & 0 & 0 & -52 & 0 \\ 0 & 231 & 0 & 52 & 0 & 0 \\ 0 & 0 & 231 & 0 & 0 & 0 \\ 0 & 52 & 0 & 52 & 0 & 0 \\ -52 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 116 & 0 & 0 & 0 & -15 & 0 \\ 0 & 116 & 0 & 15 & 0 & 0 \\ 0 & 0 & 116 & 0 & 0 & 0 \\ 0 & 15 & 0 & 26 & 0 & 0 \\ -15 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26 \end{pmatrix}$	$\begin{pmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{pmatrix}$
LS	$\begin{pmatrix} 273 & -17 & 14 & -7 & -60 & -3 \\ 2 & 260 & -8 & 64 & -3 & 22 \\ 8 & 7 & 233 & 6 & -1 & 3 \\ 1 & 61 & -2 & 55 & -1 & 6 \\ -61 & 3 & -2 & 1 & 51 & 1 \\ 0 & -1 & 0 & 0 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 217 & -54 & -5 & 19 & 61 & 17 \\ -1 & -23 & 0-294 & 30 & 28 & \\ 3 & -12 & 116 & -10 & -70 & -4 \\ -1 & -17 & 0 & -47 & 5 & 6 \\ -36 & 12 & 1 & -4 & 8 & -3 \\ 0 & 1 & 0 & 2 & 0 & 26 \end{pmatrix}$	$\begin{pmatrix} 18 & 0 & 0 & 0 & 0 & 0 \\ 0 & 105 & 0 & 0 & 0 & 0 \\ 0 & 0 & 96 & 0 & 0 & 0 \\ 0 & 0 & 0 & 247 & 0 & 0 \\ 0 & 0 & 0 & 0 & 131 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{pmatrix}$
NN	$\begin{pmatrix} 255 & 2 & -3 & 2 & -53 & 0 \\ 0 & 222 & 0 & 47 & 0 & -1 \\ 15 & 1 & 236 & 1 & -14 & 0 \\ 0 & 52 & 0 & 51 & 0 & 1 \\ -54 & -2 & 3 & -2 & 49 & 0 \\ 0 & 1 & 0 & 2 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 121 & 2 & -1 & -7 & -36 & -1 \\ 1 & 129 & 0 & 37 & 0 & -1 \\ -1 & 2 & 114 & -5 & -6 & 0 \\ 0 & 10 & 0 & 30 & -2 & 0 \\ -10 & 1 & -3 & 2 & 11 & 0 \\ 2 & 9 & 1 & 1 & 2 & 18 \end{pmatrix}$	$\begin{pmatrix} 93 & 0 & 0 & 0 & 0 & 0 \\ 0 & 104 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 52 & 0 \\ 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix}$

the damping model to this data.

6.2.1 Identification of mass, linear and quadratic damping

As the damping is modelled with linear and quadratic terms in the least squares identification method, the same will be done in this case. It should be stressed that the above procedure is executed only to be able to compare the two methods.

Table 6.1 shows the mass and damping matrices resulting from the least squares and the newly proposed identification methods together with the original model parameters used in the simulation.

From table 6.1 it can be seen that the newly proposed algorithm yields a considerably better estimate of the mass and damping parameters than the LS algorithm. Although substantial, the relative errors of the individual parameter estimates in the NN model are smaller and less frequent than in the LS estimates. Table 6.2 gives an overview of some of the errors found in the estimations. The criteria used are the maximum relative error of the elements on the main

diagonal of the matrices:

$$E_{\text{Diag}} = \max_i \left[abs \left(\frac{\mathbf{A}_{est}(i,i) - \mathbf{A}_{Model}(i,i)}{\mathbf{A}_{Model}(i,i)} \right) \right] \qquad \mathbf{A}_{Model}(i,i) \neq 0, \tag{6.3}$$

the maximum relative error of those off-diagonal elements that are non zero in the model matrix:

$$E_{\text{Off-diag}} = \max_{i,j} \left[abs \left(\frac{\mathbf{A}_{est}(i,j) - \mathbf{A}_{Model}(i,j)}{\mathbf{A}_{Model}(i,j)} \right) \right] \qquad i \neq j, \mathbf{A}_{Model}(i,j) \neq 0, \tag{6.4}$$

and the maximum error of those elements that are zero in the model matrix, relative to the largest element in the matrix:

$$E_{\text{Zero-elements}} = \max_{i,j} \left[abs \left(\frac{\mathbf{A}_{est}(i,j)}{\max(\mathbf{A}_{Model})} \right) \right] \qquad \mathbf{A}_{Model}(i,j) = 0, \tag{6.5}$$

where $\max(\mathbf{A})$ denotes the value of the largest element in matrix $\mathbf{A}$.

Table 6.2: Comparison of maximum errors obtained with a standard LS algorithm and the newly proposed method using neural networks

	$\mathbf{M}$		$\mathbf{D}_l$		$\mathbf{D}_q$	
	NN	LS	NN	LS	NN	LS
E_{Diag} [%]	10	18	56	281	97	723
$E_{Off-diag}$[%]	9	23	147	2060	NA	NA
$E_{Zero-elements}$[%]	6	10	8	60	NA	NA

The errors listed in table 6.2 are high and, in some cases, unacceptably high. However, it should be kept in mind that this identification was performed on an underactuated craft with only 1000 seconds of training data. To show the influence of the lack of excitation in the non-controllable degrees of freedom, the errors are recomputed for all controllable degrees of freedom

(surge, heave and yaw). The results, displayed in table 6.3, show more agreeable errors, in particular for the newly proposed identification method using neural networks.

Table 6.3: Comparison of errors for a 3 degree of freedom identification

	$\mathbf{M}$		$\mathbf{D}_l$		$\mathbf{D}_q$	
	NN	LS	NN	LS	NN	LS
E_{Diag} [%]	10	18	31	87	20	82
$E_{Off-diag}$ [%]	0	0	0	0	NA	NA
$E_{Zero-elements}$ [%]	6	6	2	15	NA	NA

6.2.2 Identification of the craft parameters using an extended data set

To show the beneficial influence of training with more data (longer manual control of the craft whilst recording data) another 1000 seconds of data was captured and used for identification of the craft. Table 6.4 shows the resulting mass and damping matrices and table 6.5 the corresponding errors. The NN approach clearly benefits from the increased data set, whereas the results for the LS algorithm are still relatively poor and in some cases even worse.

Table 6.4: Comparison of identified mass and damping matrices with extended data set

	$\mathbf{M}$	$\mathbf{D}_l$	$\mathbf{D}_q$
Model	$\begin{pmatrix} 231 & 0 & 0 & 0 & -52 & 0 \\ 0 & 231 & 0 & 52 & 0 & 0 \\ 0 & 0 & 231 & 0 & 0 & 0 \\ 0 & 52 & 0 & 52 & 0 & 0 \\ -52 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 116 & 0 & 0 & 0 & -15 & 0 \\ 0 & 116 & 0 & 15 & 0 & 0 \\ 0 & 0 & 116 & 0 & 0 & 0 \\ 0 & 15 & 0 & 26 & 0 & 0 \\ -15 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26 \end{pmatrix}$	$\begin{pmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{pmatrix}$
LS	$\begin{pmatrix} 282 & 12 & -1 & 9 & -60 & 0 \\ -2 & 263 & -3 & 64 & 3 & 17 \\ 6 & 1 & 234 & 1 & 1 & 1 \\ 0 & 62 & -1 & 56 & 0 & 5 \\ -63 & -3 & 1 & -2 & 51 & 0 \\ 0 & -1 & 0 & 0 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 217 & -11 & -5 & 1 & 40 & 17 \\ 16 & -26 & -2 & -176 & 48 & 31 \\ 2 & 1 & 116 & 5 & -65 & -7 \\ 4 & -18 & 0 & -19 & 13 & 7 \\ -37 & 2 & 1 & 0 & 13 & -3 \\ 0 & 1 & 0 & 1 & 0 & 26 \end{pmatrix}$	$\begin{pmatrix} 38 & 0 & 0 & 0 & 0 & 0 \\ 0 & 90 & 0 & 0 & 0 & 0 \\ 0 & 0 & 101 & 0 & 0 & 0 \\ 0 & 0 & 0 & 48 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{pmatrix}$
NN	$\begin{pmatrix} 249 & 1 & -2 & 1 & -52 & 0 \\ 0 & 229 & 0 & 48 & 0 & -3 \\ 10 & 0 & 231 & 0 & -9 & 0 \\ 0 & 52 & 0 & 52 & 0 & 0 \\ -54 & -1 & 2 & -1 & 50 & 0 \\ 0 & -1 & 0 & 1 & 0 & 53 \end{pmatrix}$	$\begin{pmatrix} 113 & -1 & 0 & -7 & -17 & -1 \\ 1 & 128 & 0 & 18 & 1 & -1 \\ -1 & 1 & 115 & -4 & -1 & 0 \\ 0 & 12 & -1 & 25 & -2 & 0 \\ -12 & -1 & -3 & 2 & 22 & 0 \\ 0 & 6 & 0 & 0 & 5 & 29 \end{pmatrix}$	$\begin{pmatrix} 102 & 0 & 0 & 0 & 0 & 0 \\ 0 & 98 & 0 & 0 & 0 & 0 \\ 0 & 0 & 101 & 0 & 0 & 0 \\ 0 & 0 & 0 & 36 & 0 & 0 \\ 0 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26 \end{pmatrix}$

Table 6.5: Comparison of maximum errors obtained with the extended data set

	$\mathbf{M}$		$\mathbf{D}_l$		$\mathbf{D}_q$	
	NN	LS	NN	LS	NN	LS
E_{Diag} [%]	8	22	12	173	20	62
$E_{Off-diag}$ [%]	8	23	20	1273	NA	NA
$E_{Zero-elements}$ [%]	4	7	6	56	NA	NA

6.2.3 Influence of the mass matrix identification on the damping identification

The increased accuracy in the identification of the damping parameters using neural networks can mainly be ascribed to the increased accuracy with which the mass matrix is identified. Due to the doubled training data set, the dynamics of the craft for low velocity are captured with a greater accuracy. Because the identification of the mass matrix relies on the NN approximation of the dynamics for low velocities, the mass matrix can thus be distilled with a higher accuracy. As described in section 4.3.4, the estimated mass matrix is used in the subsequent identification of the damping parameters. This results in the algorithm being extremely sensitive to errors in the mass matrix approximation. In particular the correct estimation of the zero elements in the mass matrix is important. This was verified through simulations, with the original data set of a 1000 second long simulation, in which all elements in the identified mass matrix that should be zero, were manually set to zero. The resulting mass matrix is:

$$\mathbf{M}_{NN} = \begin{pmatrix} 255 & 0 & 0 & 0 & -53 & 0 \\ 0 & 222 & 0 & 47 & 0 & 0 \\ 0 & 0 & 236 & 0 & 0 & 0 \\ 0 & 52 & 0 & 51 & 0 & 0 \\ -54 & 0 & 0 & 0 & 49 & 0 \\ 0 & 0 & 0 & 0 & 0 & 53 \end{pmatrix}. \tag{6.6}$$

Using the mass matrix from equation 6.6 for generation of the training data and subsequent identification of the damping, yields a maximum relative error of 31% for the diagonal elements of both $\mathbf{D}_l$ and $\mathbf{D}_q$. This is an improvement of respectively 80% and 210%. Use of the original mass matrix,$\mathbf{M}_{Model}$, for the identification yields maximum relative errors of the diagonal elements of 10% and 25% respectively. This shows that the correct identification of zero elements in the mass matrix is indeed of great importance. The decrease in $E_{zero-elements}$ of 50 $\%$ can thus be concluded to be of great influence on the improved identification of the damping parameters.

6.2.4 Identification of the damping using a known mass matrix

It should be noted that in the least squares identification, the Coriolis and centripetal terms were neglected. This insurmountably introduces an error, the magnitude of which depends on the relative influence of the Coriolis and centripetal forces and moments. For craft travelling at low speed the contribution of these forces and moments will be moderate and the simplification can thus be justifiable. However, to get around the introduction of this error, the damping terms can be identified once more with the least squares method, but now with the assumption that the mass matrix has already been identified. In many practical situations this is indeed not such a bad assumption as the mass matrix can be identified accurately from craft geometries. For this

Table 6.6: Estimated damping matrices for LS algorithm when the mass matrix is assumed to be known

$\mathbf{D}_l$	$\mathbf{D}_q$
$\begin{pmatrix} 112 & 0 & 0 & -8 & -27 & -1 \\ 1 & 128 & 0 & 31 & 1 & -1 \\ -1 & 1 & 115 & -4 & -1 & 0 \\ 0 & 12 & -1 & 22 & -2 & 0 \\ -12 & -1 & -3 & 3 & 19 & 0 \\ 0 & 5 & 0 & 0 & 5 & 22 \end{pmatrix}$	$\begin{pmatrix} 103 & 0 & 0 & 0 & 0 & 0 \\ 0 & 98 & 0 & 0 & 0 & 0 \\ 0 & 0 & 101 & 0 & 0 & 0 \\ 0 & 0 & 0 & 49 & 0 & 0 \\ 0 & 0 & 0 & 0 & 39 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 \end{pmatrix}$

Table 6.7: Comparison of errors in estimated damping matrices when the mass matrix is assumed to be known

	$\mathbf{D}_l$			$\mathbf{D}_q$		
	NN	LS_1	LS_2	NN	LS_1	LS_2
E_{Diag} [%]	12	173	24	20	62	63
$E_{Off-diag}$ [%]	20	1273	107	NA	NA	NA
$E_{Zero-elements}$ [%]	6	55	6	NA	NA	NA

experiment, the mass matrix identified with the NN identification method is used as the *a priori* known mass matrix. Hence the training data used for training of the damping neural networks (see section 4.3.4) can be used, which enables a fair comparison of the damping approximation using the least squares method and the proposed method using neural networks.

The resulting estimates for the linear and quadratic damping matrices are shown in table 6.6 and the errors in table 6.7. For ease of reference the errors obtained with the LS algorithm when identifying all three parameters ($\mathbf{M}$, $\mathbf{D}_l$ and $\mathbf{D}_q$), indicated with LS_1, and the newly proposed method (table 6.5) are repeated in table 6.7. The estimates making use of the *a priori* known mass matrix, are indicated with LS_2.

Table 6.7 shows that the LS algorithm, in the most favourable circumstances, still yields a less accurate estimation than the newly proposed identification method using neural networks.

6.3 Conclusions

In this chapter a comparison of the newly proposed identification method using neural networks with a standard least squares optimisation algorithm was presented. The results obtained with the newly proposed method are significantly more accurate than those obtained with the LS optimisation algorithm. The higher accuracy can mainly be attributed to the fact that neural networks are better capable of filtering the non-zero mean and correlated noise terms present in the velocity measurements. Additionally, outliers in the measurements (forces and moments generated through collisions with the sides of the pool and due to surfacing) are successfully ignored by the neural networks, but do pose problems for the LS method.

It was shown that an increased data set significantly improves the estimates obtained through the newly proposed identification method. However, the superior accuracy is accompanied by a substantially increased computational complexity. Furthermore, the identification of the damping is sensitive to inaccurate identification of the mass matrix. Accurate identification of the latter is thus important for a good overall identification result.

Chapter 7

Experiments with the Minesniper

The feasibility of the novel identification method, presented in chapter 4 and subjected to simulations in chapter 5, is tested in an experiment in this chapter. The parameters to be identified are the damping parameters in a torpedo shaped underwater vehicle, called Minesniper. The Minesniper is a craft that is designed to find, approach and detonate sea mines in a controlled fashion.

First, in section 7.1, an appropriate model for the Minesniper will be presented and the identification strategy will be outlined. Then, in section 7.2, the results of this identification strategy, applied to the Minesniper craft, are presented and conclusions are drawn in section 7.3.

Figure 7.1: The Minesniper MK II, courtesy of Kongsberg ASA

7.1 Neural modelling of the Minesniper

The Minesniper is a defensive weapon used for finding and detonating sea mines in a controlled way. This torpedo shaped craft, which is shown in figure 7.1, is developed by Kongsberg ASA in Norway and is currently in its final testing phase.

Currently tethered using an optical fibre, the Minesniper receives commands from a surface unit controlled by a human operator. The operator can keep track of the position of the craft through a short base line system. Two hydrophonic receivers, 10 metres apart, are used to obtain the position and attitude of the craft. Although the accuracy of this short base line system is limited, reliable values for the velocity and the acceleration of the craft can be obtained through post processing of the position data.

In the Minesniper, a moving mass enables the torpedo shaped underwater craft to induce a pitch moment and thus change its pitch orientation. As a result the dynamics model of the craft exhibits time-varying parameters that are normally constant. The mass matrix, and thus the Coriolis matrix, will become time dependent due to the changing centre of gravity. Closely following the derivation using Newtonian formalisms as in Fossen (2002), a model including

the forces and moments due to the sliding mass, can be obtained. In appendix B this derivation is presented. The resulting expression is given in equation 7.1, with the vector of forces and moments due to the sliding mass, written as a vector product of a matrix $\mathbf{V}$ with the velocity $\boldsymbol{\nu}$.

$$\mathbf{M}(\mathbf{r}_g^b)\dot{\boldsymbol{\nu}} + (\mathbf{C}(\boldsymbol{\nu}, \mathbf{r}_g^b) + \mathbf{D}(\boldsymbol{\nu}))\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}, \mathbf{r}_g^b) + \mathbf{V}\boldsymbol{\nu} = \boldsymbol{\tau}, \tag{7.1}$$

with:

$$V = \begin{bmatrix} \mathbf{0}^{3\times3} & -m\mathbf{S}(\dot{\mathbf{r}}_g^b) \\ \mathbf{0}^{3\times3} & -2\dot{\mathbf{r}}_b \left[m_b\mathbf{r}_b + m\mathbf{r}_g^b\right] \end{bmatrix} \tag{7.2}$$

The explicit dependency of the various matrices on the changing centre of gravity is represented by $\mathbf{r}_g^b$ in equation 7.1.

The original model of the Minesniper used for simulations was derived using existing techniques and software such as WAMIT (Refsnes, 2003). For slender bodies, such as the Minesniper, these techniques generally give good results for the mass and added mass matrices. However, for the damping, the identified parameters are at best a reasonable approximation (Refsnes and Sørensen, 2004). This is due to the fact that, for example, WAMIT only calculates the potential damping. The inaccuracies in the identification of the damping are illustrated in figure 7.2, in which both the measured trajectory and a trajectory obtained from simulations, are shown. Dots are placed on the tracks every second to indicate time in this figure.

To improve the identification of the damping, neural networks are used in an identification scheme as outlined in chapter 4. Based on the real acceleration, $\dot{\boldsymbol{\nu}}_{ppm}$, and the acceleration obtained from a control plant model (cpm), in which the exact mass and input forces are known, but the damping is regarded to be zero, the product of damping and velocity is obtained for all velocities at which the craft travels during the experiments:

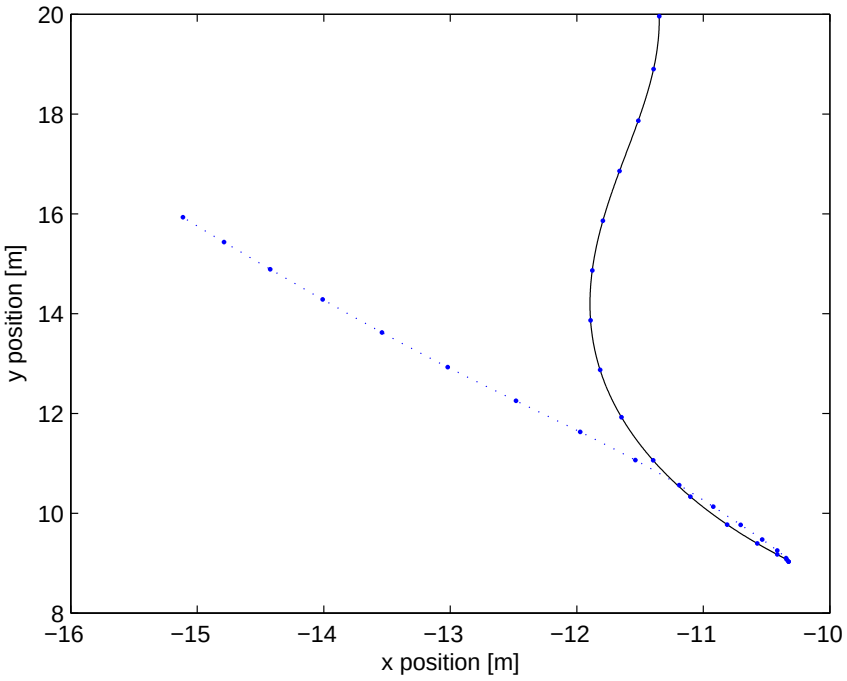


Figure 7.2: Predicted (solid line) and measured (dotted line) trajectory of the MineSniper in a horizontal plane for the original model

$$\mathbf{D}(\boldsymbol{\nu})\,\boldsymbol{\nu} = \mathbf{M}\left[\dot{\boldsymbol{\nu}}_{cpm} - \dot{\boldsymbol{\nu}}_{ppm}\right]. \tag{7.3}$$

Consecutively six separate neural networks are trained to each represent one element of the six element vector, $\mathbf{D}(\boldsymbol{\nu})\,\boldsymbol{\nu}$. As the inputs to the neural networks are the velocity of the craft, the neural networks are trained to perform a mapping from $\boldsymbol{\nu}$ to $\mathbf{D}(\boldsymbol{\nu})\,\boldsymbol{\nu}$. After training the neural networks can be used in the model as shown in equation 7.4:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \underline{\mathbf{D}\boldsymbol{\nu}} + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}, \tag{7.4}$$

where $\mathbf{D}\boldsymbol{\nu}$ is underlined to emphasise that this term is not a product of $\mathbf{D}$ and $\boldsymbol{\nu}$, but is the output of the neural networks. In chapter 5 the identification of the damping was performed

using back propagation neural networks. During the identification of damping in the Minesniper, however, it was found that insufficient information was available to train the neural networks for all possible velocities in all 6 degrees of freedom. If, during prediction of the craft behaviour, the neural network is presented with a velocity that is significantly different from those available in the training data set, the output of the back propagation neural network is unpredictable. This is highly undesirable as it may lead to instabilities of the model. Radial basis function (RBF) networks are used to prevent the unreliable predictions from occurring. In an RBF network the neurons are placed in an m-dimensional space, where m is the dimension of the input vector. The neurons are spread out over the m-dimensional space in such a way that the mean square error of predictions, measured over the full training set, is minimised. An RBF network possesses the property of zero prediction when an input vector is presented that is new to the network. New in this sense means that the euclidian distance between the presented input vector and the positions of all the nodes in the RBF network is larger than a predetermined value. The prediction of the damping will thus in the worst case be zero rather than highly unpredictable in case of the back propagation network.

7.2 Practical results

The Minesniper MKII is torpedo shaped and is 1.93m long and 0.17m in diameter. Its weight is 40kg and it has a nominal speed of 3 knots[18]. Main propulsion is achieved using two 150W/24V DC electric motors located on each side of the hull at the center of the vehicle providing surge and yaw motion. In transit, pitch and thereby depth is controlled by movement of a 1.7kg mass that is placed in the rear half of the vehicle. In addition, a 70W/24V DC vertical motor is placed

[18] 1 Knot equals 0.514 m/s

at the center of the vehicle, primarily intended for depth control in the vicinity of the target. With this actuator system, the vehicle is capable of controlling surge, heave, pitch and yaw.

Using the derived model (equation 7.1), the input forces from the thrusters and sliding pitch mass were calculated for the test runs. During these runs the position and attitude of the craft was measured and recorded using a short base line acoustical position measurement system. After post processing of the position data to obtain smooth signals, the velocity and acceleration of the craft were obtained through differentiation and transformation of the position data. This is admittedly a method prone to the introduction of large errors, but necessary, given the fact that only position measurements were available. The obtained data was then used to calculate the training data for the neural networks as described by equation 7.3.

RBF network training was performed with a total of 50 seconds of experimental data. 15 Seconds of experimental data was used in a test set to validate the training results. The RBF networks each consist of 500 neurons. As in RBF networks, the size of the hidden layer depends on the amount of information that has to be represented by the network, it can be expected, that the amount of neurons necessary for identification of the damping for a full data set, will increase substantially.

Figure 7.3 shows results of the RBF network training on the training data set. As can be seen, the RBF network models the damping accurately, resulting in a correct prediction of the position up to 10 seconds into the simulation.

Performance of the RBF networks on the test data set is shown in figure 7.4. As the test set contains a high amount of not previously seen data, the original classical model of the damping is used in parallel to the RBF networks. The classical model represents the damping with a linear and a quadratic term as shown in equation 7.5:

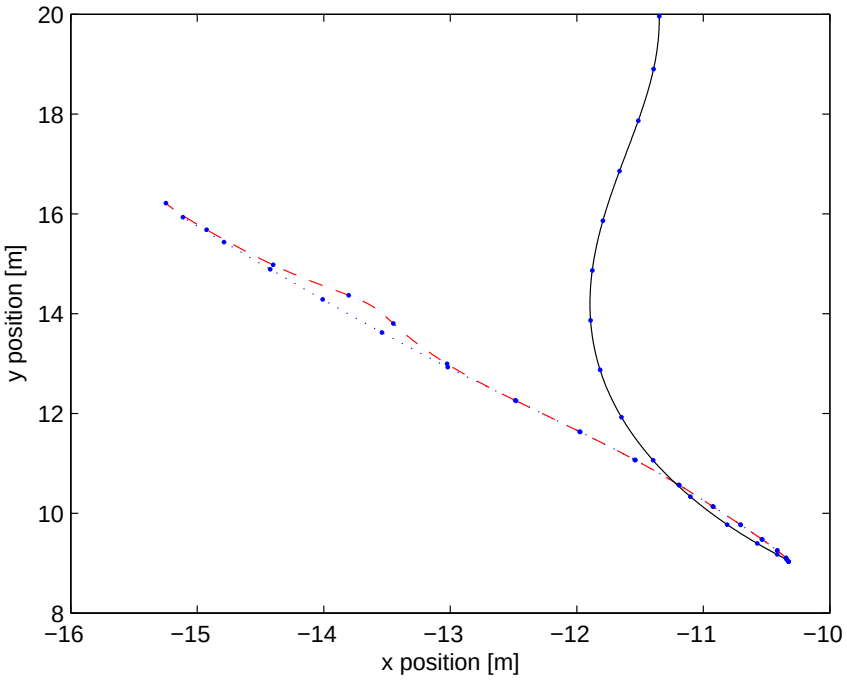


Figure 7.3: Predicted (solid: original damping, dashed: RBF network damping) and measured (dotted line) trajectories of the Minesniper in a horizontal plane for the training data set

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 83 & 0 & 0 & 0 & -20 \\ 0 & 0 & 85 & 0 & 30 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 27 & 0 & 9.5 & 0 \\ 0 & -17 & 0 & 0 & 0 & 6.8000 \end{bmatrix} + \\ + \quad \mathrm{diag}([7\,100\,250\,50\,90\,90]) \cdot |\boldsymbol{\nu}|. \tag{7.5}$$

In case an input vector is presented to the RBF networks, that is significantly different from all input vectors used for training, the model will choose the output of the classical damping

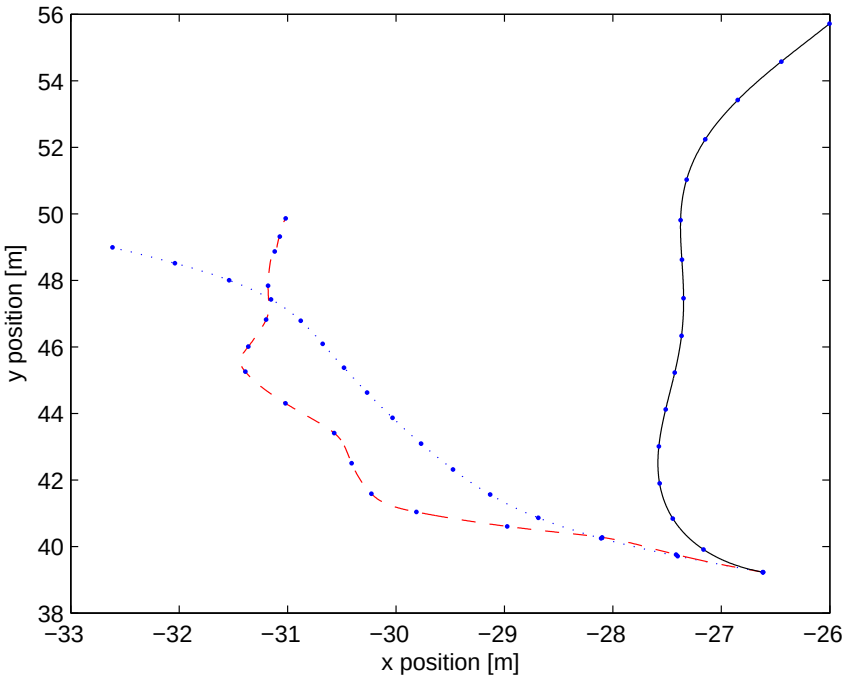


Figure 7.4: Predicted (solid original damping, dashed RBF network damping) and measured (dotted line) trajectories of the Minesniper in a horizontal plane for the test data set

model rather than the output of the RBF networks.

As can be seen from figure 7.4, the RBF networks do not succeed to predict the position of the craft very accurately. However, this was not expected as the test data set is considerably different from the training data set. It can be seen though, that the prediction is more accurate than the prediction done by modelling the damping according to equation 7.5.

To gain insight in the damping estimates learned by the RBF networks, estimates of the diagonal terms of the damping matrices are distilled from the RBF networks. The damping is assumed to be composed of a linear and a quadratic term. By presenting the full training data set to the network, all values for the damping forces are obtained. Consecutively a least squares algorithm is used to obtain a representation of the damping forces and moments as a function of the velocity, $\boldsymbol{\tau_D} = f(\boldsymbol{\nu}, \boldsymbol{\nu} \cdot |\boldsymbol{\nu}|)$, in a diagonal form: $\boldsymbol{\tau_D} = \text{diag}([\tau_{D_{X,u}} \tau_{D_{Y,v}} \tau_{D_{Z,w}} \tau_{D_{K,p}} \tau_{D_{L,q}} \tau_{D_{M,r}}])$.

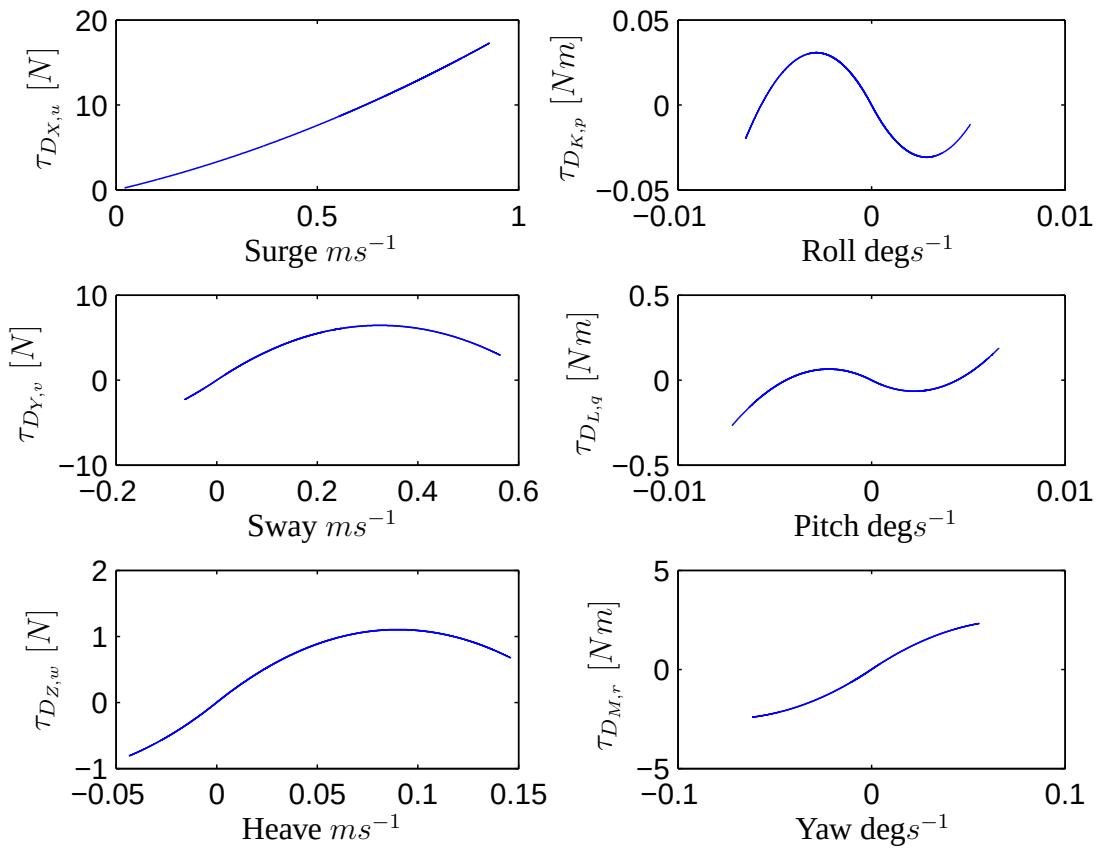


Figure 7.5: Approximates of damping, as modelled by the RBF networks, in 6 degrees of freedom as a function of speed in the same degree of freedom.

The approximations of the diagonal elements of the damping matrix thus obtained, are shown in figure 7.5.

For comparison the same plots were generated for the diagonal damping parameters used in the classical damping model. The results are depicted in figure 7.6.

Comparison of figures 7.5 and 7.6 shows that the damping estimates in surge, sway, heave and yaw obtained with the RBF networks are physically plausible. The negative slope for higher sway and heave velocity may be due to the insufficient size of the data set that was available. However, this requires further research. For both roll and pitch the found values are physically

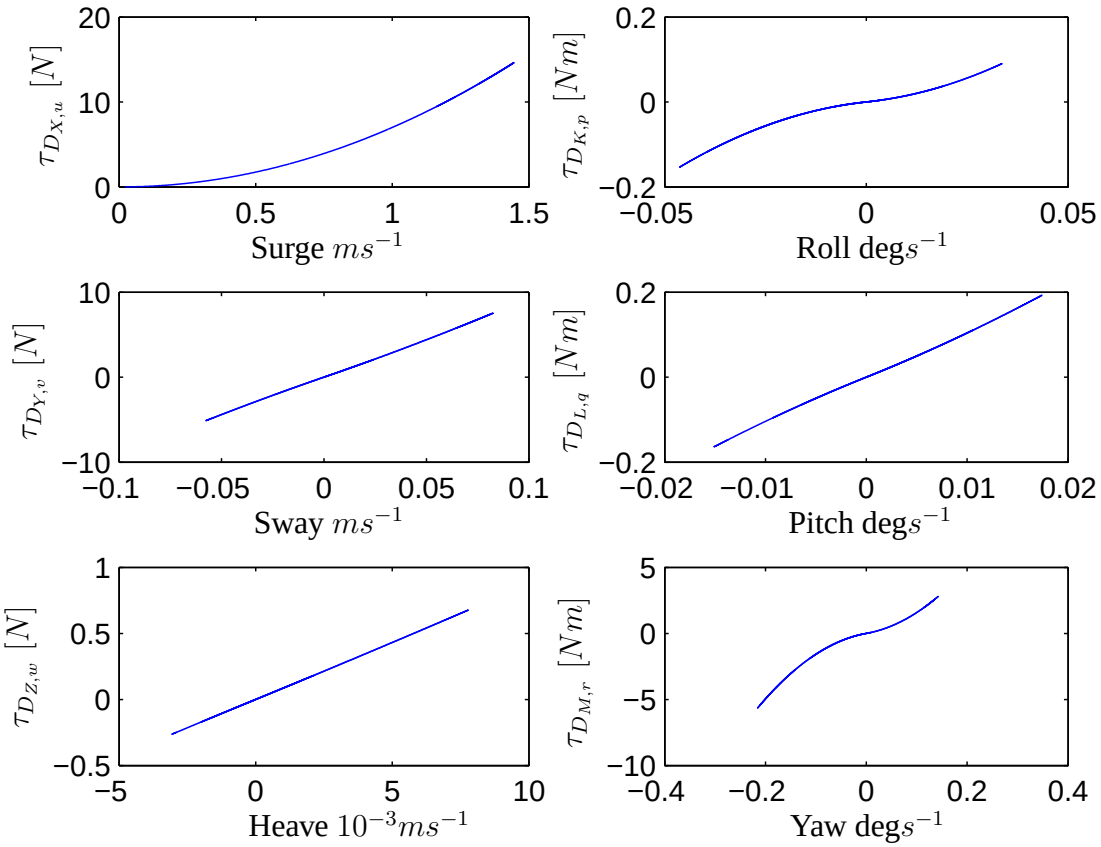


Figure 7.6: Approximates of damping, as modelled in the classical model, in 6 degrees of freedom as a function of speed in the same degree of freedom.

impossible. From the plots a negative damping should be concluded for the regions around zero velocity. However, the amplitude of the excitation in these degrees of freedom is too low to obtain reliable results.

7.3 Conclusions

In this chapter, identification of the damping of an underwater vehicle using radial basis function networks was discussed. Neural networks are used for the identification of the damping as they can represent the damping without using an explicit model. Only a limited data set was available

for training of these networks. As the data set does not cover the whole region of operation, back propagation neural networks are not suitable because they yield unpredictable outputs for regions that are not represented in the data set. Instead, RBF networks were used as they have the convenient property that their output tends to zero for inputs different from the ones that were used for training.

To improve the prediction of damping for these new velocities, a classical damping model is used in parallel to the RBF networks. Results using experimental data were presented, showing the advantages of the presented identification scheme. Other than for open loop predictions of the position, the model could function as an improved model in model predictive control, or the model parameters could be used directly in a conventional control scheme. Due to the parallel operation of the classical damping model and the RBF networks, it is more likely that stability can be guaranteed than in case of a standard neurocontroller. This, however, remains to be demonstrated.

The size of an RBF network grows with the size of the training data set. Therefore, back propagation neural networks are preferred when the size of the data set increases. Because of the mentioned disadvantage of back propagation neural networks, namely that their output is unpredictable for not previously visited states, it is of prime importance to obtain a rich data set.

Chapter 8

Conclusions and future work

8.1 Summary of the work performed in this PhD

In this work the main focus was on identification of the dynamics of underwater vehicles. Having explored the field of neural control of underwater vehicles, a new classification of the approaches for neural control of underwater vehicles was made. The conclusions drawn from the classification, pointed in the direction of control systems which employ neural networks for changing dynamics, but that still, to some extent, rely on classical controllers for reasons of stability and robustness. In this particular approach, called Augmented Control (AC), a lot can be gained if an accurate model of the dynamics is available. Often, however, this is not the case as certain parameters in the model are not fully understood and, as a result, not accurately identifiable using existing techniques. For this reason, the focus of this PhD was placed on the identification of accurate models using neural networks. Neural networks were chosen as they can map any smooth real function on a closed set. Rather than using one neural network to learn the full dynamics, an approach using several neural networks was developed. The main advantages are two-fold. As the neural networks are used to identify individual model parameters, learning becomes an easier task. This is demonstrated by the fact that the neural networks used for making the six dimensional identification are relatively small. More important however, is

that the model with separate neural networks for individual parameters can easily be reformulated to be used in a controller. This is not a trivial task for a neural network representing the full dynamics.

8.2 Main contributions

Initially a new classification of neural control for underwater vehicles was developed and presented. The advantage of this new classification is that it clearly shows what strategies offer what advantages and, often more importantly, what disadvantages. The classification thus gives a direction for future efforts in the field. A review journal paper introducing the new classification was published in the Elsevier journal Engineering Applications of Artificial Intelligence (Van de Ven et al., 2005).

Consecutively a new identification approach making use of neural networks was developed. The obtained models are novel and highly interesting as they open up ways to model parameters for which no, or only limited, physical insights are available. This aptitude for identifying parameters in a model-free fashion is combined with the possibility to add as much model information as known or desirable. The novel approach is not limited to the identification of underwater vehicles. It can be used for any process as long as a sufficient number of parameters are measurable or deductable from measurements. The results were published in Van de Ven et al. (2004c), Van de Ven et al. (2004a) and Van de Ven et al. (2004b). Additionally this work presents one of the first models for all six degrees of freedom.

The newly proposed identification method using neural networks was thoroughly tested in simulations. A journal paper, detailing this method and presenting a simulation study, has been accepted for publication in the Elsevier journal Control Engineering Practice. The simulations

show that an accurate identification is possible and that the obtained models can easily be used for control purposes. Comparison with a least squares algorithm showed that the newly proposed identification yields a considerable improvement in accuracy, albeit at the expense of increased computational complexity.

Testing of the novel approach using a real world system has been performed at the Marine Technology Center of the Norwegian University for Science and Technology (NTNU) in Trondheim, Norway. Based on a limited test set, the feasibility of identifying improved damping estimates using the proposed identification strategy was demonstrated. Of notable importance in this identification was that the measurements were obtained from a long base line system with low accuracy, thus demonstrating the aptitude of the neural networks for handling noisy measurements.

8.3 Conclusions

The advantages offered by neural networks when used for control of underwater vehicles, are considerable. Due to their unpredictability the seas, oceans and other waterways of our planet pose considerable problems for classical control strategies. Neural networks offer the possibility to adapt controllers on the fly, thus guarding the craft from damage or even loss when circumstances take an unexpected turn.

However, as with the current state of research in the field of artificial intelligence, the stability of neural controllers cannot be guaranteed, a hybrid solution combining classical reliable, but inflexible, controllers and neural controllers should be pursued. The review presented in chapter 3 and the consecutive classification framework, clearly point in this direction.

An accurate forward model of the craft dynamics is potentially important for these hybrid

controllers. In chapter 4, a new identification approach is presented based on the use of neural networks for the identification of individual model parameters. The obtained model is more accurate than state of the art models that rely on incomplete models of the physical mechanisms governing the dynamics of the craft.

Through simulations and experiments the novel approach is shown to be practically applicable.

8.4 Future work

In the work presented, the thruster dynamics were assumed to be known. A further step towards a fully automatic identification algorithm would be to incorporate models for the thrusters and other actuators. For underwater vehicles, the functionality of these actuators can be drastically decreased due to, for example, sea weed in the propeller or a chipped propeller blade. For this reason it would be interesting to also use neural networks to capture the actuator dynamics.

Training of the back propagation neural networks, extensively used in this PhD, is part of ongoing research efforts in the international artificial intelligence society. Of particular interest for the use of neural networks in online control applications, is a proof of stability in both the learning process and the control process.

8.4.1 Proof of stability of neural network learning

A proof of stability for neural network learning is desirable, as it opens up the way for reliable use of online adapting neural networks. As neural network training is typically a non-linear optimisation problem, these proofs are not easily found. However, as in many other areas, Lyapunov stability analysis (Khalil, 2002) may also prove useful in this area. Even if a proof were to be

found for a subset of the used neural networks, this would be highly valuable.

8.4.2 Proof of stability of neural network control

A next step is to develop a framework for stability and robustness analysis of control schemes using neural networks. Acceptance and adaptation by the industry depend on these stability proofs. The main reason for pursuing an augmented controller is a direct result of the current doubts with regards to the stability of neural network controllers. If stability can be guaranteed, controllers can rely on neural networks even more, thus optimally benefiting from the flexibility offered by neural networks.

8.4.3 Training of neural networks with constraints

As certain parameters found in the models for underwater vehicles are symmetric, it may be interesting to constrain the training algorithms to obtain symmetric solutions only. This way, more *a priori* knowledge can be used during the neural network training stage, thus possibly increasing the approximation accuracy of the neural networks.

Bibliography

A. Jovicevic Bekic, D. F., Jovicevic, D., 2004. The use of artificial neural network in identification of women with high risk for breast cancer. European Journal of Cancer Supplements 2, 44–5.

Akkizidis, I., Roberts, G., 1998. Fuzzy modelling and fuzzy-neuro motion control of an autonomous underwater robot. In: 5th Int. Workshop on Advanced Motion Control. pp. 641–46.

Boroushaki, M., Ghofrani, M., Lucas, C., Yazdanpanah, M., 2003. Identification and control of a nuclear reactor core (vver) using recurrent neural networks and fuzzy systems. Nuclear Science, IEEE Transactions on 50, 159–74.

Campa, G., Sharma, M., Calise, A., Innocenti, M., 2000. Neural network augmentation of linear controllers with application to underwater vehicles. In: Proc. of the 2000 American Control Conf. Vol. 1. pp. 75–79.

Cheron, G., Draye, J.-P., Bourgeios, M., Libert, G., 1996. A dynamic neural network identification of electromyography and arm trajectory relationship during complex movements. Biomedical Engineering, IEEE Transactions on 43, 552–558.

Comoglio, R., Pandya, A., 1992. Using a cerebellar model arithmetic computer (cmac) neural network to control an autonomous underwater vehicle. In: Int. Joint Conf. on Neural Networks. Vol. 2. pp. 781–6.

Dorf, R., Bishop, R., 1995. Modern Control Systems, 7th Edition. Addison-Wesley Publishing Company.

Farrell, J., Goldenthal, B., Govindarajan, K., 1990. Connectionist learning control systems: submarine depth control. In: Proc. of the 29th IEEE Conf. on Decision and Control. Vol. 4. pp. 2362–7.

Fossen, T., 1994. Guidance and Control of Underwater Vehicles. John Wiley & Sons.

Fossen, T., 2002. Marine Control Systems. Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles, 1st Edition. Marine Cybernetics, AS.

Fujii, T., Ura, T., 1990. Development of motion control system for auv using neural nets. In: Proc. of the (1990) Symposium on Autonomous Underwater Vehicle Technology. pp. 81–6.

Fujii, T., Ura, T., Kuroda, Y., Chiba, H., Nose, Y., Aramaki, K., 1993. Development of a versatile test-bed ”twin-burger” toward realization of intelligent behaviors of autonomous underwater vehicles. In: Proc. of IEEE OCEANS ’93. pp. 186–91.

Gillard, D., Bollinger, K., 1996. Neural network identification of power system transfer functions. Energy Conversion, IEEE Transactions on 11, 104–10.

Gollub, G., Van Loan, C., 1996. Matrix Computations, 3rd Edition. The Johns Hopkins University Press.

Guo, J., Chiu, F., Wang, C.-C., 1995. Adaptive control of an autonomous underwater vehicle testbed using neural networks. In: Proc. of IEEE OCEANS ’95. Vol. 2. pp. 1033–9.

Gupta, M., Qi, J., 1991. On fuzzy neuron models. In: Proc. Int. Joint. Conf. on Neural Networks. Vol. 2. pp. 431–6.

Hagan, M., Menhaj, M., 1994. Training feedforward networks with the marquardt algorithm. Neural Networks, IEEE Transactions on 5, 989–93.

Hayashi, Y., Buckley, J., Czogola, E., 1992. Fuzzy neural network with fuzzy signals and weights. In: Proc. Int. Joint. Conf. on Neural Networks. Vol. 2. pp. 696–701.

Haykin, S., 1999. Neural Networks, A comprehensive foundation, 2nd Edition. Prentice Hall.

Hong, S. e. a., 2000. Development of technologies for navigation and manipulator system of a semi-autonomous underwater vehicle. Korea Ocean Research & Development Institute, Korea, (in Korean).

Horikawa, S., Furuhashi, T., Uchikawa, Y., 1992. On fuzzy modelling using fuzzy neural networks with the back-propagation algorithm. IEEE Transactions on Neural Networks 3, 801–6.

Hornik, K., Stinchombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Computation 2, 359–66.

Humphreys, D., 1976. Development of the equations of motion and transfer functions for underwater vehicles. Tech. rep., Naval Coastal Systems Center.

Ishii, K., Fujii, T., Ura, T., 1995. An on-line adaptation method in a neural network based control system for auvs. IEEE Journal of Oceanic Engineering 2 (3), 221–8.

Ishii, K., Ura, T., 2000. An adaptive neural-net controller system for an underwater vehicle. Control Engineering Practice 8, 177–84.

Ishii, K., Ura, T., Fujii, T., 1994. A feedforward neural network for identification and adaptive control of autonomous underwater vehicles. In: 1994 IEEE Int. Conf. on Neural Networks. Vol. 5. pp. 3216–21.

Jang, J., 1992. Self-learning fuzzy controllers based on temporal back propagation. IEEE Trans. Neural Networks 3, 714–23.

Kadirkamanathan, V., Chan, C., Cheong, K., 1995. Multiple rbf networks for on-line identification of nonlinear systems. In: Artificial Neural Networks, 1995., Fourth International Conference on. pp. 306–11.

Keller, J., Hunt, D., 1985. Incorporating fuzzy membership functions into the perceptron algorithm. IEEE Trans. Pattern Anal. Mach. Intell. 7, 693–9.

Keller, J., Tahani, H., 1992. Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks. Internation Journal of Approximate Reasoning 6, 221–40.

Khalil, H. K., 2002. Nonlinear systems, 3rd Edition. Prentice Hall.

Kim, T., Yuh, J., 2001. A novel neuro-fuzzy controller for autonomous underwater vehicles. In: Proc. of the 2001 Int. Conf. on Robotics & Automation. pp. 2350–5.

Kodogiannis, P., Lisboa, G., Lucas, J., 1996. Neural network modelling and control for underwater vehicles. Artificial Intelligence in Engineering 1.

Labonte, G., 2002. Fast adaptive control of a non-linear system by an adaline: motion in a fluid. In: Proc. of the 2002 Int. Joint Conf. on Neural Networks. Vol. 2. pp. 1837–41.

Levenberg, K., 1944. A method for the solution of certain non-linear problems in least-squares. Quarterly of Applied Mathematics 2, 164–8.

Li, J.-H., Lee, P.-M., Lee, S.-J., 2002. Neural net based nonlinear adaptive control for autonomous underwater vehicles. In: Proc. of the 2002 IEEE Int. Conf. on Robotics and Automation. Vol. 2. pp. 1075–80.

Lin, C., Lee, C., 1996. Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems. Prentice Hall.

Lin, C., Lee, S., 1991. Neural-network-based fuzzy logic control and decision system. IEEE Trans. Comput. 40, 1320–36.

Ljung, L., 1987. System Identification: Theory for the User, 1st Edition. Prentice Hall.

Marquardt, D., 1963. An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics 11, 431–41.

Matthews, M., Moschytz, G., 1994. The identification of nonlinear discrete-time fading-memory systems using neural network models. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on 41, 740–51.

McKerrow, P., 1993. Introduction to Robotics. Addison-Wesley.

Molnar, L., Toal, D., Flanagan, C., Hayes, M., March 16 - 17 2004. The tethra autonomous underwater vehicle platform: system description and controller development. In: Proc. of 2004 ATUV - Advances in Technology for Underwater Vehicles International Conference. ExCel, London, United Kingdom, pp. 82–7.

Morabito, F., Versaci, M., 2003. Fuzzy neural identification and forecasting techniques to process experimental urban air pollution data. Neural Networks 16, 493–506.

Myers, C., 2005. Man & machine. H_2Ops march, 32–4.

Narendra, K., 1996. Neural networks for control: theory and practice. Proceedings of the IEEE 84, 1385–1406.

Narendra, K., 1997. Neural networks for real-time control. In: Proc. of the 36th Conference on Decision and Control. pp. 1026–31.

Narendra, K., Parthasarathy, K., 1990. Identification and control of dynamical systems using neural networks. IEEE Transactions on Neural Networks 1, 4–27.

Nelles, O., 2001. Nonlinear system identification. Springer Verlag.

Niyogi, P., Girosi, F., Poggio, T., 1998. Incorporating prior information in machine learning by creating virtual examples. Proceedings of the IEEE 86, 2196–2209.

Nolan, S., 2003. The development of testbed vehicles for underwater systems testing and auv controller experimentation. Master's thesis, University of Limerick, Ireland.

Pai, P., Lin, C., 2005. A hybrid arima and support vector machines model in stock price forecasting. Omega 33, 497–505.

Poggio, T., Girosi, F., 1990. Networks for approximation and learning. Proceedings of the IEEE 78, 1481–97.

Pollini, L., Innocenti, M., Nasuti, F., 1997. Robust feedback linearization with neural network for underwater vehicle control. In: Proc. of Oceans '97. MTS/IEEE. Vol. 11. pp. 12–16.

Polycarpou, M. M., Ioannou, P., 1993. Stable nonlinear system identification using neural network models. Neural Networks in Robotics, 165–77.

Reed, R., 1993. Pruning algorithms-a survey. Neural Networks, IEEE Transactions on 4, 740 – 7.

Refsnes, J. E., 2003. Modelling and robust observer design for way-point tracking of underwater vehicle. Master's thesis, Trondheim, Norway.

Refsnes, J. E., Sørensen, A. J., 2004. Design of control system of torpedo shaped rov with experimental results. In: OCEANS '04, MTS/IEEE TECHNO-OCEAN '04. Vol. 1. Kobe, Japan, pp. 264–270.

Rosenblatt, F., 1962. Principles of Neurodynamics. Spartan Books, Washington DC.

Rumelhart, D., Hinton, G., Williams, R., 1986. Learning internal representations by error propagation. MIT Press, Cambridge, Ch. 8.

Seube, N., 1991. Neural network learning rules for control: application to auv tracking control. In: Proc. of the 1991 IEEE Conf. on Neural Networks for Ocean Engineering. pp. 185–96.

Tsoukalas, L. H., Uhrig, R., 1997. Fuzzy and Neural Approaches in Engineering. John Wiley & Sons, New York.

Tsukamoto, C., Lee, W., Yuh, K., Choi, S., Lorentz, J., 1997. Comparison study on advanced thruster control of underwater robots. In: Proc. of the 1997 IEEE Int. Conf. on Robotics and Automation. pp. 1845–50.

Ura, T., Fujii, T., Nose, Y., Kuroda, Y., 1990. Self-organizing control system for underwater vehicles. In: Proc. of OCEANS '90. pp. 76–81.

Van de Ven, P., Flanagan, C., Toal, D., 2004a. Identification of underwater vehicle dynamics with neural networks. In: OCEANS '04 MTS/IEEE, Conference proceedings, November 2004. Vol. 3. pp. 1198–1204.

Van de Ven, P., Flanagan, C., Toal, D., 2005. Neural network control of underwater vehicles. Engineering Applications of Artificial Intelligence 18, 533–547.

Van de Ven, P., Flanagan, C., Toal, D., Lyons, W., 2004b. Underwater vehicle model identification with the aid of neural networks. In: Smart Engineering System Design, Proc. of the Artificial Neural Networks in Engineering Conf.

Van de Ven, P., Johansen, T. A., Sørensen, A. J., Flanagan, C., Toal, D., 2004c. Neural network augmented identification of underwater vehicle models. In: IFAC conference on Control Applications in Marine Systems, July 2004. pp. 263–8.

Venugopal, K., Sudhakar, R., Pandya, A., 1992. On-line learning control of autonomous underwater vehicles using feedforward neural networks. IEEE Journal of Oceanic Engineering 17, 308–319.

WAMIT, 2005. www.wamit.com.

Wang, J., Lee, C., 2002. Self-adaptive recurrent neuro-fuzzy control for an autonomous underwater vehicles. In: Proc. of the 2002 IEEE Int. Conf. on Robotics & Automation. pp. 1095–1100.

Wang, J., Lee, C., Yuh, J., 1999a. An on-line self-organizing neuro-fuzzy control for autonomous underwater vehicles. In: Proc. of the 1999 Int. Conf. on Robotics & Automation. pp. 2416–21.

Wang, J., Lee, C., Yuh, J., 1999b. Self-adaptive neuro-fuzzy control with fuzzy basis function network for autonomous underwater vehicles. In: Proc. of the 1999 IEEE/RSJ Internationational Conf. on Intelligent Robots and Systems. pp. 130–5.

Wang, J., Lee, C., Yuh, J., 2000. Self-adaptive neuro-fuzzy systems with fast parameter learning for autonomous underwater vehicles. In: Proc. of the 2000 IEEE Int. Conf. on Robotics & Automation. pp. 3861–6.

Werbos, P., 1974. Beyond regression: New tools for prediction and analysis in the behavioural science. Ph.D. thesis, Harvard University, Boston.

W.G. Vieira, V.M.L. Santos, F. C. J. P., Fileti, A., 2005. Identification and predictive control of a fcc unit using a mimo neural model. Chemical Engineering and Processing 44, 855–68.

Williams, R., Peng, J., 1990. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Computation 2, 490–501.

Williams, R., Zipser, D., 1989. A learning algorithm for continually running fully recurrent neural networks. Neural Computation 1, 270–80.

Yager, R., 1992. Owa neurons: A new class of fuzzy neurons. In: Proc. of the Int. Joint. Conf. Neural Networks. Vol. 1. pp. 226–31.

Yamamoto, I., 1995. Application of neural network to marine vehicle. In: Proc. of the 1995 IEEE Int. Conf. on Neural Networks. Vol. 1. pp. 220–4.

Yuh, J., 1990. A neural net controller for underwater robotic vehicles. IEEE Journal of Oceanic Engineering 15, issue 3, 161–6.

Yuh, J., 1994. Learning control for underwater robotic vehicles. IEEE Control Systems Magazine 14, issue 2, 39–46.

Yuh, J., Lakshmi, R., 1993. An intelligent control system for remotely operated vehicles. IEEE Journal of Oceanic Engineering 18, issue 1, 55–62.

Appendices

Appendix A

The back propagation algorithm

In the back propagation algorithm, pioneered by Werbos (1974) and rediscovered and popularised by Rumelhart et al. (1986), the gradient is calculated at the output and fed back into the network. In this appendix it will be shown that, using the chain rule of differentiation, a local gradient vector for each neuron in the network can be calculated. Back propagation owes its name to the fact that these local gradients are calculated based on back propagated gradients, calculated at the outputs.

A.1 Derivation of the back propagation algorithm

Since its rediscovery and popularisation for machine learning by (Rumelhart et al., 1986), back propagation has been the most popular learning algorithm for MLPs. It is based on the calculation of output gradients of the network error as a function of the network weights.

In a MLP, input data is fed through the network in a feedforward fashion. As depicted in figure A.1, inputs enter the network in the input layer and are sent through the network towards the output layer. From layer to layer the inputs are multiplied by a weight factor and summed in the neurons of the next layer. In those neurons, of which one is depicted in figure A.2, the weighted inputs are summed. A bias term is added to this sum and a (non)linear function, the so called activation function, is applied to the sum. The result of this sum is then sent to the

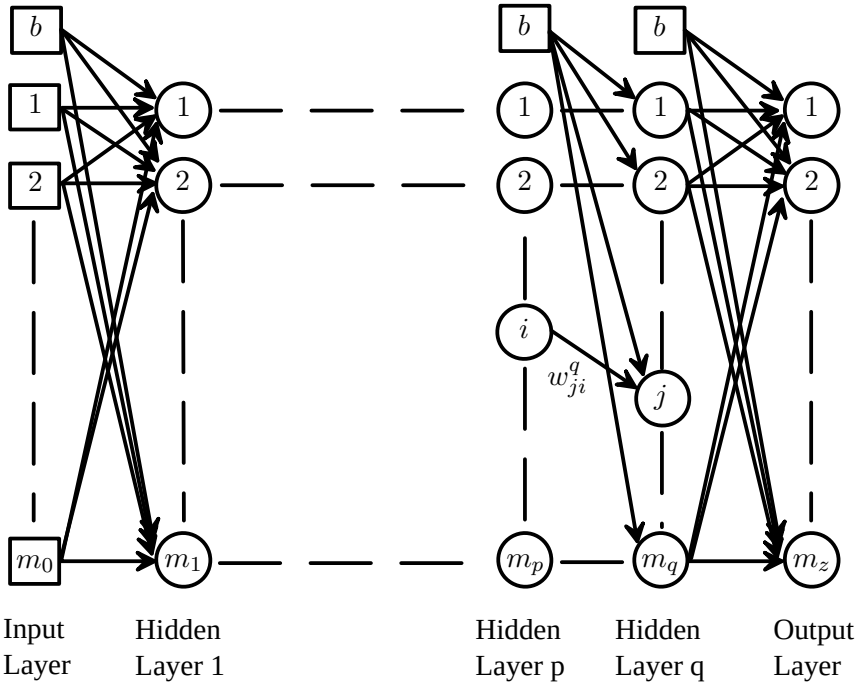


Figure A.1: Multi layer perceptron

next layer. Once the processed input data reaches the output layer, the result is compared to the desired outputs and the error calculated. Based on the gradient of the error with respect to the network's weights, the weights are adjusted such as to walk in a downhill direction on the error surface.

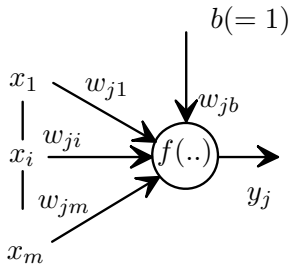


Figure A.2: One neuron from a MLP

For a derivation of the learning rule the error function will first have to be defined. A quadratic form is chosen, yielding the following error at time step n:

$$E(n) = \frac{1}{2}\sum_{k} e_k(n) \qquad k \in C \tag{A.1}$$

with C the set of output neurons and the error for output neuron k:

$$e_k(n) = d_k(n) - y_k(n), \tag{A.2}$$

and

$$y_k(n) = \phi_k(v_k(n)), \tag{A.3}$$

is the output of neuron k with activation function ϕ_k. The argument of the activation function or in other words the sum of all inputs to the neuron, $v_k(n)$ is defined as:

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n), \tag{A.4}$$

where m is the amount of neurons in the previous layer.

The weight update dictated by the back propagation learning algorithm, is based on knowledge of the gradient of the error function with respect to the weights:

$$\Delta w_{ki}(n) = -\eta \frac{\delta E(n)}{\delta w_{ki}(n)}, \tag{A.5}$$

where the index ki indicates the weight from neuron i to neuron k. Using equation A.1, A.2 and A.3, equation A.5 can be rewritten as:

$$\begin{aligned}\Delta w_{ki}(n) &= -\eta\frac{\delta E(n)}{\delta w_{ki}} = \\ &= -\eta\frac{\delta E(n)}{\delta e_k(n)}\frac{\delta e_k(n)}{\delta y_k(n)}\frac{\delta y_k(n)}{\delta v_k(n)}\frac{\delta v_k(n)}{\delta w_{ki}(n)} = -\eta\cdot e_k(n)\cdot -1\cdot\dot{\phi}_k(v_k(n))\cdot y_i(n).\end{aligned} \tag{A.6}$$

Defining the local gradient as

$$\delta_k(n) = -\frac{\delta E(n)}{\delta v_k(n)} = \frac{\delta E(n)}{\delta e_k(n)}\frac{\delta e_k(n)}{\delta y_k(n)}\frac{\delta y_k(n)}{\delta v_k(n)} = e_k(n)\cdot\dot{\phi}_k(v_k(n)), \tag{A.7}$$

yields:

$$\Delta w_{ki}(n) = -\eta\delta_k(n)\cdot y_i(n). \tag{A.8}$$

The local gradient has a physical interpretation as the direction in which the weights should be changed to decrease the output error, but its definition will also aid in the following steps. From equation A.8 it is possible to calculate the weight updates for weights that are directly connected to the output neurons. However, for the hidden weights an alternative formulation, not making explicit use of the output $y_i(n)$, has to be found as this output is not available for the hidden neurons. For this purpose the local gradient δ_k is reformulated:

$$\delta_j(n) = -\frac{\delta E(n)}{\delta v_j(n)} = \frac{\delta E(n)}{\delta y_j(n)}\frac{\delta y_j(n)}{\delta v_j(n)} = -\frac{\delta E(n)}{\delta y_j(n)}\dot{\phi}_j(v_j(n)), \tag{A.9}$$

where the index j has been used to stress the fact that the local gradient pertains to hidden

neurons. Remains to be calculated the partial derivative $\frac{\delta E(n)}{\delta y_j(n)}$.

$$\frac{\delta E(n)}{\delta y_j(n)} = \sum_k e_k \frac{\delta e_k(n)}{\delta y_j(n)} = \sum_k e_k \frac{\delta e_k(n)}{\delta v_k(n)} \frac{\delta v_k(n)}{\delta y_j(n)}, \tag{A.10}$$

where the chain rule for differentiation has been used. From equation A.2 the derivative of the error at neuron j towards $v_k(n)$ can be found:

$$\frac{\delta e_k(n)}{\delta v_k(n)} = -\dot{\phi}_k(v_k(n)). \tag{A.11}$$

The derivative of $v_k(n)$ towards $y_j(n)$ can be found from equation A.4:

$$\frac{\delta v_k(n)}{\delta y_j(n)} = w_{kj}(n). \tag{A.12}$$

Combining equations A.9, A.10, A.11 and A.12 gives the local gradient for an arbitrary hidden neuron j:

$$\delta_j(n) = \dot{\phi}_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{A.13}$$

For an arbitrary neuron the local gradient can be calculated as a weighted sum of all local gradients of neurons connected to the neuron under investigation.

Using equation A.13 in equation A.8 yields the weight update for an arbitrary weight in the neural network as the product of the local gradient of the neuron the weight is connected to and the signal fed forward by the weight:

$$\Delta w_{ji}(n) \quad = \quad -\eta \delta_j(n) \cdot y_i(n). \tag{A.14}$$

Figure A.3 shows the signals used for the update of weight w_{ji}.

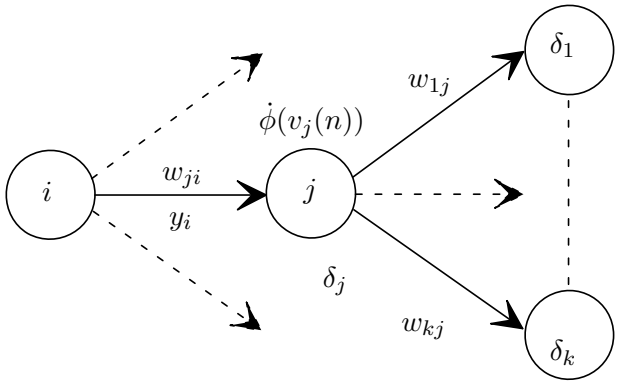


Figure A.3: Neuron signals used in back propagation weight updates

A.1.1 Sequential and batch updates

In the previous derivation of the back propagation algorithm it was assumed that the backward step is performed after presentation of each data sample. As a result the weight update will only be based on the presentation of one data sample and will thus rarely be in the direction that would have been optimum for the whole data set. In other words, sequential back propagation, as the aforementioned described algorithm is referred to, will zig-zag through the error surface towards a minimum. By first presenting the whole data set to the MLP and then deciding on what weight update to perform, the zig-zag course can be overcome. This is called batch updating. The resulting convergence is generally better, but is paid for by being computationally more expensive. Due to its 'jumpy' behaviour, sequential back propagation has the capacity to escape a local minimum whereas batch back propagation will stay in a minimum once it reaches this minimum.

A.2 Optimisation techniques

In this section a brief overview of linear and nonlinear optimisation algorithms will be given. The focus will be on those methods that are used in the various implementations of the back propagation algorithm.

A.2.1 Steepest Descent

Although the gradient of the error function has already been used in the previous without a formal introduction, it is beneficial for the following derivations to formally define this gradient. The gradient ∇E always refers to the gradient of the loss function E with respect to the weight vector $\mathbf{w}$, $\nabla E = \frac{\delta E}{\delta \mathbf{w}}$. As ∇E always points in the direction of greatest ascent, it is intuitively attractive to try and follow the error function in the opposite direction, (locally) the direction of greatest decrease. Hence in steepest descent the parameter update is chosen as:

$$\mathbf{w}(k) = \mathbf{w}(k-1) - \eta(k-1)\nabla E(k-1). \tag{A.15}$$

A.2.2 Newton's method

Steepest descent only makes use of first order derivatives, whereas higher order information might be available. To profit from higher order derivatives the error E is approximated using a second order Taylor series:

$$E(\mathbf{w}+\mathbf{h}) = E(\mathbf{w}) + \mathbf{h}\nabla E + \frac{1}{2}\mathbf{h}\frac{\delta^2 E}{\delta \mathbf{w}\delta \mathbf{w}^T}\mathbf{h}^T + O(\mathbf{h}^3) \tag{A.16}$$

If E from equation A.16 were purely quadratic, the optimum weights would follow straight

from equation A.16. After all for:

$$y = C + \mathbf{x}\mathbf{B} + \frac{1}{2}\mathbf{x}\mathbf{A}\mathbf{x}^T, \tag{A.17}$$

$\mathbf{x}_{opt}$ minimising equation A.17 is given by:

$$\mathbf{x}_{opt} = \mathbf{A}^{-1}\mathbf{B}, \tag{A.18}$$

under the condition that $\mathbf{A}$ is positive definite. Using this result in combination with equation A.16 leads to a parameter update based on both the gradient and the Hessian:

$$\mathbf{w}(k) = \mathbf{w}(k-1) - \eta(k-1)\mathbf{H}^{-1}(k-1)\nabla E(k-1), \tag{A.19}$$

with the Hessian defined as:

$$\mathbf{H}(k) = \frac{\delta^2 E}{\delta \mathbf{w} \delta \mathbf{w}^T}. \tag{A.20}$$

In practice the error surface is only quadratic close to an optimum and thus convergence cannot be expected in one iteration. Other problems are that the Hessian is not guaranteed to be positive definite, especially not for the initial weight matrix, as the latter is chosen more or less randomly. However, if the Hessian is positive definite, quadratic rate of convergence can be expected, which is the fastest rate of convergence normally obtained in nonlinear optimisation algorithms (Nelles, 2001, Section 4.4.4).

A.2.3 Regularisation using weight decay

An intrinsic problem encountered in using the Hessian for weight updates is its poor conditioning. Each eigenvalue of the Hessian corresponds to one of the weight parameters (Nelles, 2001). A high eigenvalue indicates that the corresponding weight parameter has a significant effect on the error. A low eigenvalue means that the corresponding parameter barely affects the error. If, at the end of training, certain parameters still correspond to low eigenvalues, these parameters are superfluous and can be removed from the model, which is called pruning. However, during training they have a negative result on the convergence. The eigenvalues of the Hessian are a measure of the sensitivity of the loss function for the corresponding parameter. This means that the loss function will only decrease very slowly for that particular parameter. A change in this parameter will barely decrease the error and training of the whole network can thus be prolonged simply because of the insignificant parameters. Several methods of regularisation are possible and in this appendix it is not intended to discuss all these methods. However, one method, called weight decay, will be briefly discussed as it is used extensively in practice, including the work presented in this thesis.

In weight decay (Reed, 1993), the ill-conditioning of the Hessian matrix is counteracted by adding a term $\lambda \mathbf{w}^T \mathbf{w}$ to the error function E. This results in the Hessian being equal to:

$$\mathbf{H} = \frac{\delta^2 E}{\delta \mathbf{w} \delta \mathbf{w^T}} + \lambda \mathbf{I}. \tag{A.21}$$

The extra factor added to the diagonal of the Hessian matrix will not affect the eigenvalues with significant influence on the error function. However all low eigenvalues will effectively be set to λ and hence also for these weight parameters a significant weight update will be performed. Even though this does not improve the significance of the corresponding parameter, it does speed

up convergence of the training algorithm.

A.2.4 The Levenberg-Marquardt Algorithm

An algorithm first used for neural network training by Hagan and Menhaj (1994) uses the above described regularisation technique. This training algorithm is based on a nonlinear optimisation algorithm first presented by Levenberg (1944). Almost 20 years later a publication by Marquardt (1963) was the start for the algorithm's present popularity. As mentioned in section A.2.2, to use the Hessian, second order derivatives need to be calculated, increasing the computational complexity of the algorithm considerably. Rather than using the exact Hessian, an approximation is used in the Levenberg-Marquardt algorithm, thus decreasing the computational complexity of obtaining the Hessian (Hagan and Menhaj, 1994). Using an alternative expression for the i^{th} component of the gradient:

$$\nabla E_i = \frac{\delta E}{\delta w_i} = \sum_{k=1}^{N} e(k) \frac{\delta e(k)}{\delta w_i} \tag{A.22}$$

the $(i,j)^{th}$ element of the Hessian can be written as:

$$H(i,j) = \frac{\delta E}{\delta w_i \delta w_j} = \sum_{i=1}^{N} \left(\frac{\delta e(i)}{\delta w_i} \frac{\delta e(i)}{\delta w_j} \right) + \sum_{i=1}^{N} \left(e(i) \frac{\delta^2 e(i)}{\delta w_i \delta w_j} \right). \tag{A.23}$$

Equation A.23, if performed for all elements of the Hessian matrix, equates to:

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} + \sum_{i=1}^{N} e(i) L(i), \tag{A.24}$$

where $L(i)$ denotes the Hessian of $e(i)$.

The jacobian **J**, used in equation A.24, is:

$$\mathbf{J} = \frac{\delta \mathbf{e}}{\delta \mathbf{w}} = \begin{bmatrix} \frac{\delta e_1}{\delta w_1} & \cdots & \frac{\delta e_1}{\delta w_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta e_N}{\delta w_1} & \cdots & \frac{\delta e_N}{\delta w_n} \end{bmatrix}, \tag{A.25}$$

where N denotes the amount of output neurons and n denotes the total amount of weights. Using equation A.25, equation A.24 can be written as:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} + \frac{\delta \mathbf{J}}{\delta \mathbf{w}} \cdot \mathbf{e}. \tag{A.26}$$

The second term in equation A.26 can only be obtained by calculating second order derivatives whereas the first term is readily available. In the Levenberg-Marquardt algorithm, the Hessian is for those reasons, approximated by the first term in equation A.26, and a diagonal matrix λI is added to obtain proper conditioning of the Hessian:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} + \lambda \mathbf{I}. \tag{A.27}$$

Comparing the update given by equation A.28 to the Newton update from equation A.19 clearly shows the alterations made in the Levenberg-Marquardt update.

$$\Delta \mathbf{w} = (\mathbf{J}^T\mathbf{J} + \lambda I)^{-1}\mathbf{J}E. \tag{A.28}$$

Another way of looking at the regularisation term, λ, in equation A.28 is to see it as a term that chooses between a Newton update and a steepest descent update. If the Hessian is poorly conditioned, λ will be chosen large and the update is effectively a steepest descent update. When

the Hessian is conditioned satisfactory, for example around a local or global optimum, λ will be chosen smaller and the update is a Newton update.

The Levenberg-Marquardt algorithm has shown to be a relatively efficient algorithm for batch training of MLPs and is therefore often used. Also in this study the Levenberg-Marquardt algorithm is applied to the MLP training.

Appendix B

Derivation of dynamics model with sliding mass

In Fossen (2002), the rigid-body dynamics are derived by firstly observing that the velocity of the craft's centre of gravity is a sum of the velocity of an arbitrary point within the craft and the vector cross product of the craft's rotational velocity with the vector connecting the two points:

$$\mathbf{v}_c^b = \mathbf{v}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{r}_g^b. \tag{B.1}$$

This velocity can be transformed to the n-frame using the rotation matrix $\mathbf{R}_b^n$:

$$\mathbf{v}_c^n = \mathbf{R}_b^n[\mathbf{v}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{r}_g^b]. \tag{B.2}$$

Taking the derivative towards time while noting that $\mathbf{r}_g^b$ is not constant, yields:

$$\dot{\mathbf{v}}_c^n = \mathbf{R}_b^n[\dot{\mathbf{v}}_o^b + \dot{\boldsymbol{\omega}}_{nb}^b \times \mathbf{r}_g^b + \boldsymbol{\omega}_{nb}^b \times \dot{\mathbf{r}}_g^b + \boldsymbol{\omega}_{nb}^b \times [\mathbf{v}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{r}_g^b]]. \tag{B.3}$$

In equation B.3 the identity $\dot{\mathbf{R}}_b^n[\mathbf{x}] = \mathbf{R}_b^n[\boldsymbol{\omega}_{nb}^b \times \mathbf{x}]$ (see (Fossen, 2002, Theorem 2.2 in Section 2.2.1)) is used.

The acceleration described by equation B.3 can simply be related to forces working on the craft through:

$$m \cdot \dot{\mathbf{v}}_c^n = \mathbf{f}_c^n = \mathbf{R}_b^n \mathbf{f}_c^b, \tag{B.4}$$

where m represents the total mass of the craft. Combining equations B.3 and B.4 and omitting $\mathbf{R}_b^n$ on either side, results in:

$$\mathbf{f}_o^b = \mathbf{f}_c^b = m[\dot{\mathbf{v}}_o^b + \dot{\boldsymbol{\omega}}_{nb}^b \times \mathbf{r}_g^b + \boldsymbol{\omega}_{nb}^b \times \dot{\mathbf{r}}_g^b + \boldsymbol{\omega}_{nb}^b \times [\mathbf{v}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{r}_g^b]]. \tag{B.5}$$

For translational motion the only extra term due to the moving mass is $\boldsymbol{\omega}_{nb}^b \times \dot{\mathbf{r}}_g^b$. For rotational motion a similar derivation, again based on Fossen (2002), is followed. The angular momentum $\mathbf{h}_o^b$ around the origin o can be expressed in a body fixed coordinate system as:

$$\mathbf{h}_o^b = \mathbf{I}_o \boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times \mathbf{v}_o^b, \tag{B.6}$$

and its derivative as:

$$\dot{\mathbf{h}}_o^b = \mathbf{I}_o \dot{\boldsymbol{\omega}}_{nb}^b + \dot{\mathbf{I}}_o \boldsymbol{\omega}_{nb}^b + m\dot{\mathbf{r}}_g^b \times \dot{\mathbf{v}}_o^b + m\dot{\mathbf{r}}_g^b \times \mathbf{v}_o^b. \tag{B.7}$$

In contrast to the truly rigid-body model, $\mathbf{I}_o$ and $\mathbf{r}_g^b$ in equation B.7 are functions of time. Since the relation between the angular momentum about the centre of gravity and an arbitrary point in the body is:

$$\mathbf{h}_o = \mathbf{h}_c + m\mathbf{r}_g \times \mathbf{v}_c, \tag{B.8}$$

equations B.6 and B.7 can be stated in a body-fixed system rotating around the centre of gravity as:

$$\mathbf{h}_c^b = \mathbf{h}_o^b - m\mathbf{r}_g^b \times \mathbf{v}_c^b = \mathbf{I}_o\boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times \mathbf{v}_o^b - m\mathbf{r}_g^b \times \mathbf{v}_c^b, \tag{B.9}$$

and

$$\dot{\mathbf{h}}_c^b = \mathbf{I}_o\dot{\boldsymbol{\omega}}_{nb}^b + \dot{\mathbf{I}}_o\boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times [\dot{\mathbf{v}}_o^b - \dot{\mathbf{v}}_c^b] + m\dot{\mathbf{r}}_g^b \times [\mathbf{v}_o^b - \mathbf{v}_c^b]. \tag{B.10}$$

Again, the matrix $\mathbf{R}_b^n$ can be used to perform a transformation from the body-fixed coordinate frame to the NED frame:

$$\begin{aligned}
\dot{\mathbf{h}}_c^n &= \mathbf{R}_b^n\dot{\mathbf{h}}_c^b + \dot{\mathbf{R}}_b^n\mathbf{h}_c^b \\
&= \mathbf{R}_b^n(\mathbf{I}_o\dot{\boldsymbol{\omega}}_{nb}^b + \dot{\mathbf{I}}_o\boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times [\dot{\mathbf{v}}_o^b - \dot{\mathbf{v}}_c^b] + m\dot{\mathbf{r}}_g^b \times [\mathbf{v}_o^b - \mathbf{v}_c^b] + \\
&+ \boldsymbol{\omega}_{nb}^b \times [\mathbf{I}_o\boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times [\mathbf{v}_o^b - \mathbf{v}_c^b]]),
\end{aligned} \tag{B.11}$$

where again the identity: $\dot{\mathbf{R}}_b^n[\mathbf{x}] = \mathbf{R}_b^n[\boldsymbol{\omega}_{nb}^b \times \mathbf{x}]$ was used. Noting that, if the NED frame is assumed to be an inertial frame,

$$\dot{\mathbf{h}}_c^n = \mathbf{m}_c^n, \tag{B.12}$$

and that $\mathbf{m}_c^n$ can be written in terms of body-fixed variables as:

$$\mathbf{m}_c^n = \mathbf{R}_b^n(\mathbf{m}_o^b - \mathbf{r}_g^b \times \mathbf{f}_c^b), \tag{B.13}$$

an expression for $\mathbf{m}_o^b$ can be found. Substituting equation B.5 in equation B.13 and subsequently substituting the result in equation B.11 yields:

$$\mathbf{I}_o\dot{\boldsymbol{\omega}}^b_{nb} + \dot{\mathbf{I}}_o\boldsymbol{\omega}^b_{nb} + m\mathbf{r}^b_g \times [\dot{\mathbf{v}}^b_o - \dot{\mathbf{v}}^b_c] + m\dot{\mathbf{r}}^b_g \times [\mathbf{v}^b_o - \mathbf{v}^b_c] + \boldsymbol{\omega}^b_{nb} \times [\mathbf{I}_o\boldsymbol{\omega}^b_{nb} + m\mathbf{r}^b_g \times [\mathbf{v}^b_o - \mathbf{v}^b_c]] =$$

$$= \quad \mathbf{m^b_o} - \mathbf{r^b_g} \times \mathbf{m}[\dot{\mathbf{v}}^\mathbf{b}_\mathbf{o} + \dot{\boldsymbol{\omega}}^\mathbf{b}_\mathbf{nb} \times \mathbf{r^b_g} + \boldsymbol{\omega}^\mathbf{b}_\mathbf{nb} \times \dot{\mathbf{r}}^\mathbf{b}_\mathbf{g} + \boldsymbol{\omega}^\mathbf{b}_\mathbf{nb} \times [\mathbf{v}^\mathbf{b}_\mathbf{o} + \boldsymbol{\omega}^\mathbf{b}_\mathbf{nb} \times \mathbf{r}^\mathbf{b}_\mathbf{g}]] \tag{B.14}$$

Rearranging equation B.14 yields:

$$\begin{aligned}\mathbf{m^b_o} \quad &= \quad \mathbf{I}_o\dot{\boldsymbol{\omega}}^b_{nb} + \dot{\mathbf{I}}_o\boldsymbol{\omega}^b_{nb} + m\mathbf{r}^b_g \times [(\underline{\dot{\mathbf{v}}^b_o - \dot{\mathbf{v}}^b_c}) + \dot{\mathbf{v}}^b_o + \underline{\dot{\boldsymbol{\omega}}^b_{nb} \times \mathbf{r}^b_g + \boldsymbol{\omega}^b_{nb} \times \dot{\mathbf{r}}^b_g} + \\ &+ \quad \boldsymbol{\omega}^b_{nb} \times [\mathbf{v}^b_o + \underset{\cdots}{\boldsymbol{\omega}^b_{nb} \times \mathbf{r}^b_g}]] + \boldsymbol{\omega}^b_{nb} \times [\mathbf{I}_o\boldsymbol{\omega}^b_{nb} + \underset{\cdots}{m\mathbf{r}^b_g \times [\mathbf{v}^b_o - \mathbf{v}^b_c]}] + m\dot{\mathbf{r}}^b_g \times [\mathbf{v}^b_o - \mathbf{v}^b_c]\end{aligned} \tag{B.15}$$

Equation B.15 can be somewhat simplified. Taking the derivative of equation B.1 with respect to time gives:

$$\dot{\mathbf{v}}^b_c = \dot{\mathbf{v}}^b_o + \dot{\boldsymbol{\omega}}^b_{nb} \times \mathbf{r}^b_g + \boldsymbol{\omega}^b_{nb} \times \dot{\mathbf{r}}^b_g, \tag{B.16}$$

which can be used to cancel the terms in equation B.15 underlined with a solid line.

For the second simplification the Jacobi identity is used with the substitution $\mathbf{c} = \mathbf{a} \times \mathbf{b}$:

$$\mathbf{a} \times (\mathbf{b} \times (\mathbf{a} \times \mathbf{b})) + \mathbf{b} \times ((\mathbf{a} \times \mathbf{b}) \times \mathbf{a} = \mathbf{0} \tag{B.17}$$

Let $\mathbf{a} = \mathbf{r}^b_g$ and $\mathbf{b} = \boldsymbol{\omega}^b_{nb}$, then the first term in equation B.15 underlined with a dotted line can be expressed as $m\mathbf{a} \times (\mathbf{b} \times (\mathbf{b} \times \mathbf{a}))$ which can be rewritten:

$$m\mathbf{a}\times(\mathbf{b}\times(\mathbf{b}\times\mathbf{a}))=-m\mathbf{a}\times(\mathbf{b}\times(\mathbf{a}\times\mathbf{b}))=m\mathbf{b}\times((\mathbf{a}\times\mathbf{b})\times\mathbf{a})=$$

$$=\quad -m\mathbf{b}\times(\mathbf{a}\times(\mathbf{a}\times\mathbf{b}))=-m\boldsymbol{\omega}_{nb}^{b}\times(\mathbf{r}_{g}^{b}\times(\mathbf{r}_{g}^{b}\times\boldsymbol{\omega}_{nb}^{b})) \tag{B.18}$$

With equation B.1 the last term in equation B.18 exactly cancels the second term in equation B.15 underlined with a dotted line. The one remaining term with $[\mathbf{v}_{o}^{b}-\mathbf{v}_{c}^{b}]$ can be written in terms of a vector cross product, using equation B.1. With these substitutions equation B.15 simplifies to:

$$\begin{aligned}\mathbf{m_o^b} &= \mathbf{I}_{o}\dot{\boldsymbol{\omega}}_{nb}^{b}+\dot{\mathbf{I}}_{o}\boldsymbol{\omega}_{nb}^{b}+m\mathbf{r}_{g}^{b}\times[\dot{\mathbf{v}}_{o}^{b}+\boldsymbol{\omega}_{nb}^{b}\times\mathbf{v}_{o}^{b}]\\ &+ \boldsymbol{\omega}_{nb}^{b}\times\mathbf{I}_{o}\boldsymbol{\omega}_{nb}^{b}+m\dot{\mathbf{r}}_{g}^{b}\times(\mathbf{r}_{g}^{b}\times\boldsymbol{\omega}_{nb}^{b})\end{aligned} \tag{B.19}$$

Now the derivative of the inertia tensor, $\dot{\mathbf{I}}_{o}$, with respect to time remains to be calculated. As the parallel axes theorem states that:

$$\mathbf{I}_{o}=\mathbf{I}_{c}-m\left(\mathbf{r}_{g}^{b}\right)^{2}, \tag{B.20}$$

The derivative of $\mathbf{I}_{o}$ can be written as:

$$\dot{\mathbf{I}}_{o}=\dot{\mathbf{I}}_{c}-2m\mathbf{r}_{g}^{b}\dot{\mathbf{r}}_{g}^{b}. \tag{B.21}$$

The term $\dot{\mathbf{I}}_{c}$ can be found by writing the inertia as a sum of inertias:

$$\mathbf{I}_c = \sum_n \mathbf{I}_{c_n} + \mathbf{I}_{c_b}, \tag{B.22}$$

with $\mathbf{I}_{c_n}$ the inertia of all elements, except from the sliding mass, that make up the Minesniper and thus its inertia matrix. As the only changing term is the inertia due to the sliding battery, this term has been explicitly added in equation B.22. Using the parallel axes theorem once more, equation B.22 can be rewritten as:

$$\mathbf{I}_c = \sum_n \mathbf{I}_{c_n} + \mathbf{I}_{bat} - m_b \left(\mathbf{r}_b\right)^2 \tag{B.23}$$

where $\mathbf{I}_{bat}$ is the battery's inertia matrix in diagonal form, m_b is the battery's mass and $\mathbf{r}_b$ is its distance to the origin c. With equation B.23, $\dot{\mathbf{I}}_c$ can be written as:

$$\dot{\mathbf{I}}_c = -2m_b\mathbf{r}_b\dot{\mathbf{r}}_b \tag{B.24}$$

The position of the centre of gravity and its velocity can be written as:

$$\mathbf{r}_g^b = \frac{m_1\mathbf{r}_1 + m_2\mathbf{r}_2 + \cdots + m_n\mathbf{r}_n}{m} + \frac{m_b}{m}\mathbf{r}_b \tag{B.25}$$

$$\dot{\mathbf{r}}_g^b = \frac{m_b}{m}\dot{\mathbf{r}}_b, \tag{B.26}$$

where $m_i\ i \in 1, 2, ...n$ are all mass elements contained in the craft, apart from the battery mass, and $\mathbf{r}_i\ i \in 1, 2, ...n$ are their positions. $\dot{\mathbf{r}}_b$ Is the velocity of the sliding battery mass. Combining equations B.21, B.24 and B.25, $\dot{\mathbf{I}}_o$ becomes:

$$\dot{\mathbf{I}}_o = -2m_b\dot{\mathbf{r}}_b[\mathbf{r}_b + \mathbf{r}_g^b] \tag{B.27}$$

Now, equation B.5 and B.19 can be combined, obtaining the rigid-body dynamics in six degrees of freedom.

$$\left[\begin{array}{rcl} \mathbf{f_o^b} &=& m[\dot{\mathbf{v}}_o^b + \dot{\boldsymbol{\omega}}_{nb}^b \times \mathbf{r}_g^b + \boldsymbol{\omega}_{nb}^b \times \dot{\mathbf{r}}_g^b + \boldsymbol{\omega}_{nb}^b \times [\mathbf{v}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{r}_g^b]] \\ \mathbf{m_o^b} &=& \mathbf{I}_o\dot{\boldsymbol{\omega}}_{nb}^b - 2m_b\dot{\mathbf{r}}_b[\mathbf{r}_b + \mathbf{r}_g^b]\boldsymbol{\omega}_{nb}^b + m\mathbf{r}_g^b \times [\dot{\mathbf{v}}_o^b + \boldsymbol{\omega}_{nb}^b \times \mathbf{v}_o^b] \\ & & +\boldsymbol{\omega}_{nb}^b \times \mathbf{I}_o\boldsymbol{\omega}_{nb}^b + m\dot{\mathbf{r}}_g^b \times (\mathbf{r}_g^b \times \boldsymbol{\omega}_{nb}^b) \end{array}\right. \tag{B.28}$$

Using the cross product operator S as defined in Fossen (2002):

$$\mathbf{S}(\boldsymbol{\lambda})\mathbf{a} = \boldsymbol{\lambda} \times \mathbf{a}, \tag{B.29}$$

equation B.28 can be written as a product of matrices with vectors:

$$\begin{aligned} & \begin{bmatrix} mI & -mS(\mathbf{r}_g^b) \\ mS(\mathbf{r}_g^b) & \mathbf{I}_o \end{bmatrix} \cdot \begin{bmatrix} \dot{\boldsymbol{\nu}}_o^b \\ \dot{\boldsymbol{\omega}}_{nb}^b \end{bmatrix} + \\ + & \begin{bmatrix} 0^{3\times3} & -mS(\boldsymbol{\nu}_o^b) - mS(\boldsymbol{\omega}_{nb}^b)S(\mathbf{r}_g^b) \\ -mS(\boldsymbol{\nu}_o^b) + mS(\mathbf{r}_g^b)S(\boldsymbol{\omega}_{nb}^b) & -S(\mathbf{I}_o\boldsymbol{\omega}_{nb}^b) \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\nu}_o^b \\ \boldsymbol{\omega}_{nb}^b \end{bmatrix} + \\ & + \begin{bmatrix} 0^{3\times3} & -S(m_b\dot{\mathbf{r}}_b) \\ 0^{3\times3} & -2m_b\dot{\mathbf{r}}_b[\mathbf{r}_b + \mathbf{r}_g^b] + S(m_b\dot{\mathbf{r}}_b)S(\mathbf{r}_g^b) \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\nu}_o^b \\ \boldsymbol{\omega}_{nb}^b \end{bmatrix}, \end{aligned} \tag{B.30}$$

which, following the notation in Fossen (2002), can be written as:

$$\mathbf{M} \cdot \dot{\boldsymbol{\nu}} = \mathbf{C} \cdot \boldsymbol{\nu} + \mathbf{V} \cdot \boldsymbol{\nu} \qquad \mathbf{V} = \begin{bmatrix} 0^{3\times3} & -S(m_b\dot{\mathbf{r}}_b) \\ 0^{3\times3} & -2m_b\dot{\mathbf{r}}_b[\mathbf{r}_b + \mathbf{r}_g^b] + S(m_b\dot{\mathbf{r}}_b)S(\mathbf{r}_g^b) \end{bmatrix} \tag{B.31}$$

Appendix C

Matlab routines

Throughout this work Matlab routines were used for data mining, training of neural networks and validation of the results. As most of these routines were tailor made for the tasks described in this thesis, the essential routines are listed in this appendix.

Data was collected using the virtual world described in section 5.3, made by Dr. Edin Omerdic. This simulator environment was changed to incorporate the model of Tethra, described in section 5.2 and generated in Matlab with the routines *TethraParams.m* and *BuildTethraModel.m*.

Measurements can be impaired with noise, as described in section 5.3.1, with *AddNoise.m*.

Training of the neural networks is performed using the routines *Fwdprop.m*, *Backprop.m* and *TrainLM.m*, which respectively implement forward propagation of signals through a neural network, back propagation of errors through a neural network in order to obtain weight updates or the Jacobian[19], and Levenberg-Marquardt training which uses the first two routines.

Rather than using the Levenberg-Marquardt training algorithm available from the Matlab neural nets toolbox, these algorithms were purposely written for several reasons. The standard routines do not offer the flexibility of using several activation functions in one layer in the neural networks. This, even though not used so far, was envisaged to be beneficial and therefore implemented in the new algorithms. Additionally it was difficult and slow to use the original routines

[19]For standard back propagation, the back propagation algorithm yields weight updates. For the Levenberg-Marquardt algorithm the back propagation algorithm is modified to yield the Jacobian, $\frac{\delta E}{\delta \mathbf{w}}$

in Simulink. By omitting the extensive checks performed in the original routines, robustness is lost, but acceptable speed in Simulink simulations is gained. Lastly, making changes to the original routines is cumbersome and demands a complete overview of the routines. Due to the large amount of subroutines and built-in functions used in the original Matlab code, obtaining such an overview requires a large amount of time.

To extract the learned mass matrix from the neural networks, as described in section 4.3.3, the routine *CheckNeuralM.m* was used. After extraction of the mass matrix, training data for identification of the damping matrices can be obtained using *CalcDv.m*.

Using the above mentioned Levenberg-Marquardt algorithm the neural networks can be trained to represent the damping matrices, as described in section 4.3.4. To obtain insight in what functions are approximated by these neural networks, the learned functions can be extracted and fitted to a quadratic model using *CheckNeuralD.m*. The errors presented in chapter 6 were obtained using *CalcErrors.m*.

```
function [Icg_t, M_t,CG,TotalMass,CB,buoyancy]=TethraParams(AllElements);

% Structure Element
%
%   element = struct('Xdim',0,'Ydim',0,'Zdim',0,Wallthickness,0,'WallMass',0,
%   'InsideMass',0,Shape,'Cylinder','Position',BodyPos);
%   BodyPos = [x y z phi theta psi]';
%
%   Shape can either be 'Cylinder' or 'Box'
%
%*************************************************************************

Nelements = length(AllElements);
P = [0 0 0]';
A = [0 0 0]';
M = 0;
B = 0;
for i = 1:Nelements
    P = [P AllElements(i).Position(1:3,1)];
    A = [A AllElements(i).Position(4:6,1)];
    M = [M (AllElements(i).WallMass+AllElements(i).InsideMass)];
    if AllElements(i).Shape == 'Cylinder'
        B = [B pi*(AllElements(i).Ydim/2)^2*AllElements(i).Xdim];
    else
        if AllElements(i).Shape == 'Box'
            B = [B AllElements(i).Xdim*AllElements(i).Ydim*AllElements(i).Zdim];
        else
            display('Error in shape of element');
            disp(i);
        end
    end
end
P = P(:,2:end);
A = A(:,2:end);
M = M(:,2:end);
B = B(:,2:end);

% Inertia matrices of elements relative to origin inside element
I=cell(Nelements,1);
for i =1:Nelements
    if AllElements(i).Shape == 'Cylinder'
        I(i,1) = {CylinderInertia(AllElements(i).Xdim,AllElements(i).WallThickness,...
            AllElements(i).Ydim-2*AllElements(i).WallThickness,AllElements(i).WallMass,...
            AllElements(i).InsideMass)};
    else
        if AllElements(i).Shape == 'Box'
            I(i,1) = {0};       % to be calculated
        else
            display('Error in shape of element');
            disp(i);
        end
```

```
    end
end

% Calculate centre of gravity
[CG, Mt] = CentreOfGrav(M,P);

% Calculate cenre of buoyancy
[CB,bouyancy] = CentreOfBouy(B,P);

% Total Rigid body Inertia matrices transposed to origin in CG
Icg_t = 0;
for i = 1:Nelements
    Icg_t=Icg_t + InertiaTranspose(InertiaRotate(I{i},A(:,i)), P(:,i), CG, M(i));
end

% Calculation of mass matrices
TotalMass = 0;
for i = 1:Nelements
    TotalMass = TotalMass+M(i);
end

M_rb = Massmatrix(TotalMass,Icg_t,CG);

M_t = M_rb;

% ************************************************************************
%
%                              Function library
%
% ************************************************************************

function [cg,mg]=CentreOfGrav(M,P)
% Calculate centre of gravity in an iterative way taking two point masses
% at a time

cg = P(:,1);
mg  = M(1);
for i = 2:length(M)
    cg = (cg*mg+P(:,i)*M(i))/(mg+M(i));
    mg=mg+M(i);
end

% ************************************************************************
function [cb,bouyancy]=CentreOfBouy(B,P)
% Calculate centre of bouyancy in an iterative way taking two point masses
% at a time

Ro_w = 1025;
cb = P(:,1);
bouyancy  = B(1);
for i = 2:length(B)
```

```
    cb = (cb*bouyancy+P(:,i)*B(i))/(bouyancy+B(i));
    bouyancy=bouyancy+B(i);
end
bouyancy = bouyancy*Ro_w;

% ************************************************************************
function I_c=CylinderInertia(L,Tw,Di,Mw,Mc);

% Calculate Inertia matrix for cylinder with origin in center of cylinder at L/2:
%
% Length:                     L
% Wall thickness:             Tw
% Internal diameter:          Di
% Mass of cylinder wall:      Mw
% Mass of cylinder contents:  Mc

R1 = Di/2;
R2 = R1+Tw;

Ix = .5*Mw*R1^2 + .5*(R1^2+R2^2)*Mc;
Iy = ((1/12)*L^2+.25*R1^2)*Mw+((1/12)*L^2+.25*(R1^2+R2^2))*Mc;
Iz = Iy;
Iyx = 0;
Ixy = Iyx;
Izx = 0;
Ixz = Izx;
Iyz = 0;
Izy = Iyz;

I_c =  [ Ix -Ixy -Ixz;
           -Iyx Iy  -Iyz;
           -Izx -Izy Iz];

% ************************************************************************
function I_new=InertiaTranspose(I_old,OriOld,OriNew,m);
% Transpose Inertia for change of origin

r=OriNew-OriOld;
I_new = I_old - m*(r*r' - r'*r*eye(3));

% ************************************************************************
function I_new=InertiaRotate(I_old,Attitude);
% Transpose Inertia for change of attitude expressed as [phi theta psi]' relative to
% body fixed frame.
% Rotation of coordinate system is according to zyx convention: first rot in z, then
% y then x

phi = Attitude(1);
theta = Attitude(2);
psi = Attitude(3);
```

```
R = [cos(psi)*cos(theta) sin(psi)*cos(theta) -sin(theta);...
    -sin(psi)*cos(phi)+cos(psi)*sin(theta)*sin(phi)
     cos(psi)*cos(phi)+sin(phi)*sin(theta)*sin(psi) cos(theta)*sin(phi);...
     sin(psi)*sin(phi)+cos(psi)*cos(phi)*sin(theta)
    -cos(psi)*sin(phi)+sin(theta)*sin(psi)*cos(phi) cos(theta)*cos(phi)];

I_new = R*I_old*R';

% ********************************************************************************
function I_new=AddedMassTranspose(I,OriOld,OriNew)

r=OriNew-OriOld;
I_new = Hmtrx(r)'*I*Hmtrx(r);

% ********************************************************************************
function M_rb=Massmatrix(m,I_o,r_g);
% Calculate Rigid Body Mass matrix

M_rb =[m*eye(3)     -m*Smtrx(r_g)   ;
       m*Smtrx(r_g)  I_o            ];

% ********************************************************************************
function C_rb = CCmatrix(M,nu);
% Calculate Centripetal - Coriolis matrix from M_rb and speed (Marine
% Control Systems pp. 58 - 59

v1 = nu(1:3,1);
v2 = nu(4:6,1);

M_11 = M(1:3,1:3);
M_12 = M(1:3,4:6);
M_21 = M(4:6,1:3);
M_22 = M(4:6,4:6);

C_rb = [zeros(3,3)              -Smtrx(M_11*v1+M_12*v2);
        -Smtrx(M_11*v1+M_12*v2) -Smtrx(M_21*v1+M_22*v2)];

% ********************************************************************************
function g=RestoringForces(W,B,eta,r_g,r_b)

phi = eta(4);
theta = eta(5);
psi = eta(6);
g=9.81;

% Transformation matrix from n to body
Rn2b = [cos(psi)*cos(theta) sin(psi)*cos(theta) -sin(theta);...
        -sin(psi)*cos(phi)+cos(psi)*sin(theta)*sin(phi)
         cos(psi)*cos(phi)+sin(phi)*sin(theta)*sin(psi) cos(theta)*sin(phi);...
        sin(psi)*sin(phi)+cos(psi)*cos(phi)*sin(theta)
       -cos(psi)*sin(phi)+sin(theta)*sin(psi)*cos(phi) cos(theta)*cos(phi)];
```

```
%In body coordinates
f_b = Rn2b*[0 0 -B*g]';
f_g = Rn2b*[0 0 W*g]';

g=[f_b+f_g; Smtrx(r_g)*f_g+Smtrx(r_b)*r_b];
```

```
function [I,Ma,CG,m,CB,b,Dq]=BuildTethraModel;

% This function builds a model of Tethra based on its simple elements
% (all cylinders).
%
%   I:  Inertia relative to chosen origin
%   CG: Centre of gravity
%   m:  Total mass of craft
%   CB: Centre of bouyancy
%   b:  Total buoyancy

float = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',0,...
    'InsideMass',0,'Shape','Cylinder','Position',0);
hull = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',0,...
    'InsideMass',0,'Shape','Cylinder','Position',0);
frame = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',0,...
    'InsideMass',0,'Shape','Cylinder','Position',0);
endplate = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',...
    0,'InsideMass',0,'Shape','Cylinder','Position',0);
thruster = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',...
    0,'InsideMass',0,'Shape','Cylinder','Position',0);
PickUp = struct('Xdim',0,'Ydim',0,'Zdim',0,'WallThickness',0,'WallMass',0,...
    'InsideMass',0,'Shape','Cylinder','Position',0);

%   BodyPos = [x y z phi theta psi]';

ro_w = 1024;

% Positions and (point)masses of objects in body fixed coordinates,
% origin in centre of lowest hull


% Distance to Dino's origin
Shift_Dino = [0;0;-.43];
% CG_Dino = CG_Pepijn + Shift_Dino;

% Definition of craft

%%%%% Definition of frame


frame.Ydim = 2.5e-2;
frame.Zdim = 2.5e-2;
frame.WallThickness = 1.5e-3;
FrameDensity = 7850;
frame.Shape = 'Cylinder';
Rout = frame.Ydim/2;
Rin = frame.Ydim/2-frame.WallThickness;

Lframe = 0.5;
frame.Xdim = Lframe;
```

```
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [.78,-.249,-.956,-5,0,90]’;
F1 = frame;
frame.Position = [.78,.249,-.956,5,0,90]’;
F2 = frame;
frame.Position = [.25,-.249,-.956,-5,0,90]’;
F3 = frame;
frame.Position = [.25,.249,-.956,5,0,90]’;
F4 = frame;
frame.Position = [-.25,-.249,-.956,-5,0,90]’;
F5 = frame;
frame.Position = [-.25,.249,-.956,5,0,90]’;
F6 = frame;
frame.Position = [-.78,-.249,-.956,-5,0,90]’;
F7 = frame;
frame.Position = [-.78,.249,-.956,5,0,90]’;
F8 = frame;
frame.Position = [0,-.47,0,0,0,0]’;
F11 = frame;
frame.Position = [0,0,0,0,0,0]’;
F12 = frame;
frame.Position = [0,.47,0,0,0,0]’;
F13 = frame;

Lframe = 0.48;
frame.Xdim = Lframe;
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [0,0,-.978,0,0,0]’;
F16 = frame;
frame.Position = [.25,-.516,-.695,0,90,4]’;
F17 = frame;
frame.Position = [-.25,-.516,-.695,0,90,4]’;
F18 = frame;
frame.Position = [.25,.516,-.695,0,90,-4]’;
F23 = frame;
frame.Position = [-.25,.516,-.695,0,90,-4]’;
F24 = frame;
frame.Position = [-.6475,.491,-.701,0,119.6,4]’;
F25 = frame;
frame.Position = [-.6475,-.491,-.701,0,119.6,-4]’;
F27 = frame;
frame.Position = [.6475,.491,-.701,0,60.4,4]’;
F29 = frame;
frame.Position = [.6475,-.491,-.701,0,60.4,-4]’;
F31 = frame;

Lframe = 0.46;
```

```
frame.Xdim = Lframe;
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [.25,-.502,-.228,0,90,-8]';
F19 = frame;
frame.Position = [-.25,-.502,-.228,0,90,-8]';
F20 = frame;
frame.Position = [.25,.502,-.228,0,90,8]';
F21 = frame;
frame.Position = [-.25,.502,-.228,0,90,8]';
F22 = frame;


Lframe = 0.6;
frame.Xdim = Lframe;
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [-.383,.477,-.2335,0,119.6,4]';
F26 = frame;
frame.Position = [-.383,-.477,-.2335,0,119.6,-4]';
F28 = frame;
frame.Position = [.383,.477,-.2335,0,60.4,4]';
F30 = frame;
frame.Position = [.383,-.477,-.2335,0,60.4,-4]';
F32 = frame;

Lframe = 0.94;
frame.Xdim = Lframe;
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [.25,0,0,0,0,90]';
F9 = frame;
frame.Position = [-.25,0,0,0,0,90]';
F10 = frame;

Lframe = 1.56;
frame.Xdim = Lframe;
frame.WallMass = FrameDensity*pi*(Routˆ2-Rinˆ2)*Lframe;
frame.InsideMass = ro_w*pi*Rinˆ2*Lframe;

frame.Position = [0,-.498,-.934,0,0,0]';
F14 = frame;
frame.Position = [0,.498,-.934,0,0,0]';
F15 = frame;

FrameSet = [F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16...
            F17 F18 F19 F20 F21 F22 F23 F24 F25 F26 F27 F28 F29 F30 F31 F32];
```

```
%%%%% Definition of floats


float.Ydim = 0.16;
float.Zdim = 0.16;
float.WallThickness = .4e-2;
TubeDensity = 1500;
FloatDensity = 150;
Rout = float.Ydim/2;
Rin = float.Ydim/2-float.WallThickness;
float.Shape = 'Cylinder';


Lfloat = 1.98;
float.Xdim = Lfloat;
float.WallMass = TubeDensity*pi*(Rout^2-Rin^2)*Lfloat;
float.InsideMass = FloatDensity*pi*Rin^2*Lfloat;
float.Position = [0,.578,-.934,0,0,0]';
FL1 = float;

float.Position = [0,-.578,-.934,0,0,0]';
FL2 = float;

% Percentage of float flooded
pO = .40;
pFl = 1-pO;

Lfloat = 1.75;
float.Xdim = Lfloat;
float.WallMass = TubeDensity*pi*(Rout^2-Rin^2)*Lfloat;
float.InsideMass = pFl*FloatDensity*pi*Rin^2*Lfloat+pO*ro_w*pi*Rin^2*Lfloat;
float.Position = [0,.498,-1.084,0,0,0]';
FL3 = float;

float.Position = [0,-.498,-1.084,0,0,0]';
FL4 = float;

FloatSet = [FL1 FL2 FL3 FL4];

hull.Xdim = 0.48;
hull.Ydim = 0.3050;
hull.Zdim = 0.3050;
hull.WallThickness = 0.4e-2;
HullDensity = 7850;
Rout = hull.Ydim/2;
Rin = hull.Ydim/2-hull.WallThickness;
hull.WallMass = HullDensity*pi*(Rout^2-Rin^2)*hull.Xdim;
hull.Shape = 'Cylinder';

hull.InsideMass = 30;
hull.Position = [0,0,-.692,0,0,0]';
```

```
H1 = hull;

hull.InsideMass = 30;
hull.Position = [0,0,-.231,0,0,0]';
H2 = hull;

HullSet = [H1 H2];

endplate.Xdim = 0.5e-2;
endplate.Ydim = 0.3950;
endplate.Zdim = 0.3950;
endplate.WallThickness = .3950/2;
endplateDensity = 7850;
Rout = endplate.Ydim/2;
Rin = endplate.Ydim/2-endplate.WallThickness;
endplate.WallMass = endplateDensity*pi*(Rout^2-Rin^2)*endplate.Xdim;
endplate.Shape = 'Cylinder';

endplate.InsideMass = 0;
endplate.Position = [.25,0,-.692,0,0,0]';
E1 = endplate;

endplate.InsideMass = 0;
endplate.Position = [-.25,0,-.692,0,0,0]';
E2 = endplate;

endplate.InsideMass = 0;
endplate.Position = [-.25,0,-.231,0,0,0]';
E3 = endplate;

EndplateSet = [E1 E2 E3];

thruster.Xdim = .30;
thruster.Ydim = 0.085;
thruster.Zdim = 0.085;
thruster.WallThickness = 0;
thruster.WallMass = 0;
thruster.Shape = 'Cylinder';

thruster.InsideMass = 10;
thruster.Position = [0,.35,-.430,0,0,0]';
T1 = thruster;

thruster.Position = [0,-.35,-.430,0,0,0]';
T2 = thruster;

thruster.Position = [0,.28,-.730,0,90,0]';
T3 = thruster;

thruster.Position = [0,-.28,-.730,0,90,0]';
T4 = thruster;
```

```
ThrusterSet = [T1 T2 T3 T4];

PickUp.Xdim = 1;
PickUp.Ydim = 0.35;
PickUp.Zdim = 0.35;
PickUp.WallThickness = .1;
PickUp.WallMass = 30;
PickUp.Shape = 'Cylinder';

PickUp.InsideMass = 30;
PickUp.Position = [1,-1,-1,0,0,0]';

P1 = PickUp;
AllElement = [FrameSet FloatSet HullSet EndplateSet ThrusterSet];

% ************************************************************

[I,Mt,CG,m,CB,b]=TethraParams(AllElement);
CG = CG-Shift_Dino;
CB = CB-Shift_Dino;
Dq = [100 100 100 30 30 30];
```

```
function [Nu_noise, Nudot_noise, Force_noise]=AddNoise(Nu_ppm,Nudot_ppm,Tau_ppm, G_ppm);

% This function adds noise to the measurements for Nu, Nudot and the input
% forces

L = length(Nu_ppm);

% Noise on the force measurements
ForceRatio = 0.5*[1 1 1 0.1 0.1 0.1]';
ForceNoise = repmat(ForceRatio,1,L).*(ones(6,L)-2*rand(6,L));

% Noise on the acceleration measurements
AccRatio = 0.5*[1 1 1 0.1 0.1 0.1]';
AccNoise = repmat(AccRatio,1,L).*(ones(6,L)-2*rand(6,L));

% Noise on the velocity measurements
VelRatio = 0.1*ones(6,1);
VelNoise = repmat(VelRatio,1,L).*(ones(6,L)-2*rand(6,L));

Nu_noise    = Nu_ppm+abs(Nu_ppm).*VelNoise;
Nudot_noise = Nudot_ppm+abs(Nudot_ppm).*AccNoise;
Force_noise   = (Tau_ppm-G_ppm)+abs(Tau_ppm-G_ppm).*ForceNoise;
```

```
function [net ,error] = FWDprop(net)

%FWDprop executes the forward propagation of signals and returns the
%new layer outputs and the errors
%inputs:        net             = network information
%               data            = [Inputs vs time; Outputs vs time]

% structure net:
%   net.Layers.Neurons: row vector with number of nodes (including bias)
%                       for each layer
%   net.Layers.ActieFie:1xn cell (where n is number of layers) with activation
%                       function per node per layer
%   net.Layers.Weights: nxn cell (where n is number of layers) with weight matrix from
%                       layer x to layer y in position {y,x}
%   net.Bias:           1xn row vector which indicates per layer if bias is
%                       used (1=yes, 0=no)
%   net.Connect:        nxn matrix where element (y,x) determines whether
%                       there are connections from layer x to layer y
%   net.Signals:        lxts matrix where l = amount of neurons and ts is
%                       amount of timepoints. Each column holds the outputs
%                       of each neuron for a time point.

% shortcuts

Neurons = net.Layers.Neurons;
ActieFie = net.Layers.ActieFie;
W = net.Layers.Weights;
Connect = net.Connect;

N_noninput = sum(Neurons(1,2:end));
N_input = Neurons(1,1);
N_output = Neurons(1,end);

% Checks on data

% if size(Data,1) ~= N_input+N_output;
%     disp('Either inputs or outputs are not suited for amount of neurons')
% end

% range of hyperbolic tangent function
Xrange = 1;%2/3;
Yrange = 1;%1.7159;

% Find amount of data samples
Ts = size(net.Signals.V,2);

% Find amount of layers
L = size(Neurons,2);

% Fill matrix with numbers of Neurons per layer
N = Neurons;
```

```
% Fill matrix with activation functions L = 1, H = 2, S = 3, no node (fill up) = 0

F = zeros(max(N,[],2),L);
for i=1:L
    F(find(ActieFie{i,1}=='L'),i) = 1;
    F(find(ActieFie{i,1}=='H'),i) = 2;
    F(find(ActieFie{i,1}=='S'),i) = 3;
end
% Change F into column format
dummy = [];
for i=1:L
    index = find(F(:,i)==0);
    if isempty(index)==1
        dummy = [dummy; F(:,i)];
    else
        dummy = [dummy; F(1:index(1)-1,i)];
    end
end
F=dummy;

Vold = net.Signals.V;
Vnew = Vold;

% Fill array with number of weights per layer
N_W = zeros(1,L);
for i=1:L;
    for j=1:L
        if Connect(i,j) == 1
            N_W(1,j)=N_W(1,j)+Neurons(1,j)*(Neurons(1,i)-1);
        end
    end
end

% compute layer outputs per layer.
for layerprop = 1:L-1
    StartW = 1; %Start counting from first hidden node
    StartV=1;
    Cnt = 0;
    Vcnt = N_input;
    for ToLayer=2:L   %It is assumed no (recurrent) connections to layer 1 are made
        for FromLayer=1:L
            if Connect(ToLayer,FromLayer) == 1
                for ToNeuron = 2:Neurons(1,ToLayer) % start from 2 as first neuron is the b
                        N_fromlayer = Neurons(1,FromLayer);
                        VNewPos = sum(Neurons(1,1:ToLayer-1))+ToNeuron;
                        StartV = sum(Neurons(1,1:FromLayer-1))+1;
                        StopV = sum(Neurons(1,1:FromLayer));
                        StartW = sum(N_W(1,1:FromLayer-1))+(ToNeuron-2)*Neurons(1,FromLayer
                        StopW = sum(N_W(1,1:FromLayer-1))+(ToNeuron-1)*Neurons(1,FromLayer)
                        Vnew(VNewPos,:)=W(StartW:StopW,1)'*[Vold(StartV:StopV,:)];
```

```
                end
            end
        end
    end
    LinearNodes=find(F==1);
    TanhNodes=find(F==2);
    SigmoidNodes=find(F==3);
    Vold(LinearNodes,:)=Vnew(LinearNodes,:);
    Vold(TanhNodes,:)=HTan(Vnew(TanhNodes,:), Xrange, Yrange);
    Vold(SigmoidNodes,:)=Sigmoid(Vnew(SigmoidNodes,:));
    % set bias nodes to one
    for Layer=1:L
        Vold(sum(Neurons(1,1:Layer-1))+1,:)=1;
    end
end
Vnew = Vold;

% In error the factor 1/2 has been taken out to obtain the same error as in
% NNtool

output = Vnew(end-(N_output-2):end,:);
des = net.Signals.Outputs;
error = (des-output);
net.Signals.V = Vnew;
function y=Sigmoid(x)
y=1./(1+exp(-x));

function ans=HTan(x,Xrange,Yrange)
ans = Yrange*tanh(Xrange*x);
```

```
function [Output] = BackProp(net,result);

%BackProp executes the backward propagation of signals and returns the
%weight updates and jacobian
%inputs:        net             = network information
%               data            = [Inputs vs time; Outputs vs time]

% structure net:
%   net.Layers.Neurons: row vector with number of nodes (including bias)
%                       for each layer
%   net.Layers.ActieFie:1xn cell (where n is number of layers) with activation function
%                       per node per layer
%   net.Layers.Weights: nxn cell (where n is number of layers) with weight matrix from
%                       layer x to layer y in position {y,x}
%   net.Bias:           1xn row vector which indicates per layer if bias is
%                       used (1=yes, 0=no)
%   net.Connect:        nxn matrix where element (y,x) determines whether
%                       there are connections from layer x to layer y
%   net.Signals:        lxts matrix where l = amount of neurons and ts is
%                       amount of timepoints. Each column holds the outputs
%                       of each neuron for a time point.


% shortcuts

Neurons = net.Layers.Neurons;
ActieFie = net.Layers.ActieFie;
W = net.Layers.Weights;
V= net.Signals.V;
Connect = net.Connect;

N_noninput = sum(Neurons(1,2:end));
N_input = Neurons(1,1);
N_output = Neurons(1,end);

% range of hyperbolic tangent function
Xrange = 1;%2/3;
Yrange = 1;%1.7159;

% Find amount of data samples
Ts = size(V,2);

% Find amount of layers
L = size(Neurons,2);

% Fill matrix with numbers of Neurons per layer
N = Neurons;

% Fill matrix with activation functions L = 1, H = 2, S = 3, no node (fill up) = 0
F = zeros(max(N,[],2),L);
for i=1:L
```

```
    F(find(ActieFie{i,1}=='L'),i) = 'L';
    F(find(ActieFie{i,1}=='H'),i) = 'H';
    F(find(ActieFie{i,1}=='S'),i) = 'S';
end

% Change F into column format. '0' in F only fills up the matrix to a
% square matrix. '0''s in F are not included in the column version of F.
dummy = [];
for i=1:L
    index = find(F(:,i)==0);
    if isempty(index)==1
        dummy = [dummy; F(:,i)];
    else
        dummy = [dummy; F(1:index(1)-1,i)];
    end
end
F=dummy;

% Backpropagation of signals

delta = zeros(sum(N),Ts);

LinearNodes=find(F==1);
TanhNodes=find(F==2);
SigmoidNodes=find(F==3);

%determine delta at output
if result == 'Wei'  % Calculation of delta's for weight updates
    for i=sum(N,2):-1:sum(N,2)-(N_output-2)    % -1 for the bias
        out=V(i,:);
        error = net.Signals.Outputs(i-(sum(Neurons(1,1:L-1))+1),:)-out;
        switch(F(i,1))
            case 'L'        %linear activation function
                delta(i,:) = -ones(1,Ts).*error;
            case 'S'        %sigmoid activation function
                delta(i,:) = -(out.*(1-out)).*error;
            case 'H'        %Hyperbol tan activation function
                delta(i,:) = -((Xrange/Yrange)*(Yrange*(ones(1,Ts)) - out)...
                            .*(Yrange*ones(1,Ts) + out)).*error;
        end
    end
else                % Calculation of delta's for Jacobian update
    for i=sum(N,2):-1:sum(N,2)-(N_output-2)
        out=V(i,:);
        switch(F(i,1))
            case 'L'        %linear activation function
                delta(i,:) = -ones(1,Ts);
            case 'S'        %sigmoid activation function
                delta(i,:) = -(out.*(1-out));
            case 'H'        %Hyperbol tan activation function
                delta(i,:) = -(Xrange/Yrange)*(Yrange*(ones(1,Ts)) - out)...
```

```
                            .*(Yrange*ones(1,Ts) + out);
        end
    end
end


% Determine delta's in backprop fashion
% For the moment only for feedforward connections...

Nodes = sum(N);

Wpos1 = size(W,1);
for Layer = L-1:-1:2
    FirstDeltaNextLayer = sum(Neurons(1,1:Layer));
    for Neuron = Neurons(1,Layer):-1:1
        DeltaPos = sum(Neurons(1,1:Layer-1))+Neuron;
        N_nextlayer = Neurons(1,Layer+1);
        N_thislayer = Neurons(1,Layer);
        Wpos2=Wpos1;
        for i=N_nextlayer:-1:1
            switch F(DeltaPos,1);
                case 'L'    %linear activation function
                    delta(DeltaPos,:)=delta(DeltaPos,:)+W(Wpos2,1)...
                                      *delta(FirstDeltaNextLayer+i,:);
                case 'S'    %sigmoid activation function
                    delta(DeltaPos,:)=delta(DeltaPos,:)+V(DeltaPos,:)...
                        *(ones(1,Ts)-V(DeltaPos,:)).*W(Wpos2,1)*delta(FirstDeltaNextLayer+i
                case 'H'    %Hyperbol tan activation function
                    delta(DeltaPos,:)=delta(DeltaPos,:)+(Xrange/Yrange)...
                        *(Yrange*ones(1,Ts)-V(DeltaPos,:)).*(Yrange*ones(1,Ts)+V(DeltaPos,
                       .*(W(Wpos2,1)*delta(FirstDeltaNextLayer+i,:));
            end
            Wpos2 = Wpos2-(N_thislayer+1);      % 1 for the bias
        end
        Wpos1 = Wpos1-1;
    end
end

% Fill array with number of weights per layer
N_W = zeros(1,L);
for i=1:L;
    for j=1:L
        if Connect(i,j) == 1
            N_W(1,j)=N_W(1,j)+Neurons(1,j)*(Neurons(1,i)-1);
        end
    end
end

%Calculate Output which is either dWeights or the Jacobian

Output = zeros(size(W,1),Ts);
```

```
for FromLayer = 1:L
    for ToLayer = 1:L
        if Connect(ToLayer,FromLayer) == 1
            for ToNeuron= 2:Neurons(1,ToLayer)
                for FromNeuron=1:Neurons(1,FromLayer)
                    WeightPos = sum(N_W(1,1:FromLayer))-N_W(1,FromLayer)+(ToNeuron-2)...
                                *(Neurons(1,FromLayer))+FromNeuron;
                    DeltaPos = sum(Neurons(1,1:FromLayer))+ToNeuron;
                    VPos = sum(Neurons(1,1:FromLayer-1))+FromNeuron;
                    Output(WeightPos,:) = (delta(DeltaPos,:).*net.Signals.V(VPos,:));
                end
            end
        end
    end
end
```

```
function [net,delta]=TrainLM(TrainIn,TrainOut,epochs,net,DOF,pos);

% structure net:
%   net.Layers.Neurons: row vector with number of nodes (including bias)
%                       for each layer
%   net.Layers.ActieFie:1xn cell (where n is number of layers) with activation
%                       function per node per layer
%   net.Layers.Weights: column vector of weight parameters
%   net.Bias:           1xn row vector which indicates per layer if bias is
%                       used (1=yes, 0=no)
%   net.Connect:        nxn matrix where element (y,x) determines whether
%                       there are connections from layer x to layer y
%   net.Signals:        lxts matrix where l = amount of neurons and ts is
%                       amount of timepoints. Each column holds the outputs
%                       of each neuron for a time point.

% TrainOut can be supplied as a 6xts matrix. In such a case 'DOF' chooses
% the degree of freedom (ranging from 1 to 6) to be trained



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INIT %%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isstruct(net) ==0
    Weights = cell(3,3);

    AFie = cell(3,1);
    %make 2*DOF linear input nodes plus 1 node as a bias
    if DOF == 6
        AFie{1,1} = ['LLLLLLLLLLLL'];
    end
    if DOF == 4
        AFie{1,1} = ['LLLLLLLLL'];
    end
    if DOF == 1
        AFie{1,1} = ['LLL'];
    end
    AFie{2,1} = ['LHHHLL']; % hidden layer has 5 sigmoid nodes and one linear bias
    AFie{3,1} = ['LL']; %output has one linear output node plus one linear bias
    Layers = struct('Neurons',{[2*DOF+1 6 2]}, 'ActieFie',{AFie}, 'Weights', {[]});
    % 'Neurons' includes a bias for every layer

    signals = struct('V',{[]},'Outputs',{[]});
    net = struct('Layers', {Layers}, 'Bias', {[1 1 1]},...
                 'Connect',{[0 0 0 ; 1 0 0; 0 1 0]},'Signals',{signals});
end

Ts =size(TrainIn,2);
Neurons = net.Layers.Neurons;
Connect = net.Connect;
V = net.Signals.V;
```

```
W=net.Layers.Weights;

% Initiate weight matrices randomly

% Fill weight vector W with all weight matrices <> 0.
% Order is W{1,1} W{1,2} ... W{1,N} W{2,1}... W{2,N}
L = size(net.Layers.Neurons,2);

W = [];
for ToLayer = 1:L
    for FromLayer=1:L
        if net.Connect(ToLayer,FromLayer) == 1
            elements = (Neurons(ToLayer)-1)*Neurons(FromLayer);
            W = [W; (ones(elements,1)-2*rand(elements,1))];
        end
    end
end
net.Layers.Weights = W;

% % Fill array with number of weights per layer
% N_W = zeros(1,L);
% for i=1:L;
%     for j=1:L
%         if Connect(i,j) == 1
%             N_W(1,j)=N_W(1,j)+Neurons(1,j)*(Neurons(1,i)-1);
%         end
%     end
% end

% Simulation parameters
epoch = 0;
dim = length(W)
delta = 1e-3;               %LM factor
delta_max=1e20;
kappa = 10;                  %increase of LM factor

Tstart=fix(clock);

% Initialisation of error vector

net2=net;
% Compute error and output
net.Signals.V=[zeros(sum(Neurons,2),Ts)];       %clear
net.Signals.V(2:size(TrainIn,1)+1,:)=TrainIn;   % load inputs
% set bias to 1
for Layer=1:L
    BiasPos = sum(Neurons(1,1:Layer-1))+1;
    net.Signals.V(BiasPos,:)=1;
end
net.Signals.Outputs=TrainOut(pos,:);            %load outputs
[net,E]=FWDprop(net);
```

```
ErrorOld=(E*E')/Ts;

% epoch wise network training

for epoch=1:epochs
    epoch

    % Compute jacobian
    [JacobianT] = BackProp(net,'Jac');
    Hess = JacobianT*JacobianT';
    Grad = JacobianT*E';


    % Levenberg Marquardt iteration
    while delta<=delta_max
        %Compute weight updates
        dW = -(Hess+delta*diag(ones(dim,1)))\Grad;
        %dW = -inv(Hess+delta*diag(ones(dim,1)))*Grad;
        net2.Layers.Weights = W+dW;

        % Compute error and output
        net2.Signals.V=[zeros(sum(Neurons,2),Ts)];      %clear
        net2.Signals.V(2:size(TrainIn,1)+1,:)=TrainIn;  %load inputs
        % set bias to 1
        for Layer=1:L
            BiasPos = sum(Neurons(1,1:Layer-1))+1;
            net2.Signals.V(BiasPos,:)=1;
        end
        net2.Signals.Outputs=TrainOut(pos,:);           %load outputs
        [net2,E2]=FWDprop(net2);
        Error = (E2*E2')/Ts;

        if Error<ErrorOld             % error is decreasing
            delta=delta/kappa;
            ErrorOld=Error;
            net=net2;
            W=net.Layers.Weights;
            E=E2;
            break
        else
            delta=delta*kappa;
        end
    end
end
Tend = fix(clock);
fprintf('Network training duration %2i.%2i.%2i\n\n',...
    Tend(4)-Tstart(4),Tend(5)-Tstart(5),Tend(6)-Tstart(6));

SumError = 0;
    % Compute error and output
    net.Signals.V=[zeros(sum(Neurons,2),Ts)];       %clear
```

```
net.Signals.V(2:size(TrainIn,1)+1,:)=TrainIn;   %load inputs
% set bias to 1
for Layer=1:L
    BiasPos = sum(Neurons(1,1:Layer-1))+1;
    net.Signals.V(BiasPos,:)=ones(1,Ts);
end
net.Signals.Outputs=TrainOut(pos,:);            %load outputs
[net,E]=FWDprop(net);

Error=(E*E')/Ts
```

```
function Mneural = CheckNeuralM(Input,Output, netU, netV, netW, netP, netQ, netR);

% This function extracts the mass matrix from the neural networks, Mneural, using a
% least squares algorithm. As the velocity is made negligible the mass
% matrix can be obtained from: Nudot = inv(M)*Tau
%
%   Input: [Nu_ppm; Force_ppm]
%   Output:[Nudot_ppm]
%   netX: neural network for DOF X

CtrlInfo = [12  5 1;...                 %neurons per layer
            0  0 1;...                  %output only from last layer
            1  1 1;...                  %bias, 1 if bias used
            0  2 0;...                  %0 = linear, 1 Sigmoid, 2 = Htan
            1  1 1;...                  %max x of activation function
            1 1 1];                     %max y of activation function

CtrlConnect = [0 0 0;...
               1 0 0;...                %layer 1 connects to 2
               0 1 0];                  %layer 2 connects to 3

[UOld, UNew, UWeights, dUWeights] = MyInit(CtrlInfo, CtrlConnect);
[VOld, VNew, VWeights, dVWeights] = MyInit(CtrlInfo, CtrlConnect);
[WOld, WNew, WWeights, dWWeights] = MyInit(CtrlInfo, CtrlConnect);
[POld, PNew, PWeights, dPWeights] = MyInit(CtrlInfo, CtrlConnect);
[QOld, QNew, QWeights, dQWeights] = MyInit(CtrlInfo, CtrlConnect);
[ROld, RNew, RWeights, dRWeights] = MyInit(CtrlInfo, CtrlConnect);

% loading weight parameters

UWeights{2,1} = [netU.b{1,1} netU.IW{1,1}];
UWeights{3,2} = [netU.b{2,1} netU.LW{2,1}];

VWeights{2,1} = [netV.b{1,1} netV.IW{1,1}];
VWeights{3,2} = [netV.b{2,1} netV.LW{2,1}];

WWeights{2,1} = [netW.b{1,1} netW.IW{1,1}];
WWeights{3,2} = [netW.b{2,1} netW.LW{2,1}];

PWeights{2,1} = [netP.b{1,1} netP.IW{1,1}];
PWeights{3,2} = [netP.b{2,1} netP.LW{2,1}];

QWeights{2,1} = [netQ.b{1,1} netQ.IW{1,1}];
QWeights{3,2} = [netQ.b{2,1} netQ.LW{2,1}];

RWeights{2,1} = [netR.b{1,1} netR.IW{1,1}];
RWeights{3,2} = [netR.b{2,1} netR.LW{2,1}];

InvM = [];
MeanNu = mean(Input(1:6,:),2);
TauInputs = Input(7:12,:);
```

```
Inputs = [repmat(MeanNu,1,length(TauInputs)); TauInputs];
for i=1:length(Inputs)
%                           FWDprop(inp,des,Weights,LayerOutputs,Yrange, ActivationFie)
    [UOld, UNew ,error] = FWDprop([Inputs(:,i)],1,UWeights,UOld, UNew, CtrlInfo);
    DvU = UNew{end}(2:end);
%                           FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [VOld, VNew ,error] = FWDprop([Inputs(:,i)],1,VWeights,VOld, VNew, CtrlInfo);
    DvV = VNew{end}(2:end);
%                           FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [WOld, WNew ,error] = FWDprop([Inputs(:,i)],1,WWeights,WOld, WNew, CtrlInfo);
    DvW = WNew{end}(2:end);
%                           FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [POld, PNew ,error] = FWDprop([Inputs(:,i)],1,PWeights,POld, PNew, CtrlInfo);
    DvP = PNew{end}(2:end);
%                           FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [QOld, QNew ,error] = FWDprop([Inputs(:,i)],1,QWeights,QOld, QNew, CtrlInfo);
    DvQ = QNew{end}(2:end);
%                           FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [ROld, RNew ,error] = FWDprop([Inputs(:,i)],1,RWeights,ROld, RNew, CtrlInfo);
    DvR = RNew{end}(2:end);

    InvM = [InvM [DvU DvV DvW DvP DvQ DvR]'];
end

%creation of identification model

Ts = 0.1;
DOF = 6;
y1 = InvM;
u1 = TauInputs;

dat = iddata(y1',u1',Ts);

Mmodel = [1 0 0 0 1 0;...
          0 1 0 1 0 1;...
          0 0 1 0 1 0;...
          0 1 0 1 1 1;...
          1 0 1 1 1 0;...
          0 1 0 1 0 1];

% obtain LS estimate of Mass matrix using constrained optimisation
%model1=arx(dat,[diag(zeros(DOF,1)) Mmodel zeros(DOF,DOF)]);

% obtain LS estimate of Mass matrix
model1=arx(dat,[diag(zeros(DOF,1)) ones(DOF,DOF) zeros(DOF,DOF)]);
[A1,B1]=arxdata(model1);
Mneural = inv(B1);
```

```
function [TrainIn, TrainOut]=CalcDv(Nu_ppm,Nudot_ppm,Force,M);

% This function makes the training data for training the damping neural
% networks: D*Nu = M*(Nudot_cpm-Nudot_ppm)
%
% Nudot_cpm = inv(M)*(Force-C*Nu)
% Nudot_ppm = acceleration as measured
%
% (Force = Tau-g)

% Sampletime
Ts = .1;

TrainIn = [];
TrainOut = [];
InvM = inv(M);

for i=1:length(Nu_ppm)-1
    % Calculation of cpm model update, Nudot_cpm
    C = m2c(M,Nu_ppm(:,i));
    Nudot_cpm(:,i) = InvM*(Force(:,i)-InvM*C*Nu_ppm(:,i));
    % Calculation of damping*state vector
    Dx=M*(Nudot_cpm(:,i)-Nudot_ppm(:,i));
    TrainIn = [TrainIn Nu_ppm(:,i)];
    TrainOut = [TrainOut Dx];
end

% Plots
T=[0:Ts:(length(TrainIn)-1)*Ts];
figure;
subplot(6,1,1);
plot(T,TrainIn(1,:),'r');
subplot(6,1,2);
plot(T,TrainIn(2,:),'r');
subplot(6,1,3);
plot(T,TrainIn(3,:),'r');
subplot(6,1,4);
plot(T,TrainIn(4,:),'r');
subplot(6,1,5);
plot(T,TrainIn(5,:),'r');
subplot(6,1,6);
plot(T,TrainIn(6,:),'r');

figure;
subplot(6,1,1);
plot(T,TrainOut(1,:),'r');
subplot(6,1,2);
plot(T,TrainOut(2,:),'r');
subplot(6,1,3);
plot(T,TrainOut(3,:),'r');
subplot(6,1,4);
```

```
plot(T,TrainOut(4,:),'r');
subplot(6,1,5);
plot(T,TrainOut(5,:),'r');
subplot(6,1,6);
plot(T,TrainOut(6,:),'r');
```

```
function [DneuralLin, DneuralQuad]=CheckNeuralD(DataIn, netU, netV, netW, netP, netQ, netR

% This function extracts LS estimates for the linear and quadratic drag matrix.
%
% DataIn: Nu_ppm
% netX: neural network for DOF X

CtrlInfo = [12  5 1;...                 %neurons per layer
            0  0 1;...                  %output only from last layer
            1  1 1;...                  %bias, 1 if bias used
            2  2 0;...                  %0 = linear, 1 Sigmoid, 2 = Htan
            1  1 1;...                  %max x of activation function
            1 1 1];                     %max y of activation function

CtrlConnect = [0 0 0;...
               1 0 0;...                %layer 1 connects to 2
               0 1 0];                  %layer 2 connects to 3

[UOld, UNew, UWeights, dUWeights] = MyInit(CtrlInfo, CtrlConnect);
[VOld, VNew, VWeights, dVWeights] = MyInit(CtrlInfo, CtrlConnect);
[WOld, WNew, WWeights, dWWeights] = MyInit(CtrlInfo, CtrlConnect);
[POld, PNew, PWeights, dPWeights] = MyInit(CtrlInfo, CtrlConnect);
[QOld, QNew, QWeights, dQWeights] = MyInit(CtrlInfo, CtrlConnect);
[ROld, RNew, RWeights, dRWeights] = MyInit(CtrlInfo, CtrlConnect);

% loading weight parameters

UWeights{2,1} = [netU.b{1,1} netU.IW{1,1}];
UWeights{3,2} = [netU.b{2,1} netU.LW{2,1}];

VWeights{2,1} = [netV.b{1,1} netV.IW{1,1}];
VWeights{3,2} = [netV.b{2,1} netV.LW{2,1}];

WWeights{2,1} = [netW.b{1,1} netW.IW{1,1}];
WWeights{3,2} = [netW.b{2,1} netW.LW{2,1}];

PWeights{2,1} = [netP.b{1,1} netP.IW{1,1}];
PWeights{3,2} = [netP.b{2,1} netP.LW{2,1}];

QWeights{2,1} = [netQ.b{1,1} netQ.IW{1,1}];
QWeights{3,2} = [netQ.b{2,1} netQ.LW{2,1}];

RWeights{2,1} = [netR.b{1,1} netR.IW{1,1}];
RWeights{3,2} = [netR.b{2,1} netR.LW{2,1}];

Inputs = [DataIn; abs(DataIn)];
Dv = [];
for i = 1:length(Inputs)
%                           FWDprop(inp,des,Weights,LayerOutputs,Yrange, ActivationFie)
    [UOld, UNew ,error] = FWDprop(Inputs(:,i),1,UWeights,UOld, UNew, CtrlInfo);
    DvU = UNew{end}(2:end);
```

```
%                          FWDprop(inp,des,Weights,LayerOutputs,Yrange, ActivationFie)
    [VOld, VNew ,error] = FWDprop(Inputs(:,i),1,VWeights,VOld, VNew, CtrlInfo);
    DvV = VNew{end}(2:end);
%                          FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [WOld, WNew ,error] = FWDprop(Inputs(:,i),1,WWeights,WOld, WNew, CtrlInfo);
    DvW = WNew{end}(2:end);
%                          FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [POld, PNew ,error] = FWDprop(Inputs(:,i),1,PWeights,POld, PNew, CtrlInfo);
    DvP = PNew{end}(2:end);
%                          FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [QOld, QNew ,error] = FWDprop(Inputs(:,i),1,QWeights,QOld, QNew, CtrlInfo);
    DvQ = QNew{end}(2:end);
%                          FWDprop(inp,des,Weights,LayerOVtputs,Yrange, ActivationFie)
    [ROld, RNew ,error] = FWDprop(Inputs(:,i),1,RWeights,ROld, RNew, CtrlInfo);
    DvR = RNew{end}(2:end);

    Dv = [Dv [DvU DvV DvW DvP DvQ DvR]'];
end

%creation of identification model

Ts = 0.1;
DOF=6;
y1 = Dv;
u1 = [DataIn; abs(DataIn).*DataIn];

dat = iddata(y1',u1',Ts);

model1=arx(dat,[diag(zeros(DOF,1)) [ones(DOF,DOF) diag(ones(DOF,1))] zeros(DOF,2*DOF)]);
[A1,B1]=arxdata(model1);

DneuralLin = B1(:,1:6);
DneuralQuad = B1(:,7:end);
```

```
function [DiagError, OffDiagError, ZeroError, MeanError]=CalcErrors(MatModel, MatTest);

% This function calculates the following errors given a reference and a
% test matrix:
%
%   DiagError    =  relative error of the diagonal elements
%   OffDiagError =  relative error of off-diagonal elements if the
%                   corresponding element in the reference matrix is non-zero
%   ZeroError    =  error of elements that are zero in the reference matrix
%                   but non-zero in
%                   the test matrixrelative to largest element in the reference
%                   matrix
%   MeanError    =  Mean value of all relative errors of elements that are non-zero
%                   in the reference matrix

DiagModel = diag(MatModel);
DiagTest = diag(MatTest);

DiagError = 0;
for i=1:length(DiagModel)
    if DiagModel(i)~=0
        Error = abs((DiagModel(i)-DiagTest(i))/DiagModel(i));
        if Error > DiagError
            DiagError = Error;
        end
    end
end

OffDiagError = 0;
OffDiagModel = MatModel-diag(DiagModel);
OffDiagTest = MatTest-diag(DiagTest);
for i=1:size(MatModel,1)
    for j=1:size(MatModel,2)
        if OffDiagModel(i,j)~=0 & abs(OffDiagModel(i,j))
            Error = abs((OffDiagModel(i,j)-OffDiagTest(i,j))/OffDiagModel(i,j));
            if Error > OffDiagError
                OffDiagError = Error;
            end
        end
    end
end

ZeroError = 0;
for i=1:size(MatModel,1)
    for j=1:size(MatModel,2)
        if MatModel(i,j)==0
            Error = abs((OffDiagTest(i,j))/max(max(MatModel)));
            if Error > ZeroError
                ZeroError = Error;
            end
        end
```

```
    end
end

MeanError = 0;
NonZeroElementCounter = 0;
for i=1:size(MatModel,1)
    for j=1:size(MatModel,2)
        if MatModel(i,j)~=0 & MatModel(i,j)
            MeanError = MeanError+abs((MatTest(i,j)-MatModel(i,j))/MatModel(i,j));
            NonZeroElementCounter = NonZeroElementCounter+1;
        end
    end
end
MeanError = MeanError/NonZeroElementCounter;

DiagError
OffDiagError
ZeroError
```

Appendix D

Publications

List of Publications

Conference papers

Van de Ven, P.W.J., Flanagan C., Toal D., 2003. A Survey of AI techniques for Control of Underwater Vehicles. In: Proceedings of GCUV2003, 1st IFAC Workshop on Guidance and Control of Underwater Vehicles. pp. 149 - 154.

Van de Ven, P.W.J., Flanagan C., Toal D., 2003. Artificial Intelligence for the Control of Underwater Vehicles. In: Smart Engineering System Design, Proc. of the Artificial Neural Networks in Engineering Conf. pp. 559 - 564.

Van de Ven, P.W.J., Johansen T.A., Sørensen A.J., Flanagan, C., Toal, D., 2004. Neural network augmented identification of underwater vehicle models. In: Proceedings of the IFAC Conference on Control and Applications in Marine Systems. pp. 263 - 268.

Van de Ven, P.W.J., Flanagan C., Toal D., Omerdic, E., Lyons, W.B., 2004. Underwater Vehicle Model Identification with the aid of Neural Networks. In: Smart Engineering System Design, Proc. of the Artificial Neural Networks in Engineering Conf. pp. 355 - 360.

Van de Ven, P.W.J., Flanagan C., Toal D., 2003. Identification and Control of Underwater Vehicles with the Aid of Neural Networks. In: Conference proceedings of Oceans '04. pp. 1198 - 1204.

Van de Ven, P.W.J., Flanagan C., Toal D., Omerdic, E., 2004., Underwater Vehicle Model Identification with the aid of Neural Networks. In: Proceedings of the 2004 IEEE Conference on Robotics, Automation and Mechatronics. pp. 428 - 433.

Van de Ven, P.W.J., J. Refsnes, T.A. Johansen, A.J. Sørensen, C. Flanagan and D. Toal., 2005. Identification of the damping parameters of a torpedo shaped underwater vehicle using Radial Basis Functions. To be submitted to: Symposium on System Identification 2006.

Journal papers

Van de Ven, P.W.J., Flanagan C., Toal D., 2005. Neural network control of underwater vehicles. In: Engineering Applications of Artificial Intelligence 18. pp. 533 - 547.

Van de Ven, P.W.J., Johansen T.A. , Sørensen A.J., Flanagan, C., Toal, D., 2005. Neural network augmented identification of underwater vehicle models. Accepted for: Control Engineering Practice.

Key Published Papers