UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Implementation of a neurofuzzy control strategy in a heterogeneous system architecture.

POR

Piero Alessandro Riva Riquelme

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Electrónico

Profesores Guía
Dr. Daniel Sbarbaro Hofer
Dr. Miguel Figueroa Toro

Profesional Supervisor
Dr. Jorge Pezoa Núñez

Abril 2021
Concepción (Chile)

Sumario

Las redes neuronales son poderosas herramientas para el reconocimiento de patrones a la cabeza de aplicaciones en Ingeniería Eléctrica junto a data mining, big data, data science o procesamiento de señales. Esta habilidad de regresión puede ser utilizada en sistemas de control.

En esta tesis se llevó a cabo un trabajo de diseño y programación de algoritmos para la implementación de una estrategia de control empleando redes neuronales. Dada la adaptación y entrenamiento de redes neuronales, estas pueden ofrecer versatilidad en estrategias de control con el fin de crear poderosas herramientas en el control de procesos.

Múltiples programas MATLAB fueron desarrollados para estudiar la estrategia de control propuesta y sus subsecciones: 1) Operación y entrenamiento de la red neuronal, 2) el sistema a controlar y 3) la estrategia de control.

En primer lugar, una estructura neurodifusa llamada ANFIS fue seleccionada para ser la red neuronal empleada; una estructura sinérgica compuesta por lógica difusa en forma de FIS y redes adaptativas que ha demostrado ser más efectiva que ANNs al modelar superficies no lineales o para predecir series de tiempo caóticas. Se desarrolló un programa MATLAB que entrena la ANFIS para su uso como el modelo NARX de un sistema SISO lineal (motor DC) con una extensión a sistemas MIMO no lineales. Se elaboró un segundo programa MATLAB que simulara el modelo dinámico de 6 grados de libertad de un dirigible de pequeña escala por ser un sistema MIMO no lineal y de fase no mínima. Se escribió un último programa MATLAB para la estrategia de control implementada, utiliza el modelo NARX de la velocidad del eje x del dirigible simulado en la estrategia de control predictiva basada en el modelo Dynamic Matrix Control (DMC).

Resultados muestran que cuando ANFIS es entrenada con un set de entrenamiento seleccionado y el controlador es sintonizado bajo métodos heurísticos puede optimizar sus recursos (3 actuadores) para reducir el error en una forma mesurada y seguir la referencia correctamente. Todos los programas anteriores fueron traducidos a un ambiente C++ para mejorar la velocidad de cómputo y que estuvieran en este formato en la web.

Con el fin de extender las limitaciones de la estrategia de control propuesta se diseñó un kernel de CUDA C++ para calcular ANFIS en una arquitectura heterogénea usando una GPU NVIDIA. Se desarrolló un programa CUDA C++ para implementar la estrategia de control es este ambiente. Resultados muestran usos optimistas cuando se necesitan enormes estructuras de ANFIS para el control rápido de sistemas más complejos como los biológicos o robóticos.

Summary

Neural Networks are powerful pattern recognition structures in the head of today's electrical engineering community for intelligent applications in data mining, big data, data science or image processing, it is this ability that can be exploited for control purposes.

In this thesis the design and code of a series of scripts for the implementation of a control strategy employing neural networks was carried out. The adaptiveness and training characteristic of NN can bring the control strategy with versatility to be a powerful tool for the control of processes.

Multiple MATLAB scripts were developed to study the proposed control strategy and its subsections: 1) neural network operation and training, 2) the system to be controlled and 3) the control strategy.

Firstly, a neurofuzzy structure named ANFIS was selected to be the employed NN; a synergistic structure composed by fuzzy logic and adaptive networks that have shown to be more effective than Artificial Neural Networks when modeling nonlinear surfaces or to predict chaotic time series. A MATLAB script was developed to train the ANFIS for its use as the NARX model of a linear SISO system (DC motor) with an extension to nonlinear MIMO systems. A second MATLAB script was elaborated for the simulation of the 6 degree of freedom (DOF) dynamic model of a low scale blimp for being a non-minimum phase nonlinear MIMO system. A final MATLAB script was coded for the implemented control strategy, it uses the trained NARX x-axis speed model of the simulated blimp in the model-based predictive control strategy Dynamic Matrix Control.

Results show that when ANFIS is trained with selected training data and the controller is fine tuned under heuristic methods it can optimize its resources (3 actuators) to reduce the error in a measured way and follow the given reference correctly. All mentioned scripts were translated into a C++ environment to improve computation speeds and to be available in this format in the web.

To extend the limitations of the proposed control strategy, a CUDA C++ kernel was designed for computing the ANFIS structure in a heterogeneous architecture using an NVIDIA's GPU. A CUDA C++ parallel program was developed to implement the control strategy in this environment. Results show enthusiastic uses when enormous ANFIS structures are needed for the fast control of more complex systems like biological or robotic.

A mi familia y amigos, Jotapes, Esteban, Matata, Josefita, Pipe, Diegazzo, Elayas, Dano y en general a los equipos BGS y VDC que en estos tiempos de pandemia nos hemos apoyado para superar nuestros desafíos. A mis profesores guías que me ofrecieron nuevas perspectivas y orientación.

Aprovechando la oportunidad, saludar especialmente al todo cuerpo docente que me ha acompañado hasta el momento de escribir estos agradecimientos para la defensa de tesis...

¡Muchas gracias!

Contents

List of Tables

List of Figures

Nomenclature

ANFIS

α	General symbol for a parameter in the consequent or premise functions.
$\bar{w}_i$	ANFIS 3-rd layer output.
η	Parameter defining the change in membresy function parameters.
$\mu_A(x)$	Membership function A of x.
E_p	Measured error for the p-th data pair.
f	ANFIS output.
f_i	ANFIS 4-th layer weight.
J_p	Cost function for the p-th data pair.
K	Error decrease step.
$O^k_{i,p}$	Output of the i-th node in the k-th layer given training data p.
$O^L_{M,P}$	Output of the last node M in the last layer L given the last training data P.
T_p	ANFIS target value for the p-th data pair.
w_i	ANFIS 2-nd layer output.

Blimp model

δ	Thrust difference input for both motors after transformation of inputs.
$\dot{X}$	State vector containing the 6 accelerations.
$\lambda_{i,j}$	DCM elements.
μ	Motors elevation angle.
ϕ, θ, ψ	Orientation of the blimp along the ϕ, θ, ψ spherical axes.
ρ_{air}	Density of air at NTP.
ρ_{He}	Density of Helium at NTP.

A Aerodynamic forces vector.

a, b, c Ellipsoid dimension parameters.

ax, ay, az CG displacement along the x, y, z axes from the CV.

B Buoyancy forces vector.

C_l, C_m, Cn Aerodynamic coefficients along the x, y, z axes angular velocities.

CD, CY, CL Aerodynamic coefficients along the x, y, z axes linear velocities.

dx, dy, dz Motor displacement along the x, y, z axes from the CV.

e Lamb's coefficient.

F_d Dynamic forces vector.

G Gravitational forces vector.

g Acceleration of gravity at sealevel.

I_x, I_y, I_z Inertias of the blimp along the x, y, z axes.

$L_{\dot{p}}$, $M_{\dot{q}}$, $N_{\dot{r}}$ Virtual masses along the x, y, z axes angular accelerations.

M Masses and inertia matrix.

P propulsion forces vector.

p, q, r Angular velocities along the x, y, z axes.

T_{dp} Thrust in the port motor.

T_{ds} Thrust in the starboard motor.

T_{max} Thrust input for both motors after transformation of inputs.

U Input vector containing T_{ds}, T_{dp}, μ.

u, v, w Linear velocities along the x, y, z axes.

UT Input vector containing T_{max}, δ, μ.

V_n, V_e, V_u Linear velocities along the North, East, Up earth axes.

Vol Volume of the blimp.

W Weight forces vector.

X State vector containing the 6 velocities u, v, w, p, q, r.

x, y, z Position of the blimp along the x, y, z axes.

$X_{\dot{u}}$, $Y_{\dot{v}}$, $Z_{\dot{w}}$ Virtual masses along the x, y, z axes linear accelerations.

Y State vector containing the 6 positions x, y, z, ϕ, θ, ψ.

Acronyms

ANFIS Adaptive Neuro Fuzzy Inference System. 31

APE Average Percentage Error. 15

CG Center of gravity. 33

CV Center of volume. xii, 33, 37

DCM Direction Cosine Matrix. xi, 32, 35

DMC Dynamic Matrix Control. 31

NED North-East-Down. 35

CHAPTER 1

Introduction

1.1 General introduction

In today's Electrical Engineering community there has been an increased interest in topics like *big data*, *data science*, *machine learning* and *artificial intelligence* as a response to the evolving industry and problem solving for the modern society. Rising applications related to these research disciplines are being solved by hardware implementations and heterogeneous computing due to their high computing requirements.

Neural networks are known for being a powerful tool for pattern recognition and prediction models. It is in this way that their potential can be exploited for control systems. There is an ample amount of information involving control strategies employing neural networks in the literature, however, most of them only have theoretical support and few have been applied in practical uses.

This work focuses on implementing a control strategy with the use of a fuzzy neural network called ANFIS, a NN that employs fuzzy logic and *fuzzy reasoning* for pattern recognition, attaining better results than conventional NNs. Therefore, given the high computing needs of this particular network, a script implementation is done using parallel computing on a heterogeneous CPU + GPU architecture: NVIDIA's CUDA.

The control strategy proposed for this work uses a trained ANFIS NARX model of the process to make predictions for its use in Dynamic Matrix Control.

One of the main aims of this work is to control a non-linear MIMO system, in order to solve complex problems with complex solutions. In this case, the process under scrutiny is the 6 DOF dynamic model of a low scale blimp.

This thesis is organized as follows, chapter 2 explains all of the fundamentals behind this work: the ANFIS network, training and operation; Dynamic Matrix Control, formulation and its use with an ANFIS network; Computer architecture & parallel programming and ANFIS CUDA calculation; finally the Dynamic model of a blimp, fundamentals and equations. Chapter 3 contains the results of this work: ANFIS training, blimp simulations and control results. Chapter 4 comprises the discussion and conclusions of this study.

1.2 Previous works

1.2.1 ANFIS

- J. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *in IEEE Transactions on Systems, Man , and Cybernetics, Vol. 23, No. 3, pp. 665-685*, May-June 1993.

This work presents a fuzzy inference system implemented in the framework of adaptive networks called ANFIS. This network can construct an input-output mapping based on human knowledge in the form of fuzzy if-then rules and stipulated IO data pairs. The ANFIS architecture can be employed to model nonlinear functions for applications from non-linear regression to model identification to automatic control. With the use of Takagi-Sugeno fuzzy if-then rules within adaptive networks, fuzzy membership functions can be adjusted through learning algorithms and capture the spirit of a "rule of thumb" used by humans for pattern recognition. Results show that ANFIS can effectively model a highly nonlinear surface compared to artificial neural networks.

- J. R. Jang, "Input selection for ANFIS learning," *Proceedings of IEEE 5th International Fuzzy Systems, Vol. 2, pp. 1493-1499*, New Orleans, La, USA, 1996.

This work presents a quick and straightforward way of input selection for neuro-fuzzy modeling using ANFIS. The purposes of input selection include removing noise/irrelevant inputs, remove inputs that depends on other inputs, make the underlying model more concise and transparent and reduce the time for model construction. The proposed input selection method is based on the assumption that the ANFIS model with the smallest RMSE after one epoch of training has greater potential of achieving a lower RMSE when given more epochs of training. An important conclusion of input selection is that it can provide an effective means to determining input priorities for ANFIS modeling.

1.2.2 Control strategies

- K. J. Hunt, D. Sbarbaro, R. Zbikowski, P.J. Gawthrop, "Neural networks for control systems - a survey," *Automatica, Vol 28, Issue 6*, 1992, pp 1083-1112.

A variety of neural network architectures in control systems are surveyed in this work with the aim to serve as a contribution to present the basic ideas and techniques of artificial neural networks in notation familiar to control engineers. System representation and modelling by ANN is discussed and applied to nonlinear and adaptive control, finally a number of control structures (including Internal Model Control) are reviewed.

- J. R. Jang, CT. Sun, "Neuro-fuzzy modeling and control," *in Proceedings of the IEEE, Vol. 83, No. 3, pp. 378-406,*, March 1995.

This work is focused in design methods for ANFIS in both modeling and control applications. It is introduced with basic concepts of fuzzy sets, fuzzy reasoning, fuzzy if-then rules and fuzzy inference systems, goes through adaptive networks and their learning rules to ANFIS

architecture and finally a number of design techniques for fuzzy and neural controllers are described; they are listed next: A. Mimicking another working controller, B. Inverse control, C. Specialized learning control, D. Back-propagation through time and real time recurrent learning, E. Feedback linearization and sliding control, F. Gain scheduling, G. Other control schemes.

♣ E. Ita Essien, "Adaptive NeuroFuzzy Inference System (ANFIS) Based Model Predictive Control (MPC) for Carbon Dioxide Reforming Of Methane (CDRM) In A Plug Flow Tubular Reactor For Hydrogen Production," *A Thesis submitted to the Faculty of Graduate Studies and Research In Partial Fulfillment of the Requirements for the Degree of Master of Applied Science in Industrial Systems Engineering*, University of Regina, Canada, 2012.

This master of applied science thesis studies the control of a pilot-scale reformer for the production of hydrogen (H_2) extracted during carbon dioxide (CO_2) capture for its use as a sustainable energy source. The control strategy is based on a model predictive control scheme with the use of ANFIS as model for controlling the temperature in a reformer at the thermodynamically desired level. Results of this work indicate that the ANFIS model is able to accurately replicate the response of the process to changes in temperature.

1.2.3 Blimp dynamics

♣ S. B. V. Gomes, "An investigation of the flight dynamics of airships with application to the YEZ-2A," *PhD. thesis, College of aeronautics, Cranfield University*, 1990.

In this PhD thesis, an extensive investigation involving wind tunnel testing was carried out in order to construct a flight dynamics computer simulation model of airship flight with a high degree of fidelity in the aerodynamic modelling. The 6 degree of freedom (6 DOF) non-linear model was written in the simulation language ACSL and its based on wind tunnel generated data for the YEZ-2A airship in the form of look up tables for the calculation of aerodynamic forces given the shape, size and position of fins and rudders of the YEZ-2A airship. This work was considered the most comprehensive airship aerodynamics database existing at the year 1998 and due to its success it was used in one of the first internet based airship simulators and in the Bedford Advanced Flight Simulator.

1.2.4 Discussion

A deep search of a neural-based control strategy for its practical implementation was carried out. Many authors manifest that it is possible to use neural networks for control purposes and control strategies schemes are presented. These have shown to posses only theoretical support or at most indicate the way NN *could* be employed in control systems and only a few (1) posses solid practical background.

A neural network structure must be chosen for the purposes of this work. In this case it draws a lot of attention a neurofuzzy network known as ANFIS for being more efficient than artificial neural networks and for being a synergistic structure that combines fuzzy logic with adaptive networks.

A process of great complexity (as in nonlinearities or multiple inputs or numerous degrees of freedom) must be the test subject for the development and implementation of the control strategy to have a real motive as to solve complex problems with complex solutions. In this case the 6DOF dynamic model of a small scale blimp was studied and is treated as the guinea pig for this thesis.

Given the above, this work focuses in designing and coding neurofuzzy (ANFIS) control simulations in a practical implementation for the 6DOF dynamic model of a blimp to be available in the form of a GitHub repository for any control engineer with the same interest in these complex control schemes for any given application. This thesis also intends to extend this control strategy script not only to C++ (programming language used in many microcontrollers) but also to an heterogeneous architecture through the programming of a CUDA GPU in the form of parallel programming.

1.3 Hypothesis

- Neural network's powerful pattern recognition characteristic can be used in a control strategy (as in control engineering) implementation for solving high complexity problems without the need of knowing the explicit formulas involved in the model of a process.

1.4 Objectives

1.4.1 General objective

To implement a control strategy with the use of neural networks for a high complexity process.

1.4.2 Specific objectives

- Design and code a Matlab script that trains a neural network.
- Design and code a Matlab script that controls a high complexity process with neural networks.
- Design and code an accelerated algorithm that replicates the Matlab script.
- Implement the control algorithm in parallel programming.

1.5 Scope and limitations

- Due to COVID-19, this work is limited only to simulations. For the same reason, the controlled process is simulated by computer and does not consider perturbations.
- Although today's computers have extensive memory blocks, there will always be practical limitations. In particular, the neural network size will be limited to the heterogeneous system's memory.

CHAPTER 2

Theory

This chapter sets the theoretical base of this work, it presents all of the formulas, equations and their derivations used in the Matlab/CUDA scripts. It is organized as follows,

- **Section 2.1** explains the ANFIS network, its operation, fuzzy reasoning, iterative gradient descent training, size generalization, and comments.
- **Section 2.2** describes the working principle and formulation of the DMC controller, as well as the use of ANFIS in this control strategy.
- **Section 2.3** explains topics about CUDA GPU and the use of parallel programming for an ANFIS CUDA implementation.
- **Section 2.4** profoundly describe the 6DOF dynamic equations of a small scale blimp: aircraft concepts and forces acting on the blimp and their foundations.

2.1 Adaptive Neuro Fuzzy Inference System (ANFIS)

A fuzzy inference system (FIS) employing fuzzy IF - THEN rules can model the qualitative aspects of human knowledge and reasoning processes without employing precise quantitative analyses [8]. ANFIS is an adaptive network that it is embedded to a fuzzy inference system, this structure can serve as a basis for constructing a set of fuzzy if-then rules with appropriate membership functions to generate stipulated input-output pairs.

Figure 2.1 shows the RMSE (root mean squared error) curves for a 2-18-1 neural network and the ANFIS for modeling a two-input nonlinear function *sinc*,

$$z = sinc(x, y) = \frac{sin(x)}{x} \times \frac{sin(y)}{y}$$

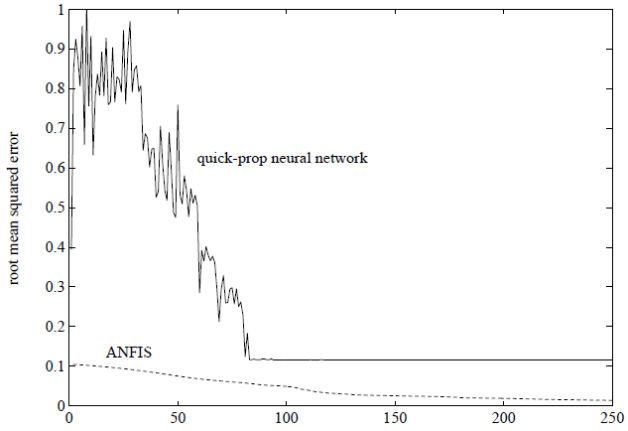


Figure 2.1: ANN pattern recognition vs ANFIS [8].

Figure 2.1 shows that ANFIS can effectively model a highly non-linear function compared to neural networks.

2.1.1 The network

The structure of a 4-rule ANFIS can be seen in figure 2.2, the parameterized neurons belong to layers 1 and 4, while neurons from layer 2, 3 and 5 are not parameterized.

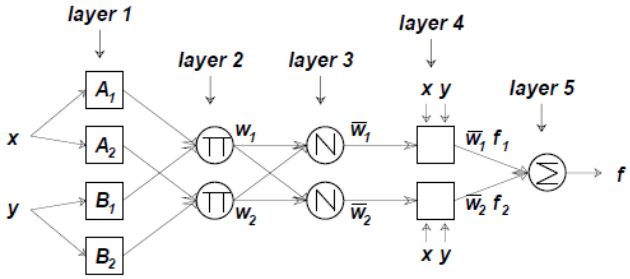


Figure 2.2: ANFIS structure for 2 inputs, 2MF's each, 1 output,and 2 rules [8].

Layer 1.- In this layer every input is fuzzyfied by Membresy Functions (MF's) like gaussian or bell functions,

$$Gauss(x; a, c) = exp\Big(- \Big(\frac{(x-c)}{a}\Big)^2\Big) \tag{2.1}$$

$$Bell(x; a, b, c) = \frac{1}{1 + (\frac{x-c}{a})^{2b}} \tag{2.2}$$

in node notation,

$$O_i^1 = \mu_{A_i}(x, a_i, c_i) \quad or \quad O_{i+2}^1 = \mu_{B_i}(y, a_i, c_i) \tag{2.3}$$

for every node in layer 1.

Training parameters are a, c (and b). Parameters from layer 1 are referred to as *premise parameters*.

Layer 2.- Every node in this layer is a circle node labeled Π which multiplies the incoming signals and sends the output as w_i.

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y) \quad for\ i = 1, 2 \tag{2.4}$$

where,

$$O_i^2 = w_i \tag{2.5}$$

Theses nodes represent the output of an AND operation and so called *rule* in fuzzy reasoning.

Layer 3.- Every node in this layer is a circle node labeled N. The i-th node represents the firing strength of the i-th rule over all others. Mathematically, it represents the normalization of rules firing strength,

$$\bar{w_i} = \frac{w_i}{\sum_{j=1}^{2} w_j} \quad for\ i = 1, 2 \tag{2.6}$$

or

$$O_i^3 = \bar{w_i} \tag{2.7}$$

Layer 4.- Every node in this layer is a parameterized node.

$$O_i^4 = \bar{w_i} f_i \tag{2.8}$$

where, f_i is a function of the inputs x, y . This function depends on the type of ANFIS used (figure 2.3). Parameters used in this layer are referred to as *consequent parameters*.

Layer 5.- The single node in this layer is a circle node labeled $\sum$ that computes the overall output as the summation of all incoming signals. It is the fuzzy equivalent of the boolean OR operation.

$$O_i^5 = f = \sum \bar{w_i} f_i \tag{2.9}$$

Figure 2.3 shows the 3 types of ANFIS reasoning. Type 1 uses nonlinear sigmoidal functions to obtain f_i, type 2 finds the geometrical center between the multiplication (or AND operation) of MFs, while type 3 uses a linear parameterization of the inputs.

Type 1 ANFIS is used in this work because sigmoidal functions are said to be more exact as they are always continuous and derivable [8]. On the other hand type 3 ANFIS has the advantage to be trained under the hybrid method (for its linearity), which is based in training the consequent parameters using the least square method [8].

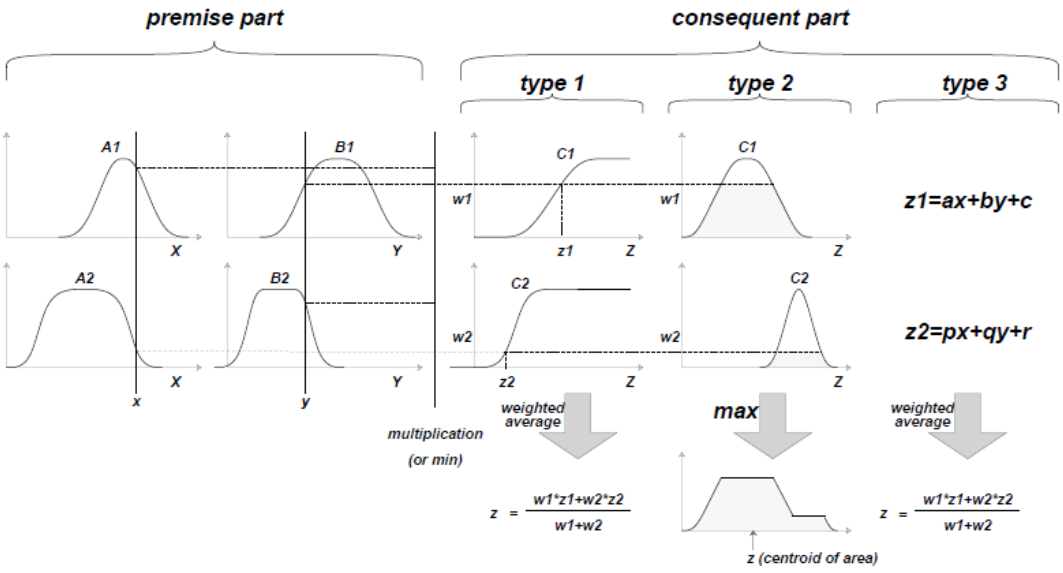


Figure 2.3: Types of ANFIS structures [8].

where,

$$Sigmoid(x; a, c) = \frac{1}{1 + exp(-\frac{x-c}{a})} \tag{2.10}$$

$$Sigmoid^{-1}(x; a, c) = c - a\ ln\Big(\frac{1}{x} - 1\Big) \tag{2.11}$$

2.1.2 Training

There are many ways to train an ANFIS network [10]. Here it will be discussed the method using the gradient of the error.

For a model to be obtained there's the need of data, represented in the form of vectors. Training sweeps through these vectors from the first to last data elements ($p \in [1, P]$) and compares what the p-th target should look like after applying some p-th input elements.

Considering this, the error for the p-th data elements in ANFIS is represented by

$$E_p = Target_p - O^5_{1,p} \tag{2.12}$$

The ultimate goal is to minimize this error by moving the premise and consequent parameters, adjusting the Gauss-Bell-Sigmoid functions to match the behavior of the process in a *reasoning* way (reasoning as in fuzzy logic operations).

Mathematically, this process can be accomplished with optimization,

$$J_p = (T_p - O^5_{1,p})^2 \tag{2.13}$$

where,

J_p = The cost function to minimize for the p-th data pairs.
T_p = The output target for the p-th data pairs.
$O^5_{1,p}$ = The actual ANFIS output for the p data pairs.

in this work the minimization process is carried out by the gradient method, so the final goal is to find the equation of

$$\frac{\partial J_p}{\partial \alpha} \tag{2.14}$$

where,

$\alpha =$ Some premise or consequent parameter.

to find out what is the direction of the gradient and where should the parameter move to.

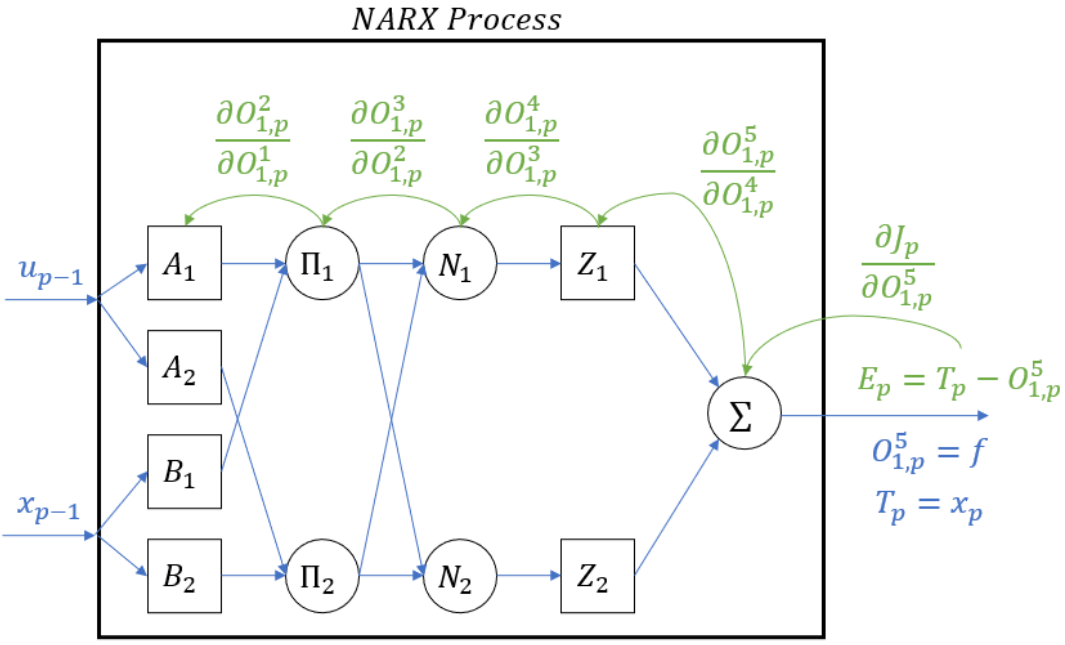


Figure 2.4: ANFIS training for a NARX process structure.

A. Consequent parameters training

The goal is to obtain a representation of $\frac{\partial J_p}{\partial \alpha}$ given by the chain rule. The ways to obtain gradients for the consequent parameters are expressed next.

Between the cost function and layer 5,

$$\begin{aligned}\frac{\partial J_p}{\partial O^5_{1,p}} &= \frac{\partial((T_p - O^5_{1,p})^2)}{\partial O^5_{1,p}} \\ &= -2E_p\end{aligned}$$

between layer 5 and f_i

$$\begin{aligned}\frac{\partial O^5_{1,p}}{\partial f_{i,p}} &= \frac{\partial(\sum \bar{w}_{k,p} f_{k,p})}{\partial f_{i,p}} \\ &= \bar{w}_{i,p}\end{aligned}$$

between f_i and consequent parameters,

$$\begin{aligned}\frac{\partial f_{i,p}}{\partial a_i} &= \frac{\partial (Sigmoid^{-1}(w_{i,p}, a_i, c_i))}{\partial a_i} \\ &= \frac{\partial (c - aln(\frac{1}{w_{i,p}} - 1))}{\partial a_i} \\ &= -ln\Big(\frac{1}{w_{i,p}} - 1\Big)\end{aligned}$$

$$\frac{\partial J_p}{\partial c_i} = 1;$$

finally,

$$\frac{\partial J_p}{\partial a_i} = \frac{\partial J_p}{\partial O^5_{i,p}} \frac{\partial O^5_{i,p}}{\partial f_{i,p}} \frac{\partial f_{i,p}}{\partial a_i} = (-2E_p)(\bar{w}_{i,p})\Big(- ln\Big(\frac{1}{w_{i,p}} - 1\Big)\Big) \tag{2.15}$$

$$\frac{\partial J_p}{\partial c_i} = \frac{\partial J_p}{\partial O^5_{i,p}} \frac{\partial O^5_{i,p}}{\partial f_{i,p}} \frac{\partial f_{i,p}}{\partial c_i} = (-2E_p)(\bar{w}_{i,p})(1) \tag{2.16}$$

to apply a change in each parameter after all data pairs have been presented,

$$\frac{\partial J}{\partial \alpha_i} = \sum_{p=1}^{P} \frac{\partial J_p}{\partial \alpha_i} \tag{2.17}$$

$$\Delta \alpha_i = -\eta \frac{\partial J_p}{\partial \alpha_i} \tag{2.18}$$

where η can be expressed in the form,

$$\eta = \frac{K}{\sqrt{\Big(\sum_{\alpha_i} \frac{\partial J}{\partial \alpha_i}\Big)^2}} \tag{2.19}$$

this way K sets the step size for the change in the parameter.

This process is then ran again and again. A training cycle for P data entries is called *epoch*. Epochs are ran N times. This way, after every epoch,

$$\alpha_{i,n} = \alpha_{i,n-1} - K \times sign\Big(\frac{\partial J}{\partial \alpha_i}\Big) \tag{2.20}$$

B. Premise parameters training

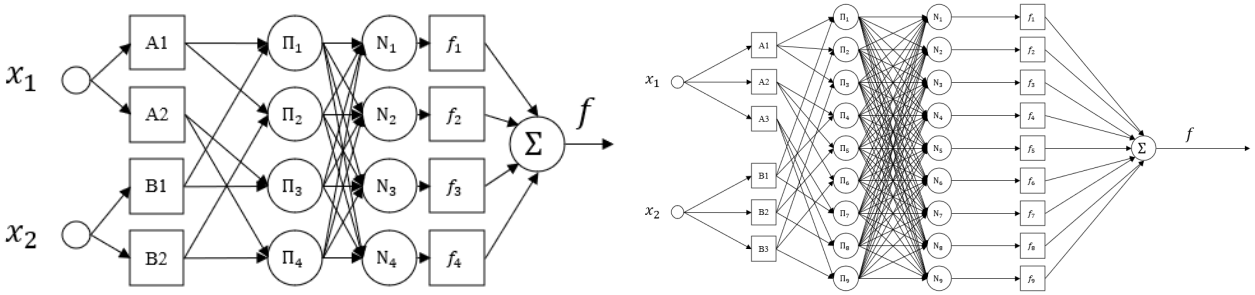
Between nodes from layer 5 and layer 2,

$$\begin{aligned}\frac{\partial O_{1,p}^5}{\partial O_{i,p}^2} &= \frac{\partial(\sum \bar{w}_{k,p} f_{k,p})}{\partial O_{i,p}^2} \\ &= \frac{\partial(\bar{w}_{1,p} f_{1,p} + \cdots + \bar{w}_{M,p} f_{M,p})}{\partial w_{i,p}} \\ &= \frac{\partial\left(\frac{(w_{1,p} f_{1,p} + \cdots + w_{M,p} f_{M,p})}{(w_{1,p} + \cdots + w_{M,p})}\right)}{\partial w_{i,p}} \\ &= \frac{f_{i,p} - O_{1,p}^5}{\sum(w_{k,p})}\end{aligned}$$

or in vector representation,

$$\frac{\partial O_p^5}{\partial O_p^2} = \begin{bmatrix} \frac{f_{1,p} - O_{1,p}^5}{\sum(w_{k,p})} & \cdots & \frac{f_{M,p} - O_{1,p}^5}{\sum(w_{k,p})} \end{bmatrix} \tag{2.21}$$

A general representation of $\frac{\partial O_{i,p}^2}{\partial O_{i,p}^1}$ holding all possible input combinations ($nRules = nFuzzy^{nInput}$) is formulated next. This part will be analyzed using 2 and 3 input ANFIS examples as shown in figure 2.5 and 2.7.



(a) 2 MF's = 4 Rules (b) 3 MF's = 9 Rules

Figure 2.5: Maximum size structures for 2 input ANFIS.

The combinations of layer 1 nodes to obtain layer 2 nodes outputs w_i are represented in table 2.1,

nFuzzy	nRules
2	4

μ_{A_i}	μ_{B_i}
1	1
1	2
2	1
2	2

nFuzzy	nRules
3	9

μ_{A_i}	μ_{B_i}
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

Table 2.1: ANFIS maximum combinations of layer 1 nodes for a 2 input ANFIS.

$\frac{\partial O_p^2}{\partial O_{i,p}^1}$ is a column vector with size *nRules*, then for every node from layer 1 we get $\frac{\partial O_p^2}{\partial O_p^1}$, a matrix containing all column vectors (figure 2.6).

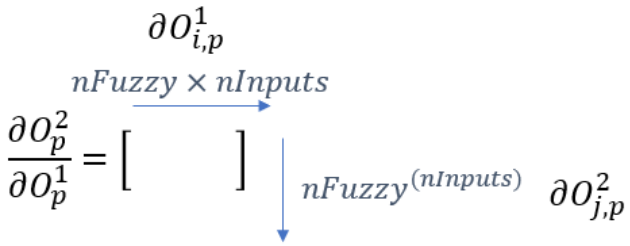


Figure 2.6: General view of matrix $\frac{\partial O_p^2}{\partial O_p^1}$.

Calculating the partial derivatives of figure 2.5 ANFISs,

$$\frac{\partial O_{i,p}^2}{\partial O_{i,p}^1} = \begin{bmatrix} O_{B_1,p}^1 & 0 & O_{A_1,p}^1 & 0 \\ O_{B_2,p}^1 & 0 & 0 & O_{A_1,p}^1 \\ 0 & O_{B_1,p}^1 & O_{A_2,p}^1 & 0 \\ 0 & O_{B_2,p}^1 & 0 & O_{A_2,p}^1 \end{bmatrix}$$

for the 3MF ANFIS,

$$\frac{\partial O_{i,p}^2}{\partial O_{i,p}^1} = \begin{bmatrix} O_{B_1,p}^1 & 0 & 0 & O_{A_1,p}^1 & 0 & 0 \\ O_{B_2,p}^1 & 0 & 0 & 0 & O_{A_1,p}^1 & 0 \\ O_{B_3,p}^1 & 0 & 0 & 0 & 0 & O_{A_1,p}^1 \\ 0 & O_{B_1,p}^1 & 0 & O_{A_2,p}^1 & 0 & 0 \\ 0 & O_{B_2,p}^1 & 0 & 0 & O_{A_2,p}^1 & 0 \\ 0 & O_{B_3,p}^1 & 0 & 0 & 0 & O_{A_2,p}^1 \\ 0 & 0 & O_{B_1,p}^1 & O_{A_3,p}^1 & 0 & 0 \\ 0 & 0 & O_{B_2,p}^1 & 0 & O_{A_3,p}^1 & 0 \\ 0 & 0 & O_{B_3,p}^1 & 0 & 0 & O_{A_3,p}^1 \end{bmatrix}$$

For a 3 input ANFIS as shown in figure 2.7,

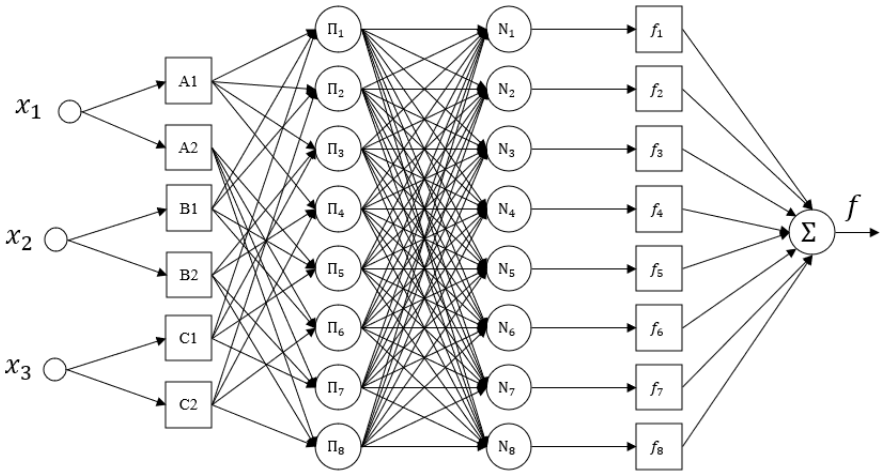


Figure 2.7: 3 input 2MFs ANFIS structure.

the combination of layer 1 nodes are represented in table 2.2,

nFuzzy		nRules
2		8
μ_{A_i}	μ_{B_i}	μ_{C_i}
1	1	1
1	1	2
1	2	1
1	2	2
2	1	1
2	1	2
2	2	1
2	2	2

Table 2.2: ANFIS maximum combinations of layer 1 nodes for 3 input ANFIS.

matrix $\frac{\partial O_p^2}{\partial O_p^1}$ results in,

$$\frac{\partial O_p^2}{\partial O_p^1} = \begin{bmatrix} O^1_{B_1,p}O^1_{C_1,p} & 0 & O^1_{A_1,p}O^1_{C_1,p} & 0 & O^1_{A_1,p}O^1_{B_1,p} & 0 \\ O^1_{B_1,p}O^1_{C_2,p} & 0 & O^1_{A_1,p}O^1_{C_2,p} & 0 & 0 & O^1_{A_1,p}O^1_{B_1,p} \\ O^1_{B_2,p}O^1_{C_1,p} & 0 & 0 & O^1_{A_1,p}O^1_{C_1,p} & O^1_{A_1,p}O^1_{B_2,p} & 0 \\ O^1_{B_2,p}O^1_{C_2,p} & 0 & 0 & O^1_{A_1,p}O^1_{C_2,p} & 0 & O^1_{A_1,p}O^1_{B_2,p} \\ 0 & O^1_{B_1,p}O^1_{C_1,p} & O^1_{A_2,p}O^1_{C_1,p} & 0 & O^1_{A_2,p}O^1_{B_1,p} & 0 \\ 0 & O^1_{B_1,p}O^1_{C_2,p} & O^1_{A_2,p}O^1_{C_2,p} & 0 & 0 & O^1_{A_1,p}O^1_{B_1,p} \\ 0 & O^1_{B_2,p}O^1_{C_1,p} & 0 & O^1_{A_3,p}O^1_{C_1,p} & O^1_{A_2,p}O^1_{B_2,p} & 0 \\ 0 & O^1_{B_2,p}O^1_{C_2,p} & 0 & O^1_{B_1,p}O^1_{C_2,p} & 0 & O^1_{A_1,p}O^1_{B_2,p} \end{bmatrix}$$

the conclusion is explicit: matrix $\frac{\partial O_p^2}{\partial O_p^1}$, is not always of the same length and its size changes exponentially when number of MFs or inputs is increased.

The way to generate this matrix (in code) is by looking at the combination table **rows** and "toggle" the node corresponding to the derivative. For example, looking at element (1,1) from table 2.2 we see μ_{A_1}. Then the element (1,1) from matrix $\frac{\partial O_p^2}{\partial O_p^1} = \frac{\partial w_1}{\partial \mu_{A_1}}$ is generated by toggling μ_{A_1} from row 1 of table 2.2 and keeping the multiplication between the other elements, resulting in $\mu_{B_1} \times \mu_{C_1}$ or $O^1_{B_1,p} O^1_{C_1,p}$ in node notation. This way the matrix can be generated automatically by using the combination tables.

To determine $\frac{\partial O^1_{i,p}}{\partial \alpha}$,

$$\begin{aligned}\frac{\partial O^1_{i,p}}{\partial a_i} &= \frac{\partial\Big(exp\Big(-\Big(\frac{(x_i-c_i)}{a_i}\Big)^2\Big)\Big)}{\partial a_i} \\ &= \frac{2exp(-\frac{(x-c)^2}{a^2})(x-c)^2}{a_i^3}\end{aligned}$$

$$\begin{aligned}\frac{\partial O^1_{i,p}}{\partial c_i} &= \frac{\partial\Big(exp\Big(-\Big(\frac{(x_i-c_i)}{a_i}\Big)^2\Big)\Big)}{\partial c_i} \\ &= \frac{2exp(-\frac{(x-c)^2}{a^2})(x-c)}{a_i^2}\end{aligned}$$

making use of the chain rule

$$\frac{\partial J_p}{\partial \alpha_i} = \frac{\partial J_p}{\partial O^5_{1,p}} \frac{\partial O^5_{1,p}}{\partial O^2_p} \frac{\partial O^2_p}{\partial O^1_{i,p}} \frac{\partial O^1_{i,p}}{\partial \alpha_i}$$

where,

$\frac{\partial O^5_{1,p}}{\partial O^2_p}$ = The partial derivative **row vector** between $O^5_{1,p}$ and layer 2 nodes.
$\frac{\partial O^2_p}{\partial O^1_{i,p}}$ = The i-th **column** of matrix $\frac{\partial O^2_p}{\partial O^1_p}$.

After all of the data has been processed in an *epoch* and $\frac{\partial J}{\partial \alpha_i}$ has been obtained, the change in a parameter is done by the same calculation as consequent parameters.

2.1.3 Iterative process

P denotes the length of the data vectors and N the number of epochs, which is a design parameter. The rule of hand is clear, more epochs means more refined training results, but while more epochs, the training process can take too much time to compile and it also may loose validation. The iterative process can be seen in figure 2.8.

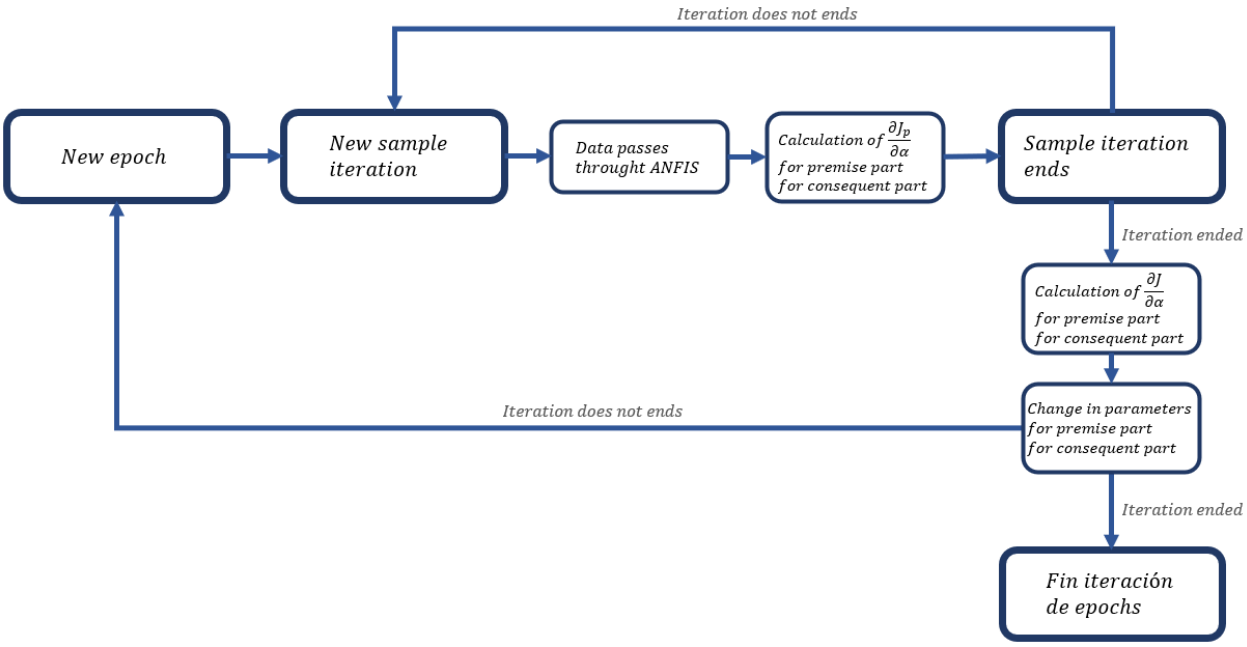


Figure 2.8: ANFIS training iterative algorithm.

2.1.4 Practical considerations

There are many practical considerations for training an ANFIS network, they are discussed in this section.

1. **Average Percentage Error (APE).** The APE is a measure of training error for each epoch, it is calculated by

$$APE = \frac{1}{P}\sum_{p=1}^{P}\frac{|T_p - O_{1,p}^5|}{|T_p|}100\% \tag{2.22}$$

   it is essential to calculate this parameter in order to measure how well the network is trained.

2. **Changing K.** K is the step size for the change in a parameter. More times than we would expect, K seems to be too small for the APE to go smaller, or too big to finally not obtain the minimum APE. A way to exploit K is to update it when the APE is not moving the way we would want to.

   2 rules are used to update this parameter,

   1.- If the error measure (APE) undergoes 4 consecutive reductions, increase K by 10%.

   2.- If the error measure undergoes 2 consecutive combinations of 1 increase and 1 reduction, decrease K by 10%.

3. **The ANFIS needs a reference.** This observation applies for two concepts.

i. On one hand, this finding is related with the type of ANFIS trained. There is a lot of theoretical information in the literature about ANFIS being used as an input/output model (see fig. 2.9) specially for control strategies involving neural networks like inverse learning control or internal model control. These I/O models cannot achieve the grade of identification as NARX models because when a NN is trained as an I/O model there will always be missing a reference point.

We can think of this concept as follows: a neural network is a powerful tool that can recognize patterns and **static** relationships between variables. Dynamic processes are systems that move with time and differential equations are the expressions that describe the behavior of them. When a differential equation is solved, an integration is produced, and when an integration is carried out, reference points are needed (commonly +C for continuous systems or y[k-N] for discrete systems). This implies that I/O models are usually not a suitable choice for process modelling because they will only extrapolate the outputs from what they have been taught from inputs. For example, if in the training data there is a considerable sudden move of the input (and the output doesn't move immediately), the network will recognize that 2 different inputs produce the same output which is wrong to assume.

(a) Input output model (b) NARX model

Figure 2.9: Common neural network models for dynamic processes.

ii. On the other hand and knowing that a NARX model is preferred, an important observation was made, and is that the ANFIS trained model cannot be sustained by itself. As the model is an approximation of the process and numerical errors are produced, the model cannot stay away from the real process as a standalone unit, it needs the real process output as an input (y[k-N]) for closer and better approximations of the real output ($\hat{y}$[k] $\approx$ y[k]).

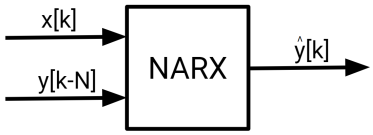


Figure 2.10: Correct use of the NARX ANFIS model.

4. **Scaling to fuzzy space.** There is one complication when training with raw data. Lets say we want to train a network using a data vector holding values from -12V to

12V. The APE is calculated as above, it has T_p as denominator, and so if this is 0 (or near 0) there will be errors and the APE will not evolve normally.

To solve this, what it is proposed in this work, is to set new limits for the data values and scale them away from 0. In this work, all training data values are scaled to 10%-90%, this way fuzzy MFs are kept within 0% to 100% and values will always be inside their range, avoiding T_i near 0.

2.2 Dynamic Matrix Control (DMC)

Predictive controllers are a suitable choice for using a NARX ANFIS in a control scheme. Other control strategies found in the literature using an ANFIS showed weak control in practical uses because their working principle is based on Input/Output models, which are not implementable in practice.

Many predictive controllers are based on using **polynomial models** to calculate the optimum control action for the controller. DMC has the advantage (with ANFIS) that it uses the step response to model the process, this step response can be generated not by a set of equations, but by using an ANFIS to simulate the behavior of the plant and future outputs.

DMC has its limitations and they are that the process must be stable and without integrators [20]. Once a step model is identified, it has the benefit that can allow complex dynamics such as nonminimum phase or delays to be described easily.

2.2.1 Formulation of the controller

A step response model is employed,

$$y(t) = \sum_{i=1}^{\infty} g_i \Delta u(t-i)$$

where,

$g_i = i$-th step coefficient as shown in figure 2.11.

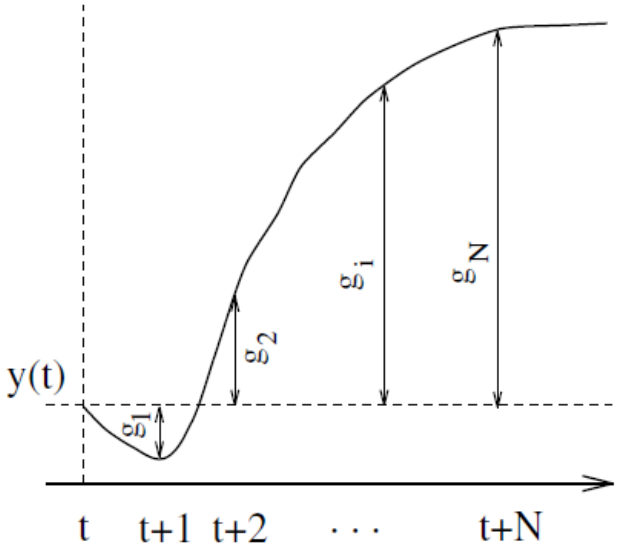


Figure 2.11: Step response example of a SISO process showing g_i coefficients [20].

The predicted values along the horizon will be,

$$\hat{y}(t+k|t) = \sum_{i=1}^{\infty} g_i \Delta u(t+k-i) + \hat{n}(t+k|t)$$

where,

$\hat{y}(t+k|t)$ = Predicted value of $y(t+k)$ at time t.
$\hat{n}(t+k|t)$ = Predicted value of the disturbance.

disturbances are considered to be constant $\hat{n}(t+k|t) = \hat{n}(t|t) = y_m(t) - \hat{y}(t|t)$, where $y_m(t)$ is the measurable output. It can be written that

$$\begin{aligned}\hat{y}(t+k|t) &= \sum_{i=1}^{k} g_i \Delta u(t+k-i) + \sum_{i=k+1}^{\infty} g_i \Delta u(t+k-i) + y_m(t) - \sum_{i=1}^{\infty} g_i \Delta u(t-i) \\ &= \sum_{i=1}^{k} g_i \Delta u(t+k-i) + y_{free}(t+k)\end{aligned}$$

where,

$y_{free}(t+k) = y_m(t) + \sum_{i=1}^{\infty}(g_{k+i} - g_i)\Delta u(t-i)$; the free response of the system.

The *free response* represents the response of the system if the control actions are kept constant through the prediction horizon. The *forced response* is the response of the output given changes in the control action (see figure 2.12).

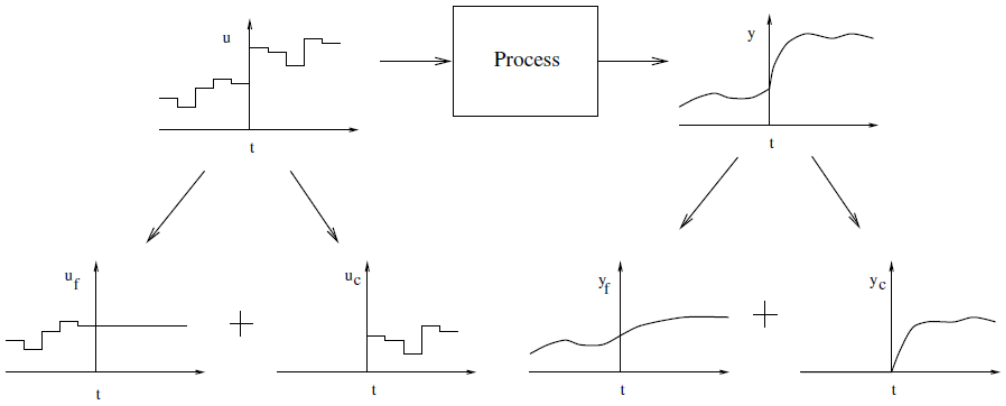


Figure 2.12: Free (f) and forced (c) responses [20].

If the process is **asymptotically stable**, the coefficients g_i of the step response tend to a constant value after N sampling periods, $g_{k+i} - g_i \approx 0$, $i > N$, so the free response can be calculated as,

$$y_{free}(t+k) = y_m(t) + \sum_{i=1}^{N}(g_{k+i} - g_i)\Delta u(t-i) \tag{2.23}$$

the predictions can be computed along the prediction horizon N_p, considering N_c control actions,

$$
\begin{aligned}
\hat{y}(t+1|t) &= g_1\Delta u(t) + y_{free}(t+1) \\
\hat{y}(t+2|t) &= g_2\Delta u(t) + g_1\Delta u(t+1) + y_{free}(t+2) \\
&\vdots \\
\hat{y}(t+N_p|t) &= \sum_{i=N_p-N_c+1}^{N_p} g_i\Delta u(t+N_p-i) + y_{free}(t+N_p)
\end{aligned}
$$

where we can define the *dynamic matrix* G,

$$
G = \begin{bmatrix}
g_1 & 0 & \dots & 0 \\
g_2 & g_1 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
g_{N_c} & g_{N_c-1} & \cdots & g_1 \\
\vdots & \vdots & \ddots & \vdots \\
g_{N_p} & g_{N_p-1} & \cdots & g_{N_p-N_c+1}
\end{bmatrix} \tag{2.24}
$$

and write a matrix equation,

$$
\hat{y} = Gu + y_{free} \tag{2.25}
$$

where,

$\hat{y}$ = A N_p size vector containing the system predictions along the horizon.
u = A N_c size vector containing the control increments.
y_{free} = Free response vector.

The goal is to drive the output as close to the setpoint as possible with the possibility of the inclusion of a penalty term on the input moves. The cost function that includes these penalties is expressed next,

$$
J = \sum_{j=1}^{N_p}[w(t+j) - \hat{y}(t+j|t)]^2|_Q + \sum_{j=1}^{N_c}[\Delta u(t+j-1)]^2|_R \tag{2.26}
$$

or,

$$
J = (w-\hat{y})^T Q(w-\hat{y}) + \Delta u^T R\Delta u \tag{2.27}
$$

where w is the reference to follow. Now, we want to find the optimal future control actions Δu. For that, we replace $\hat{y}$, compute the derivative of J with respect to Δu, and make it equal to 0, obtaining,

$$
\Delta u = (G^T QG + R)^{-1} G^T Q(w - y_{free}) \tag{2.28}
$$

As this is a predictive control strategy, only the first element of vector Δu is sent to the plant and the rest is discarded.

The difference with a **MIMO** process is that matrix G is of greater size. For example in a 2 input 2 output process we have,

$$G = \begin{bmatrix} G_{u_1y_1} & G_{u_2y_1} \\ G_{u_1y_2} & G_{u_2y_2} \end{bmatrix} \tag{2.29}$$

where,

$G_{u_jy_k}$ = Dynamic matrix holding the g_i coefficients between the j-th input and k-th output.

2.2.2 DMC with ANFIS

The general control strategy proposed in this work is presented in figure 2.13.

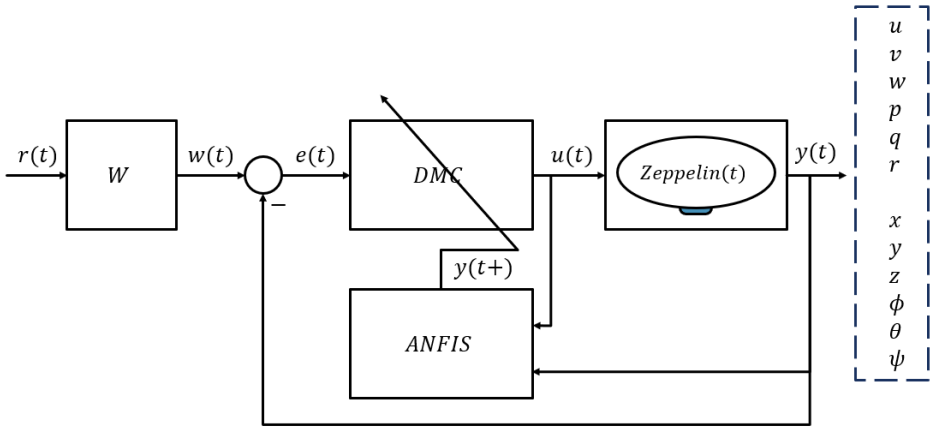


Figure 2.13: Control strategy using DMC + ANFIS

the way DMC is proposed to work with ANFIS is presented next, considering a NARX ANFIS with 4 inputs $u_1(k-1)$, $u_2(k-1)$, $u_3(k-1)$, $x_1(k-1)$ with output $x_1(k)$.

1. **Sample the output of the plant $X_1(k)$.**

2. **Scale to fuzzy values.** Using a mapping function with already defined limits (ex. [-12V,12V] $\rightarrow$ [10%,90%]),

$$\begin{aligned} x_1(k) &= Mapping(X_1(k); minIn, maxIn, minOut, maxOut) \\ &= \frac{X_1(k) - minIn}{maxIn - minIn}(maxOut - minOut) + minOut \end{aligned}$$

3. **Generate reference vector w.** Of length N_p with all values equal to the present reference ($w(k+N_p) = w(k)$). See section 2.2.3.

4. **Calculate y_{free}.** With a trained ANFIS, future output values are calculated as shown in figure 2.14, where the control actions are kept constant through the prediction horizon N_p.

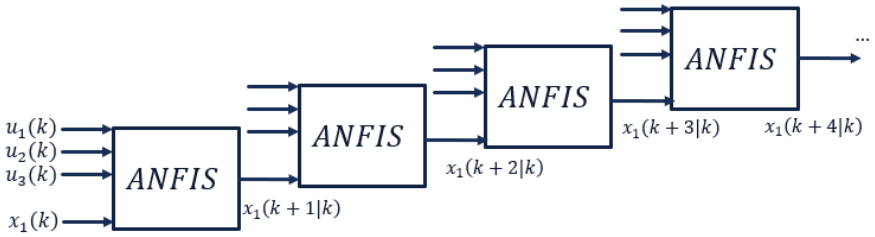


Figure 2.14: Method to obtain y_{free} using a NARX model ANFIS.

5. **Calculate y_{step}.** Very similar to y_{free}, simulate the step response, taking into account that du doesn't saturate the corresponding control action signal.

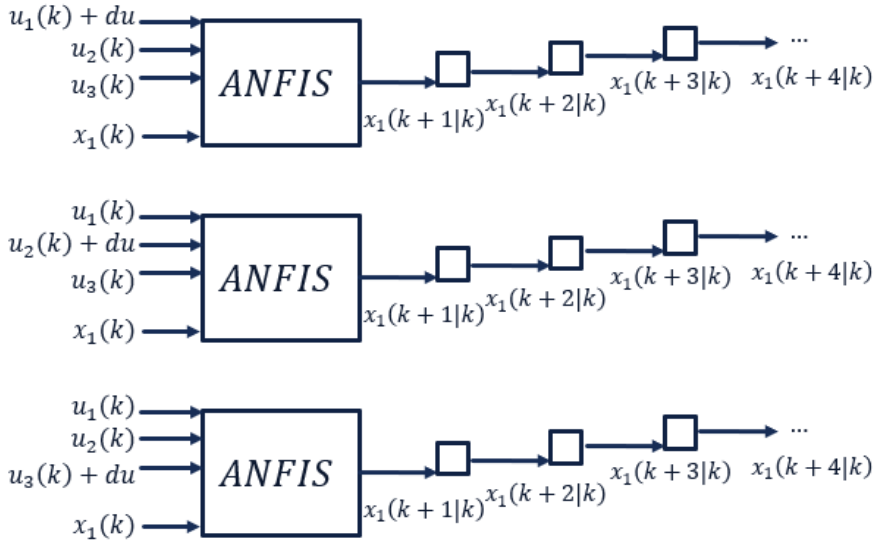


Figure 2.15: Method to obtain y_{step} using a NARX model ANFIS.

6. **Generate the dynamic matrix G.** Having y_{free} and y_{step}, calculate the g_i coefficients and assemble the dynamic matrix G.

$$g_i = \frac{y_{step} - y_{free}}{du} \tag{2.30}$$

Note that du must be the same step applied to the given input, it is not necessary to be the same for all inputs. For the example above, matrix G should be,

$$G = \begin{bmatrix} G_{u_1,y_1} & G_{u_2,y_1} & G_{u_3,y_1} \end{bmatrix}$$

7. **Solve the optimal control sequence.** Calculate

$$\Delta u = (G^T Q G + R)^{-1} G^T Q (w - y_{free})$$

8. **Take the first step.** Only the first element of Δu is taken into account. For the example above, elements 1, $1 + Nc$, and $1 + 2N_c$ are the steps for each control action. In a general way we know that,

$$u_i(k+1) = u_i(k) + \Delta u \tag{2.31}$$

9. **Scale to real values.** The control actions $u_i(k+1)$ must be scaled to the real values $U_i(k+1)$ using the mapping function.

2.2.3 Practical considerations

There are a few considerations to take into account. They are listed here,

1. **Filtered reference.** In order to make smoother transitions that can keep track of the reference without saturating the control actions, it is proposed to filter the reference before using it in the control scheme. The filtered reference w is shown in the next equation and in figure 2.16.

$$w_k = \alpha w_{k-1} + (1 - \alpha) r_k \tag{2.32}$$

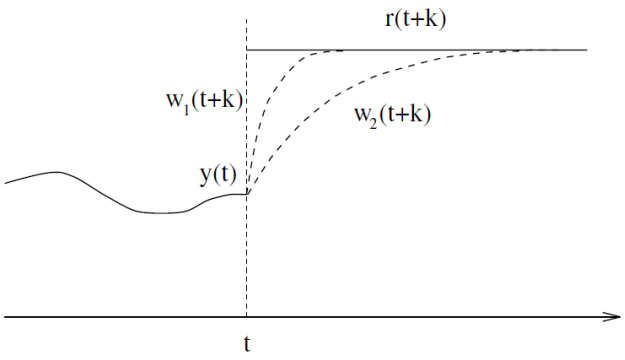


Figure 2.16: Smoother approximation of real reference r [20].

2. **Sample time T, prediction horizon N_p and control horizon N_c.** Some tips for configuring these parameters are listed here,

   - It is first recommended to choose the sample time in accordance to the process to be controlled. The election will depend on how fast the dynamics evolve. Automotive and aerospace systems are usually rapid and they require sample times of $T < 1s$, this is because systems with smaller sample time respond better to perturbations as they are "more prepared" to counter them.
   - On the other hand if T gets too small for a predictive control strategy, the prediction horizon may not see much of the behavior of the plant and control actions will be limited to a small horizon.
   - It is recommendable to set T, N_p, N_c according to the behavior of the system, and then tune the Q and R parameters for the calculation of control actions.

- Choosing a very small N_c over N_p offers less computing and as any predictive controller, decreases the errors accumulated from the calculation of horizons.

3. **Proportional gains.** If the controller is not able to keep track of the control actions by itself it may need proportional gains. A method to improve the behavior of the controller is given next,

$$u_i(k+1) = u_i(k) + Pu_i \times \Delta u \tag{2.33}$$

where,

$Pu_i =$ Proportional gain for the i-th control action.

Note that if $Pu_i = 1$ there are no changes in the controller.

2.3 Computer architecture & parallel programming

One of the intentions of this project was to design an accelerated heterogeneous architecture (FPGA) for ANFIS computation and training with the purpose of improving computation speeds and to extend the limitations of the proposed control strategy. Once the complexity of this proposal was studied and analyzed, the favorite choice was to design the ANFIS network in a GPU architecture, given the high parallel computing capacity and **already defined double precision arithmetic** which fits excellent with the operating base of ANFIS.

In the next subsections, NVIDIA's GPU CUDA architecture and parallel programming basics are presented [22].

2.3.1 Programming model

GPUs are hardware units specialized for highly parallel computations. It is this capacity that can be exploited for neural networks as they are made up of a series of layers containing parallel nodes executing arithmetic operations between them. When compared to a CPU, GPUs devote more transistors to data processing rather than executing sequential operations (fig. 2.17).

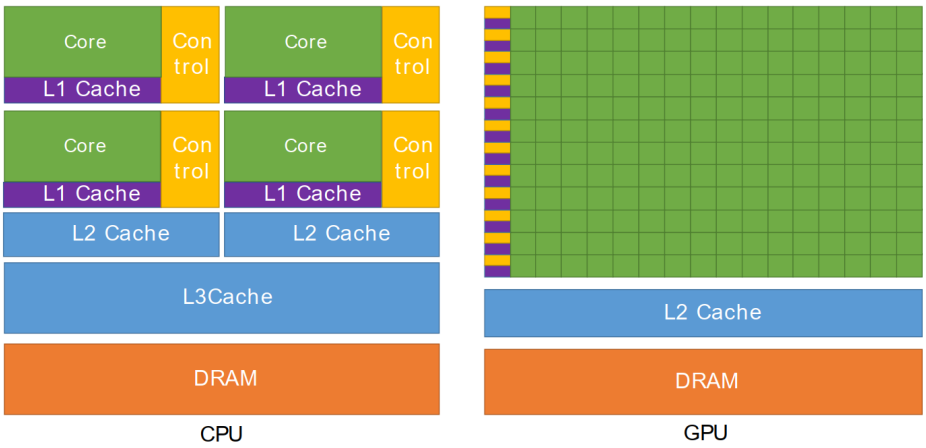


Figure 2.17: General resource comparison between CPU and GPU [22].

A few conceptual models are constructed in order to understand GPU programming, they are designed to "guide the programmer to partition the problem into coarse sub-problems that can be solved cooperatively in parallel by all threads within a CUDA block".

A. Kernels

Kernels are CUDA C++ functions that are executed N times in parallel in N different *CUDA threads*, as opposed to only once regular C++ functions.

Each thread that executes the kernel is given a unique *thread ID* that is accessible within the kernel through built-in variables, this way for example, two N sized vectors can perform any arithmetic point operation in parallel like as shown in figure 2.18.

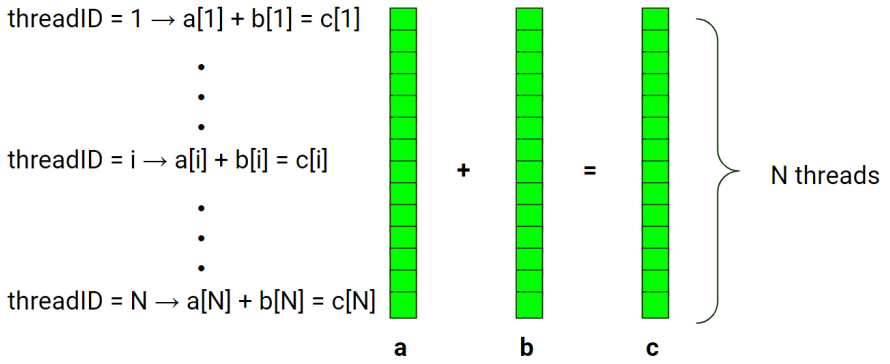


Figure 2.18: Addition example of two N sized arrays distributed along N *CUDA threads* [22].

B. Thread hierarchy

For convenience, threadID is a 3-component vector (x,y,z) so that threads can be identified using a one-dimensional, two-dimensional, or three-dimensional *thread index*. This way operations can be executed even in a cubical way.

Threads are the smaller computation units in a GPU, they are organized in groups of *blocks* and *grids* as shown in figure 2.19.

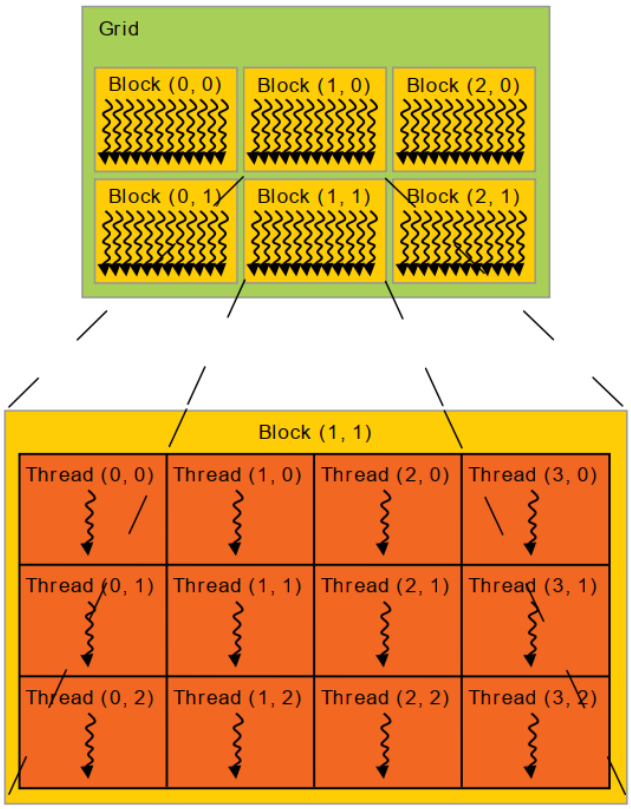


Figure 2.19: Thread hierarchy within a CUDA GPU [22].

The number of threads per block is fixed and usually around 1024 threads per block nowadays. The dimension of thread blocks is accessible through another built-in parameters

blockDim and *blockID*, this way if more than one block is to be used (more than 1024 parallel operations), the thread index for a one-dimensional array can be identified as next,

$$index = blockIdx.x \times blockDim.x + threadIdx.x \tag{2.34}$$

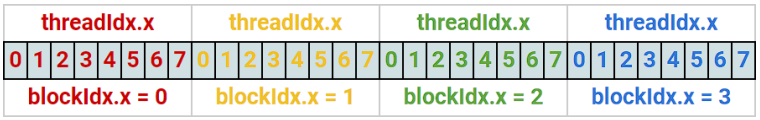


Figure 2.20: Example of thread hierarchy for 8 threads per block [22].

C. Memory hierarchy

When a kernel is executed in each thread from each block, computation occurs for defined data arrays or matrices. These one, two or three dimensional arrays do not come directly from the host (CPU) but need to be copied to the device (GPU) first. This way, a memory model is constructed.

The different memory rankings are listed next,

- **Local memory:** Only accessible from threads, relatively small. R/W.
- **Shared memory:** Only accessible from threads within a block, i.e. memory shared to 1024 threads in each block and not between blocks. R/W.
- **Constant memory:** Accessible from any thread in every grid. Read only memory.
- **Global memory:** Accessible from any thread in every grid. Of relatively big size. R/W.

It should be known that there is a trade-off in memory types between access speeds and size: memory closer to threads is smaller but faster, while global memory is bigger but relatively slower.

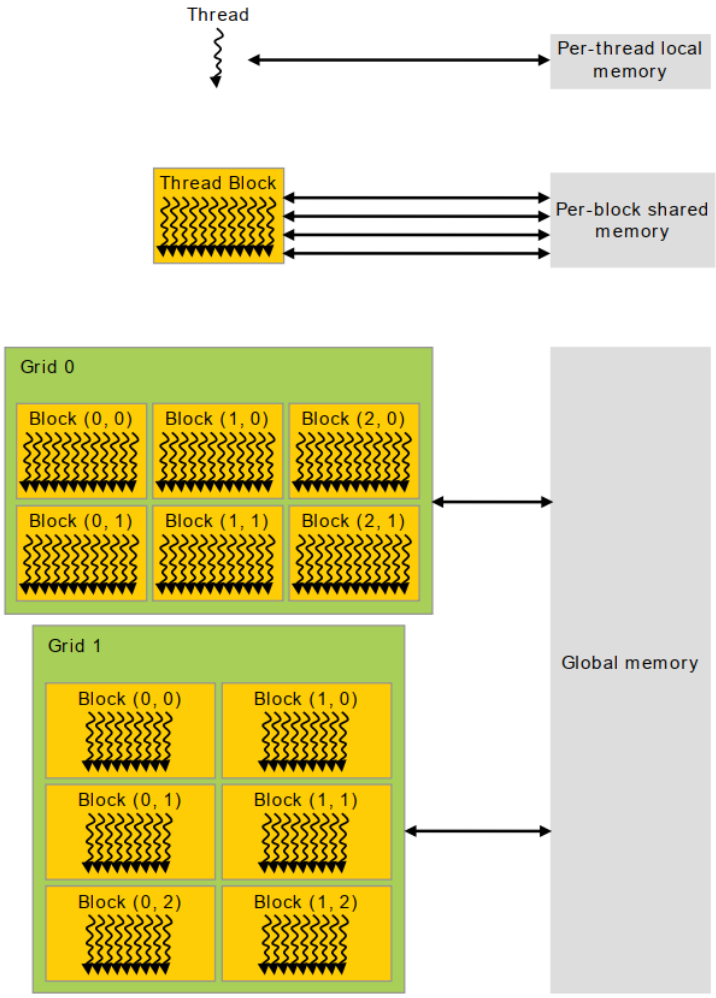


Figure 2.21: Memory hierarchy model for a CUDA GPU [22].

For practical contextualization purposes, specifications of the NVIDIA GeForce GTX 1650 GPU (the one used in this work) are shown in section A.

2.3.2 Heterogeneous computing

In summary, CUDA kernels are C++ based functions that are executed in each thread for each block, they usually perform small operations for a set of large sized arrays. Threads are grouped into blocks and blocks into grids. Arrays (memory) have to come from a place in the device (GPU), and so the memory hierarchy is presented.

Arrays involved in the kernel execution are first loaded into the *device* (GPU) from the *host* (CPU), and so a heterogeneous computing algorithm is modeled. The host sends information to the device, the device computes highly parallel operations and sends an output back to the host and keeps on executing serial instructions. Figure 2.22 shows an example model of a CUDA C++ program.

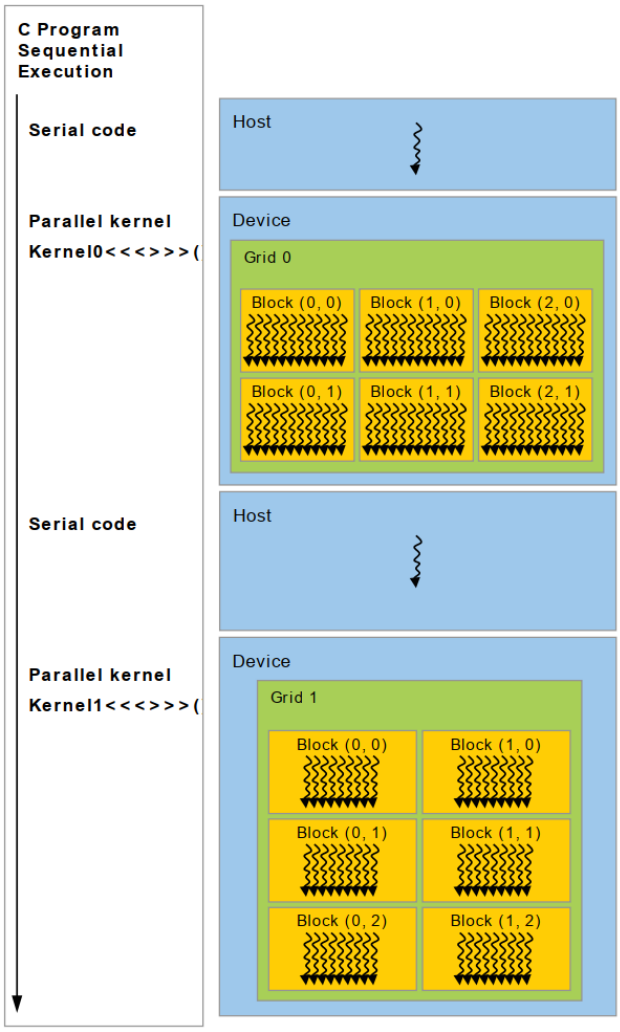


Figure 2.22: Example of heterogeneous programming [22].

2.3.3 ANFIS parallel model and control

Now that the base for parallel programming is presented, the ANFIS parallel structure is discussed.

The ANFIS kernel script is presented in the next chapters. This structure was designed to include all terms involved in the operation of ANFIS. As this is a CUDA C++ function, the kernel is defined as next,

```
__global__ void ANFIS(double* aIn, double* cIn, double* aOut, double* cOut, double* O0, double*
O1, double* O2, double* O3, double* O4, double* O5)
```

where,

aIn = double precision matrix of size `aIn[nFuzzy][nInputs]`
cIn = double precision matrix of size `cIn[nFuzzy][nInputs]`
$aOut$ = double precision array of size `aIn[nRules]`
$cOut$ = double precision array of size `cOut[nRules]`
$O0$ = double precision array of size. `O0[nInputs]`. ANFIS kernel primary inputs.
$O1$ = double precision array of size `O1[nInputs`$\times$`nFuzzy]`
$O2$ = double precision array of size `O2[nRules]`
$O3$ = double precision array of size `O3[nRules]`
$O4$ = double precision array of size `O4[nRules]`
$O5$ = double precision array of size `O5[1]`. ANFIS kernel primary output.

This way the designed kernel is to imitate an ANFIS like in figure 2.7, where every node or neuron is represented by one double precision data element and layer operations are executed in parallel.

The way the parallel CUDA C++ program works is shown in figure 2.23.

1. First, the **trained** ANFIS parameters are copied into the device (GPU) global memory.
2. When the CUDA ANFIS network is to be used, the kernel is launched inside a wrapping function so that the inputs are first copied into the device (GPU) global memory.
3. Once the device (GPU) inputs are copied and the kernel is launched, the output is extracted from the device into the host (CPU).

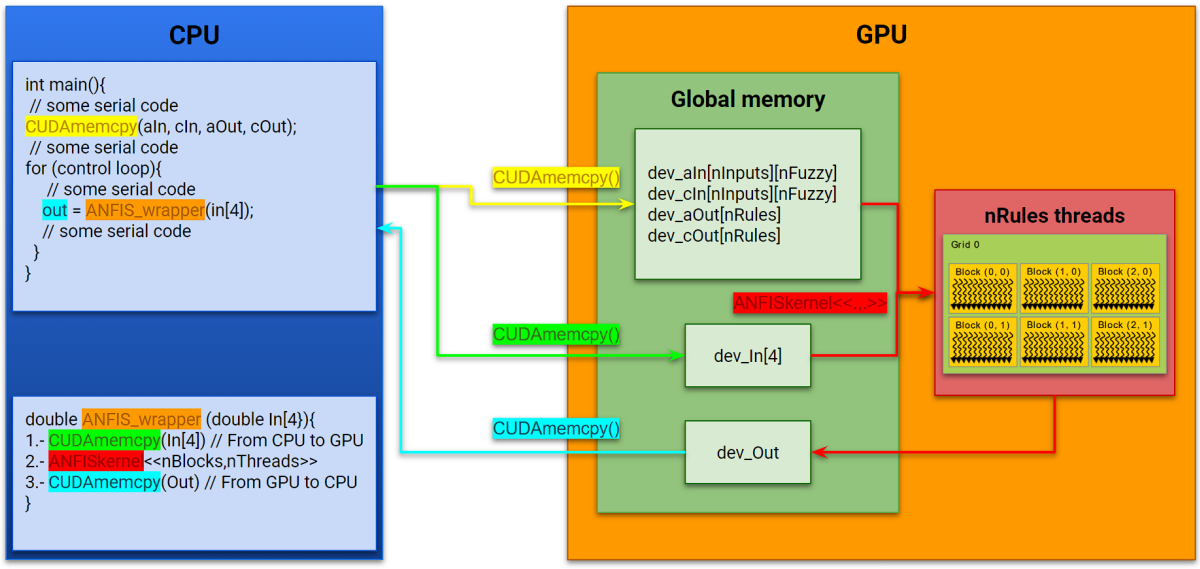


Figure 2.23: Example of a 4 input ANFIS parallel program.

where `CUDAmemcpy()` and `ANFISkernel«.,.»` are CUDA C++ APIs for easy host and device handling.

2.4 Dynamic model of a blimp

One of the objectives of this project is to make use of this control scheme using ANFIS + DMC over a complex process, with the purpose of creating advanced solutions to complex problems. For this, the proposal is to control a non-linear MIMO process like the given above: the 6DOF dynamic model of a blimp. This section presents the formulation of the blimp dynamics.

2.4.1 General aspects

Different to other aerial systems that use high speed winds to create *up* forces, blimps are based on the principle of *flotation*.

In big airships (thousands of kilos), the elevation is produced by the suction/expulsion of air from ballonets. When air is absorbed, the airship adds weight and also condenses the Helium (or non dense gas) inside of it, which makes the body more dense and thus to float less. On the contrary, expelling air produces the airship to go up (fig. 2.24).

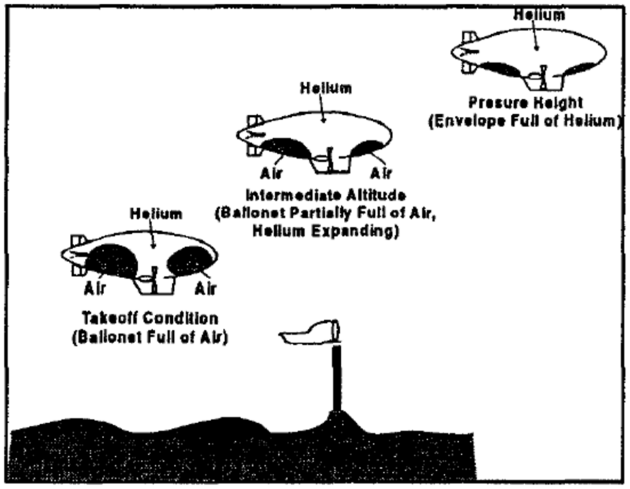


Figure 2.24: Ballonets action over the altitude in an airship [14]

The density of air, ρ_{air} changes with height and climate conditions. This is an important issue in airships, because it affects directly the control schemes for elevation.

It is mentioned in by Gomes [14] that airships are non minimum phase systems.

This work models a small scale blimp so that the ballonet system and air density changes will not be considered. Instead, for elevation, the *weight* of the blimp will match the *buoyancy* so that there are no forces acting in the *up* direction. The physical model can be observed in figure 2.25.

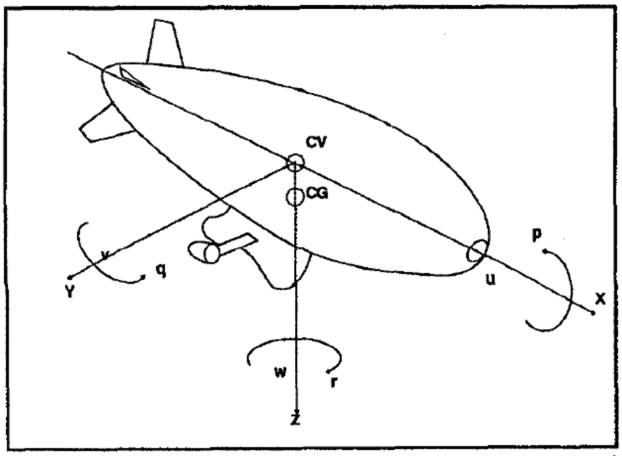


Figure 2.25: Blimp physical model on Cartesian coordinate system [14]

The mathematical model that describe the dynamics of the blimp, [4], [14],

$$M\dot{X}(t) = F_d(X(t)) + A(X(t)) + G(Y(t); \lambda_{31}, \lambda_{32}, \lambda_{33}) + P(U(t)) \tag{2.35}$$

where,

X = State vector containing 6 velocities u, v, w, p, q, r.
Y = State vector containing 6 positions x, y, z, ϕ, θ, ψ.
U = Input vector containing 3 inputs T_{ds}, T_{dp}, μ.
M = Masses and inertia matrix.
F_d = Dynamic forces vector containing the Coriolis and centrifugal forces.
A = Aerodynamic forces vector.
G = Gravitational forces vector.
λ_{ij} = Direction Cosine Matrix (DCM) elements.
P = Propulsion forces vector.

2.4.2 Masses and inertia matrix, M

Matrix M contains all of the real/virtual masses and inertias of the blimp associated to every direction. It is obtained by the analysis of the dynamics of an ellipsoid.

- The "real" part refers to the quantity of effective mass, how much the ellipsoid masses.
- The "virtual" part is produced by a physical phenomena. When an object moves in an infinite fluid, the kinetic energy of the fluid produces a small raise in the masses and inertia of the object due to the movement of the fluid masses when the object is moving. It is normally dismissed in most systems, but when dealing with low weight bodies, it is considerable [1], [2].

Matrix M is defined,

$$M = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 & 0 & ma_z - X_{\dot{q}} & 0 \\ 0 & m - Y_{\dot{v}} & 0 & -ma_z - Y_{\dot{p}} & 0 & ma_x - Y_{\dot{r}} \\ 0 & 0 & m - Z_{\dot{w}} & 0 & -ma_x - Z_{\dot{q}} & 0 \\ 0 & -ma_z - L_{\dot{v}} & 0 & I_x - L_{\dot{p}} & 0 & -J_{xz} \\ ma_z - M_{\dot{u}} & 0 & -ma_x - M_{\dot{w}} & 0 & I_y - M_{\dot{q}} & 0 \\ 0 & ma_x - N_{\dot{v}} & 0 & -J_{xz} & 0 & I_z - N_{\dot{r}} \end{bmatrix} \tag{2.36}$$

where,

m = Total real mass of the blimp.
I_x, I_y, I_z = Real inertia along the x, y, z axes.
$X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}$ = Virtual masses along the x, y, z axes.
$L_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}$ = Virtual inertias along the x, y, z axes.
a_x, a_z = x, and z distances between the Center of gravity (CG) and Center of volume (CV).

The rest of the virtual masses and inertias can be dismissed [4], [14].

A. Obtaining the masses and inertias.

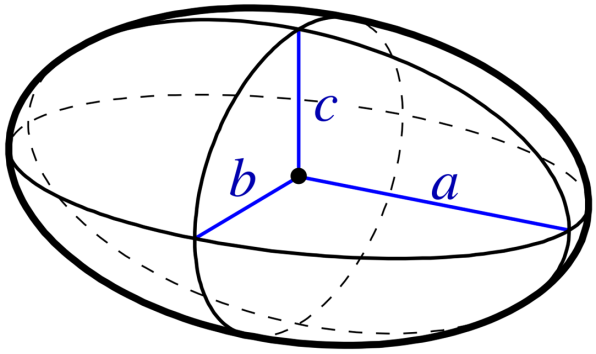


Figure 2.26: Ellipsoid dimensions with a,b,c pointing towards the x, y, z axes respectively.

The inertias of an ellipsoid like the one in figure 2.26 are given by,

$$I_x = m\frac{b^2 + c^2}{5}; \;\; I_y = m\frac{c^2 + a^2}{5}; \;\; I_z = m\frac{a^2 + b^2}{5}; \tag{2.37}$$

The virtual masses and inertias can be obtained in 2 ways, from tables (H. Lamb 1918) [1] or in an analytical form (L.B. Tuckerman 1926) [2]. This section will discuss the analytical way for a “Prolate Spheroid" (a>b=c). The “k” coefficients are found by the shape of the body, then multiplied by the density of the fluid as both take part of this phenomena.

Defining primary parameters from ellipsoid dimensions,

$$e = \sqrt{1 - \frac{c^2}{a^2}} \tag{2.38}$$

$$\alpha = \frac{1-e^2}{e^3}\Big(ln\Big(\frac{1+e}{1-e}\Big) - 2e\Big) \tag{2.39}$$

$$\beta = \gamma = \frac{1-e^2}{e^3}\Big(\frac{e}{1-e^2} - \frac{1}{2}ln\Big(\frac{1+e}{1-e}\Big)\Big) \tag{2.40}$$

Obtaining "k" coefficients,

$$k_1 = Vol\frac{\alpha}{2-\alpha} \qquad k_1' = (\frac{b^2-c^2}{b^2+c^2})^2\frac{\gamma-\beta}{2\frac{b^2-c^2}{b^2+c^2} - (\gamma-\beta)} \tag{2.41}$$

$$k_2 = Vol\frac{\beta}{2-\beta} \qquad k_2' = (\frac{c^2-a^2}{c^2+a^2})^2\frac{\alpha-\gamma}{2\frac{c^2-a^2}{c^2+a^2} - (\alpha-\gamma)} \tag{2.42}$$

$$k_3 = Vol\frac{\gamma}{2-\gamma} \qquad k_3' = (\frac{a^2-b^2}{a^2+b^2})^2\frac{\beta-\alpha}{2\frac{a^2-b^2}{a^2+b^2} - (\beta-\alpha)} \tag{2.43}$$

Finally, the virtual masses,

$$X_{\dot{u}} = -k_1\rho_{air} \qquad L_{\dot{p}} = -k_2'\rho_{air} = 0 \tag{2.44}$$

$$Y_{\dot{v}} = -k_2\rho_{air} \qquad M_{\dot{q}} = -k_2'\rho_{air} \tag{2.45}$$

$$Z_{\dot{w}} = -k_3\rho_{air} \qquad N_{\dot{r}} = -k_3\rho_{air} \tag{2.46}$$

2.4.3 Dynamic forces vector, Fd

This vector contains the dynamic forces of the ellipsoid produced by the Coriolis and centrifugal forces,

$$F_d = \begin{bmatrix} -m_z wq + m_y rv + m(a_x(q^2+r^2) - a_z rp) \\ -m_x ur + m_z pw + m(-a_x pq - a_z rq) \\ -m_y vp + m_x qu + m(-a_x rp + a_z(q^2+p^2)) \\ -(J_z - J_y)rq + J_{xz}pq + ma_z(ur - pw) \\ -(J_x - J_z)pr + J_{xz}(r^2-p^2) + m(a_x(vp-qu) - a_z(wq-rv)) \\ -(J_y - J_x)qp - J_{xz}qr + m(-a_x(ur-pw)) \end{bmatrix} \tag{2.47}$$

where,

$$\begin{aligned} m_x &= m - X_{\dot{u}} \\ m_y &= m - Y_{\dot{v}} \\ m_z &= m - Z_{\dot{w}} \\ J_x &= I_x - M_{\dot{p}} \\ J_y &= I_y - L_{\dot{q}} \\ J_z &= I_z - N_{\dot{r}} \end{aligned}$$

This form of the vector equations considers the z axis positive pointing in the *down* direction [4], [14], different from [16].

2.4.4 Gravitational forces vector, G

This vector is described by,

$$G = G(Y(t), \lambda_{31}, \lambda_{32}, \lambda_{33}) \tag{2.48}$$

where,

$Y(t)$ = State vector containing 6 positions x, y, z, ϕ, θ, ψ.
λ_{ij} = Direction Cosine Matrix (DCM) elements.

This vector contains all forces produced by *weight* and *buoyancy*, referred to the North-East-Down (NED) earth axes.

The change from a coordinate system to another can be produced by matrix $C_{b/a}$ as shown in figure 2.27 [18].

$$u^b = C_{b/a}u^a \qquad u^b = \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \tag{2.49}$$

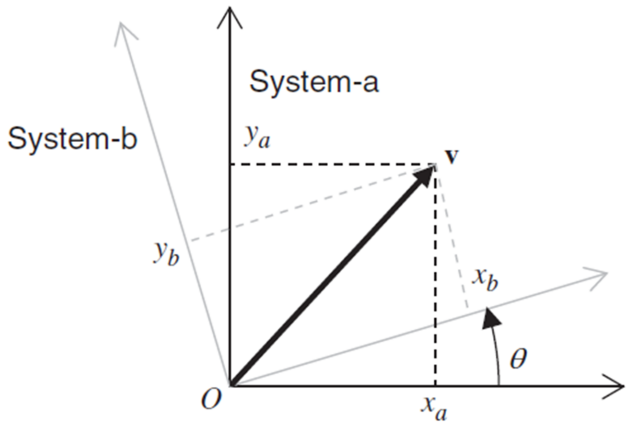


Figure 2.27: System coordinate rotation produced by matrix $C_{b/a}$ [20].

The change from coordinate system XYZ to NED can be produced by 3 consecutive axis rotations, this way the Direction Cosine Matrix (DCM) is generated.

$$DCM = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} \end{bmatrix} \tag{2.50}$$

where,

$\lambda_{11} = cos(\theta)cos(\psi)$
$\lambda_{12} = cos(\theta)sin(\psi)$
$\lambda_{13} = sin(\theta)$
$\lambda_{21} = -cos(\phi)sin(\psi) + sin(\phi)sin(\theta)cos(\psi)$
$\lambda_{22} = cos(\phi)cos(\psi) + sin(\phi)sin(\theta)sin(\psi)$
$\lambda_{23} = sin(\phi)cos(\theta)$
$\lambda_{31} = sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi)$
$\lambda_{32} = -sin(\phi)cos(\psi) + cos(\phi)sin(\theta)cos(\psi)$
$\lambda_{33} = cos(\phi)cos(\theta)$

where ϕ, θ, ψ are the sferical coordinates in the NED plane (figure 2.28). If both planes are aligned (XYZ initial conditions = NED), then these angles represent the *Roll*, *Pitch* and *Yaw* of the blimp in the XYZ plane too.

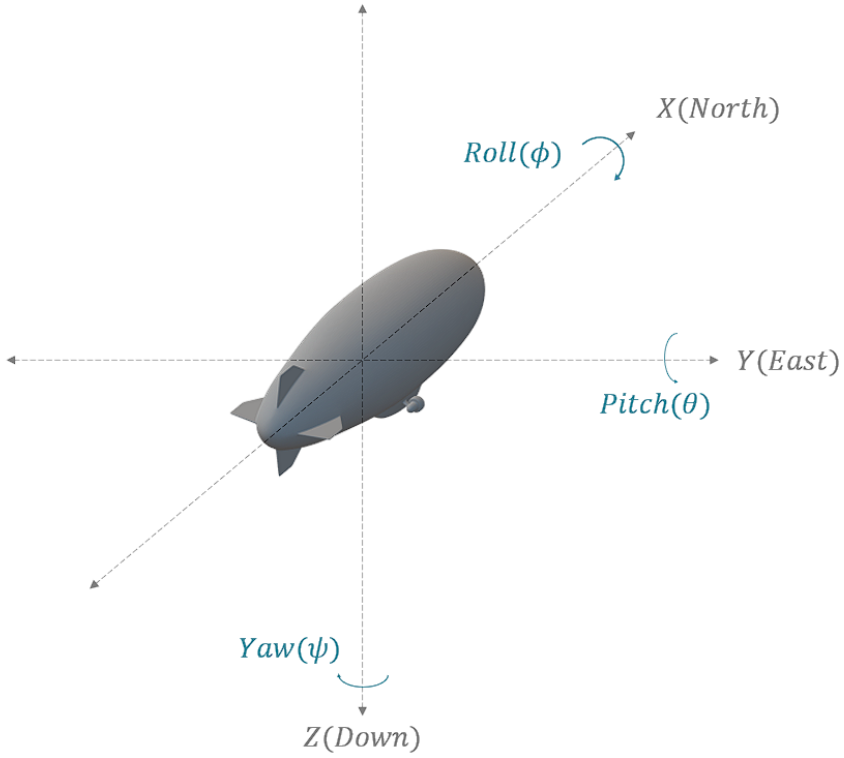


Figure 2.28: NED coordinate system.

The importance of the DCM is crucial because gravitacional forces are acting in the NED coordinate system of the blimp, which is moving in the XYZ plane. With this matrix, forces from NED can be represented in XYZ. Particularly,

$$\begin{bmatrix} V_{north} \\ V_{east} \\ V_{down} \end{bmatrix} = DCM \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.51}$$

The gravitational forces vector can be represented by,

$$G = \begin{bmatrix} \lambda_{31}(W-B) \\ \lambda_{32}(W-B) \\ \lambda_{33}(W-B) \\ \lambda_{32}a_z W \\ (\lambda_{31}a_z - \lambda_{33}a_x)W \\ \lambda_{32}a_x W \end{bmatrix} \tag{2.52}$$

where,

$W = mg$, weight force acting on the blimp.
$B = \rho_{air} g Vol$, buoyancy force acting on the blimp.

2.4.5 Propulsion forces vector, P

This vector sets the input forces to the system, it is composed by thrust forces generated from motors. The location of motors is presented in figure 2.29.

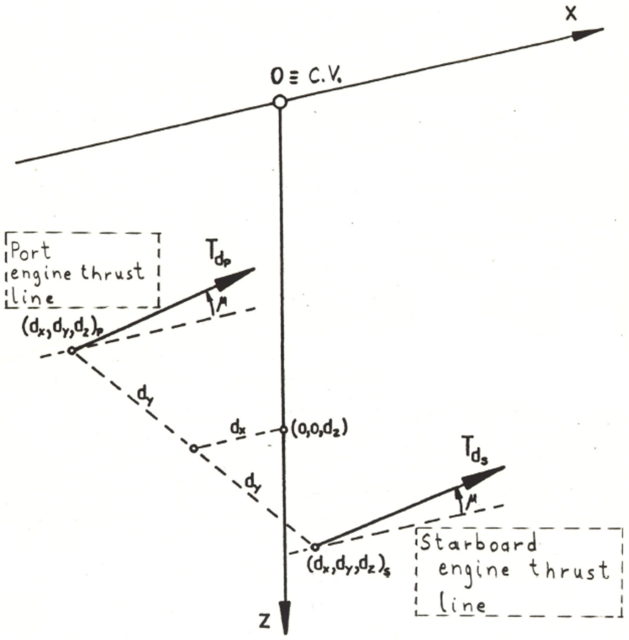


Figure 2.29: Motor distribution [4].

where T_t is not considered,

$$P = \begin{bmatrix} (T_{dp} + T_{ds})cos(\mu) \\ 0 \\ -(T_{dp} + T_{ds})sin(\mu) \\ (T_{dp} - T_{ds})sin(\mu)d_y \\ (T_{dp} + T_{ds})(d_z cos(\mu) - d_x sin(\mu)) \\ (T_{dp} - T_{ds})cos(\mu)d_y \end{bmatrix} \tag{2.53}$$

where,

T_{dp} = Port motor thrust.
T_{ds} = Starboard motor thrust.
μ = Motors angle of elevation.
d_x = Motors x axis displacement from CV.
d_y = Motors y axis displacement from CV.
d_z = Motors z axis displacement from CV.

2.4.6 Aerodynamic forces vector, A

The complexity of determining this vector resides on the blimp physical body and shape. A pair of deflectors can affect the aerodynamics to behave in so many different ways depending

on their location, area and quantity. There are many studies that try to explain these dynamics on an airship [3], [4], [14], [15], depending on the the shape of it.

This vector is of so much significance, because it contains the forces acting opposite to the velocity. This means that they are the only ones that can make the blimp stop by loosing energy, and so in a control point of view, this vector sets the stability of the system.

There are different ways to obtain the aerodynamic coefficients of the blimp. Jones [3] proposes an analytical way making use of the location of deflectors, their size and shape, and also other considerations like the gondola or the shape of the body. This scheme is also used in [15]. On the other hand Gomes [4], [14], carried out a practical investigation of airship aerodynamics by studying the forces acting on a scale blimp in a wind tunnel and tabulating the corresponding coefficients.

Both strategies have been used in many other investigations, and Gomes's work was even used in flight simulators. Sadly, when incorporated to the matlab blimp simulation, these techniques failed because the aerodynamic forces acting on the small scale blimp where too big and made the whole system unstable.

As a solution to this problem, and the need to have friction forces acting on the blimp, there was the possibility to express these forces proportional to the velocities and in an opposite direction. Finally, the aerodynamic forces vector proposed for this simulation, and that presented good simulation results was,

$$A = \begin{bmatrix} -0.5C_D||u||Area\ \hat{u} \\ -0.5C_Y||v||Area\ \hat{v} \\ -0.5C_L||w||Area\ \hat{w} \\ -0.5C_l||p||Vol\ \hat{p} \\ -0.5C_m||q||Vol\ \hat{q} \\ -0.5C_n||r||Vol\ \hat{r} \end{bmatrix} \tag{2.54}$$

where,

C_D = Drag coefficient
C_Y = Sideforce coefficient
C_L = Lift coefficient
C_l = Rolling moment coefficient
C_m = Pitching moment coefficient
C_n = Yawing moment coefficient
Vol = $\frac{4\pi abc}{3}$, volume of an ellipsoid.
$Area = Vol^{2/3}$, surface area of an ellipsoid.

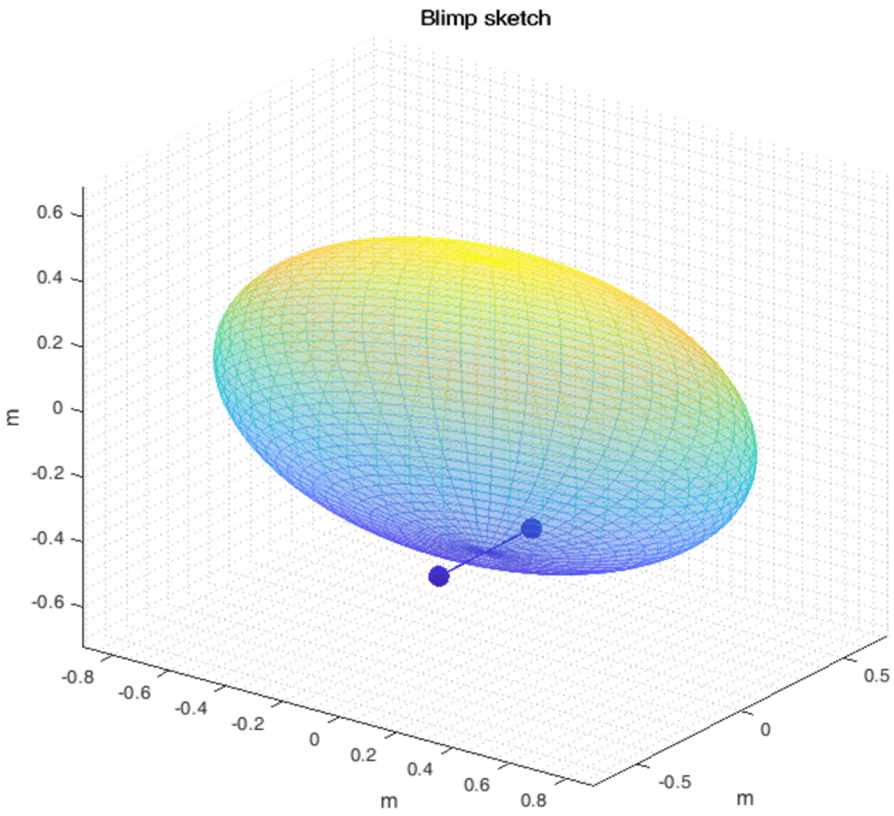


Figure 2.30: Blimp sketch.

CHAPTER 3

Results

All simulation codes used in this work can be found at this GitHub project, they are briefly described below,

I. **6DOF blimp dynamics simulation:** is a Matlab script based on section 2.4 that simulates the behavior of a small scale blimp. The inputs are defined in the code and plots come out to show the results. Its composed of near 500 lines of code.

II. **NARX model ANFIS and training implementation:** is a Matlab script based on section 2.1 where all equations for the operation and training of an ANFIS are involved. In this script an ANFIS for the SISO model of a DC motor is trained and results are shown in plots. Its composed of near 600 lines of code.

III. **Dynamic Matrix Control with ANFIS:** is a Matlab script based on section 2.2 that trains a 4 Input ANFIS for the blimp model to control u (speed in the X axis). This script trains the ANFIS, saves the trained parameters and uses them for the proposed control strategy in a simulation loop. Its composed of near 1800 lines of code.

IV. **NARX ANFIS training for blimp dynamics:** Is a C++ script that simulates the dynamics of the blimp (given inputs) and trains a 4 input ANFIS for any chosen state variable of the blimp. It then saves the results into a Matlab file with the trained parameters for plotting them, and also into a text file to copy and paste them into the CUDA C++ control strategy. Its composed of around 1000 lines of code.

V. **Dynamic Matrix Control with ANFIS in CPP:** is a CUDA C++ code that simulates the dynamics of a small scale blimp and implements the proposed control strategy in C++, it does not include kernels or CUDA APIs and the ANFIS is solved as normal C++. Its purpose is to be a point of comparison in the same environment as the CUDA implementation, in this case the ANFIS is solved as normal C++. Its composed of around 1400 lines of code.

VI. **Dynamic Matrix Control with ANFIS in CUDA:** is the CUDA implementation of the previous C++ script. The only difference is that this code computes the ANFIS network in the GPU with the defined ANFIS kernel. Its speed rates are compared with the C++ code. Its composed of around 1500 lines of code.

The importance of these scripts is that they contain the basic algorithms involved in ANFIS training, operation and the proposed control strategy that can be employed easily in any microcontroller given the C++ level of abstraction.

3.1 The Matlab simulations

This section contains the results for the different Matlab scripts. They were designed to study the composition of ANFIS (II), the blimp dynamic equations (I) and the control algorithms (III).

3.1.1 ANFIS training

This Matlab script (*II. NARX model ANFIS and training implementation*) is organized as follows,

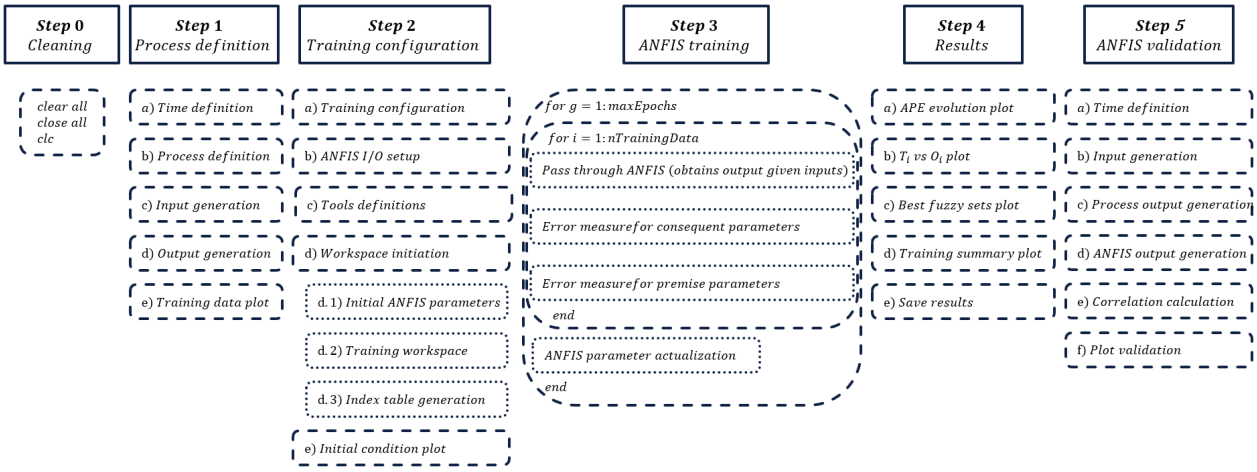


Figure 3.1: ANFIS training Matlab algorithm.

The few next subsections go through every step, their configuration and results.

A. Step 1: Process definition

The test process (the guinea pig) in this case is the polynomial model of a DC motor as given next,

$$h(z) = \frac{B(z)}{A(z)} = \frac{b_0}{a_1 z^{-1} + a_2 z^{-2}} \tag{3.1}$$

Time and process definitions are tabulated below

Parameter	*Value*
t_i	0.1
$step$	0.1
t_f	12.0
b_0	9.217786558421309
a_1	-0.680758164075105
a_2	-0.010439248103841
u_{min}	-12V
u_{max}	12V
y_{min}	-360 $\frac{rad}{s}$
y_{max}	360 $\frac{rad}{s}$
$Fuzzy_{min}$	10
$Fuzzy_{max}$	90

Table 3.1: Configuration of ANFIS training simulation.

Figure 3.2 shows the input (blue) and target (red) signals for the ANFIS training, values are scaled to the mentioned ranges and the input contains additive noise which has shown better results for training the ANFIS.

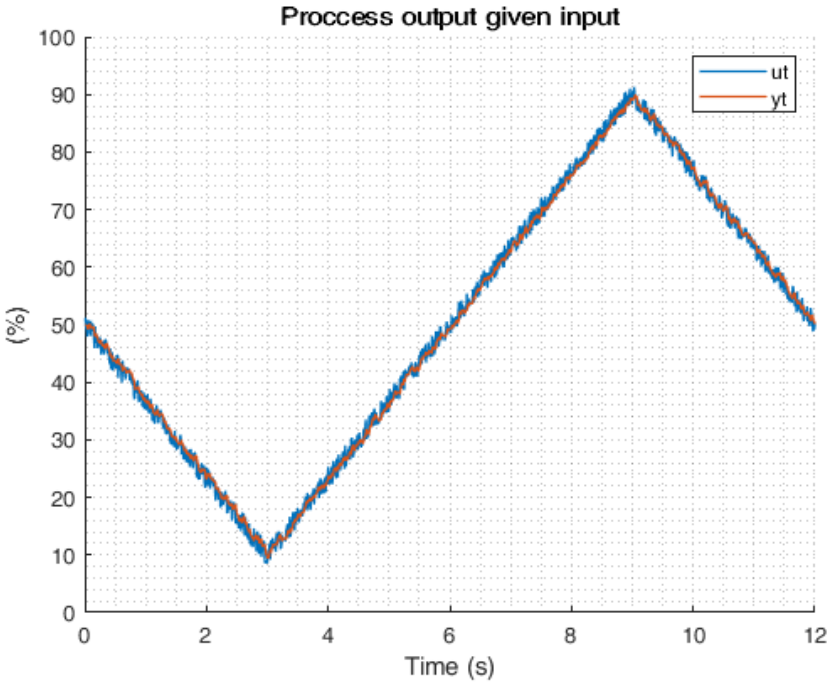


Figure 3.2: Training data for ANFIS network.

When trained with different inputs, different behaviors are exhibited. Better results are obtained when the ANFIS is trained with a triangular input as shown in figure 3.2. Other experiments included training with *ramp* or *step* inputs, but when validated, for the ramp example, the ANFIS imitates the reference and adds a small offset ($\hat{y}$[k] = y[k]+offset), not obtaining the real results ($\hat{y}$[k] = y[k+N]).

The observations behind this phenomena indicate that the ANFIS needs to know what happens when the inputs go from low to high and from high to low as in a triangular signal, rather than the one-way *ramp* input, in order to take actions against hysteresis.

B. Step 2: Training configuration

This section defines the size and configuration of the ANFIS training. Table 3.2 shows the mentioned configurations.

$Parameter$	$Value$
$Epochs$	1000
$nInputs$	2
$nFuzzy$	5
$nRules$	25
In_1	$y[k-1]$
In_2	$u[k-1]$
$Out\ (Target)$	$y[k]$
aIn_0	20
cIn_0	$0:25:100$
$aOut_0$	0.01
$cOut_0$	$0:25:100$

Table 3.2: Configuration of ANFIS training.

Figure 3.3 shows the initial configuration of fuzzy parameters. For **input MFs** the aim is to start with evenly separated functions that can be compared between them, i.e. if the Gauss functions are too tight (small aIn_0), MFs would comprise small parts of the universe ([10% - 90%]) and comparisons (AND operations) would contribute less to make inferences (IF - THEN statements) about the system's behavior. For **output MFs**, they are evenly separated and *aOut* is kept small (near 0), this is because the training eventually drives *aOut* to positive or negative values, and has shown better results.

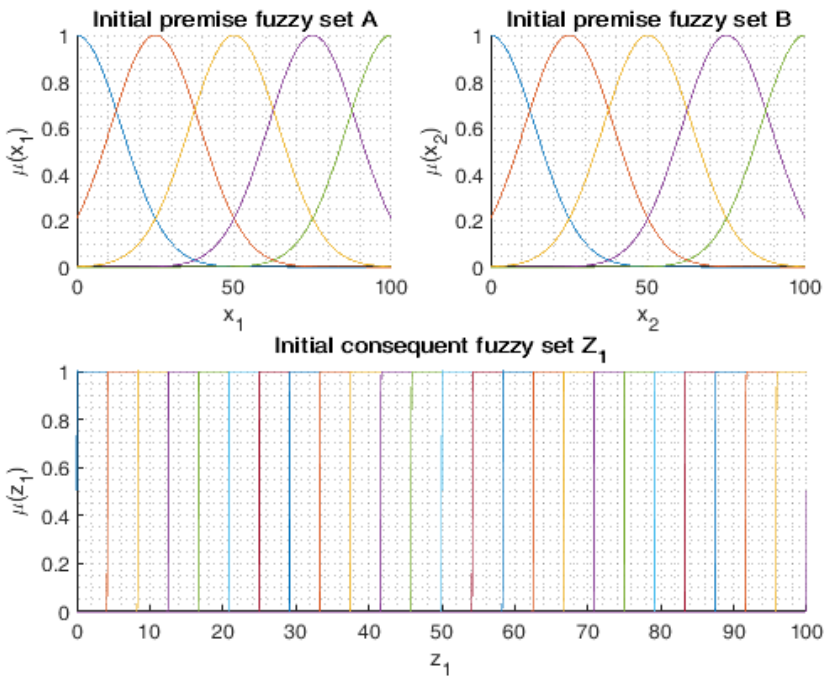


Figure 3.3: Initial fuzzy sets.

C. Step 4: Training summary (training results)

Figure 3.4 shows the APE (%) evolution. The APE evolves rapidly for 200 epochs going up and down due to the changing K rule (section 2.1.4). After this, the ANFIS is gradually adjusted to reach more detailed pattern recognition (or system identification).

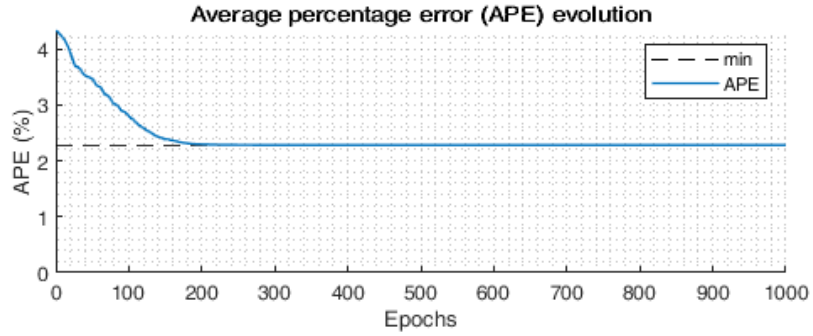


Figure 3.4: APE evolution

Figure 3.5 shows not the final fuzzy sets (those from epoch 1000), but the ones that obtained the minimum APE. For this example the premise fuzzy sets were not adjusted so as to accommodate only the consequent parameters. This way inputs have same initial gains and the ANFIS only changes $\{aOut,cOut\}$ to adapt itself,

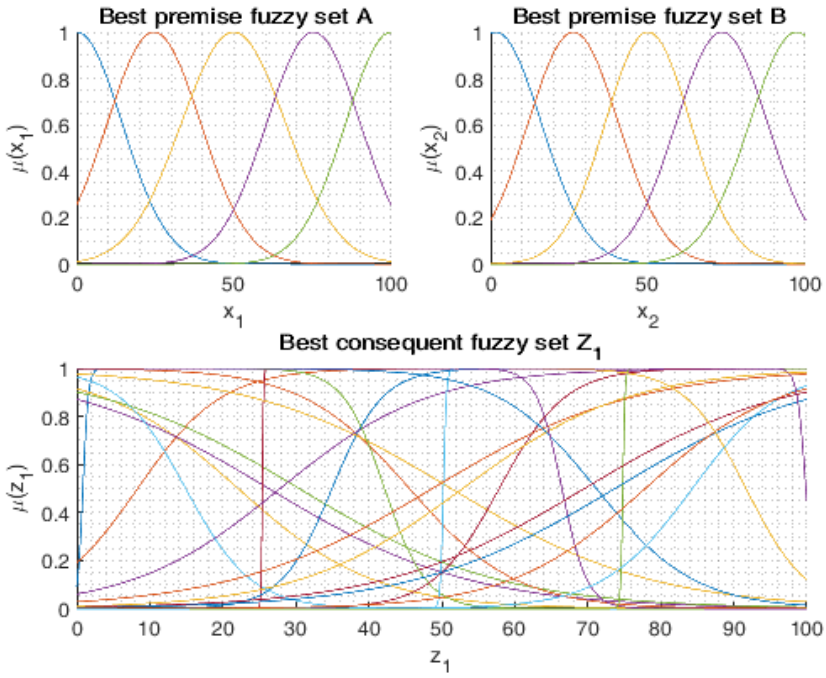


Figure 3.5: Best fuzzy sets obtained from ANFIS training.

Figure 3.6 [1] shows the trained ANFIS results,

[1]The blue vertical line is produced because the network was trained not including the first training data elements so they are set to 0 in the plot.

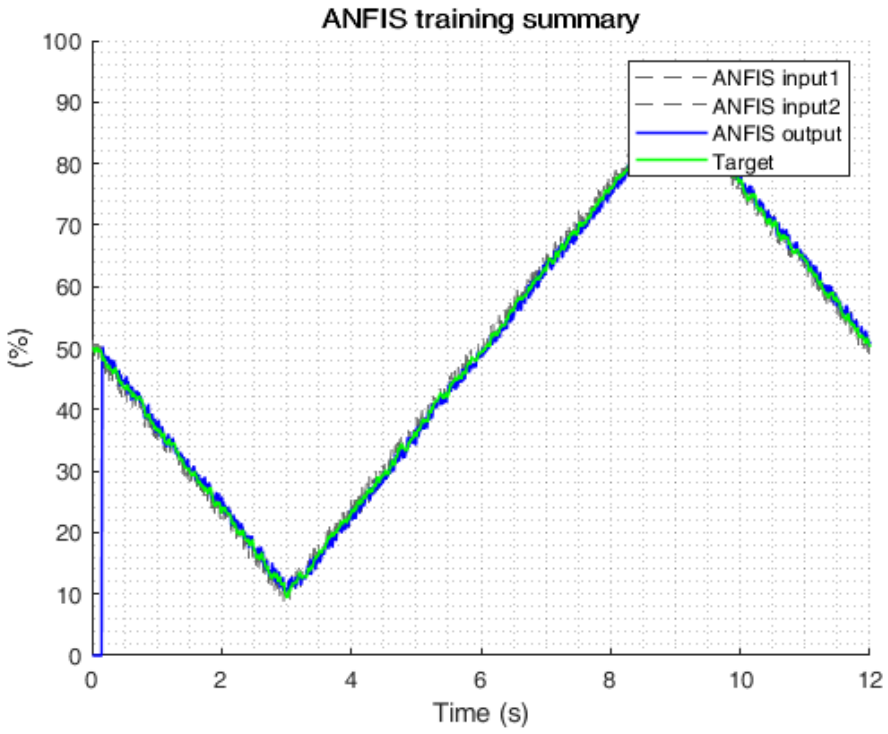


Figure 3.6: ANFIS training summary.

This figure exhibits the non-linear regression reached by the ANFIS. As it can be seen, the blue line is right next to the green line on the right side. This is produced by the use of past values ($y[k-1]$) to obtain present values ($y[k]$).

D. Step 5: Validation

In this step the ANFIS output is calculated using $In_1 = y[k]$, $In_2 = u[k]$ to generate $Out = y[k+1]$.

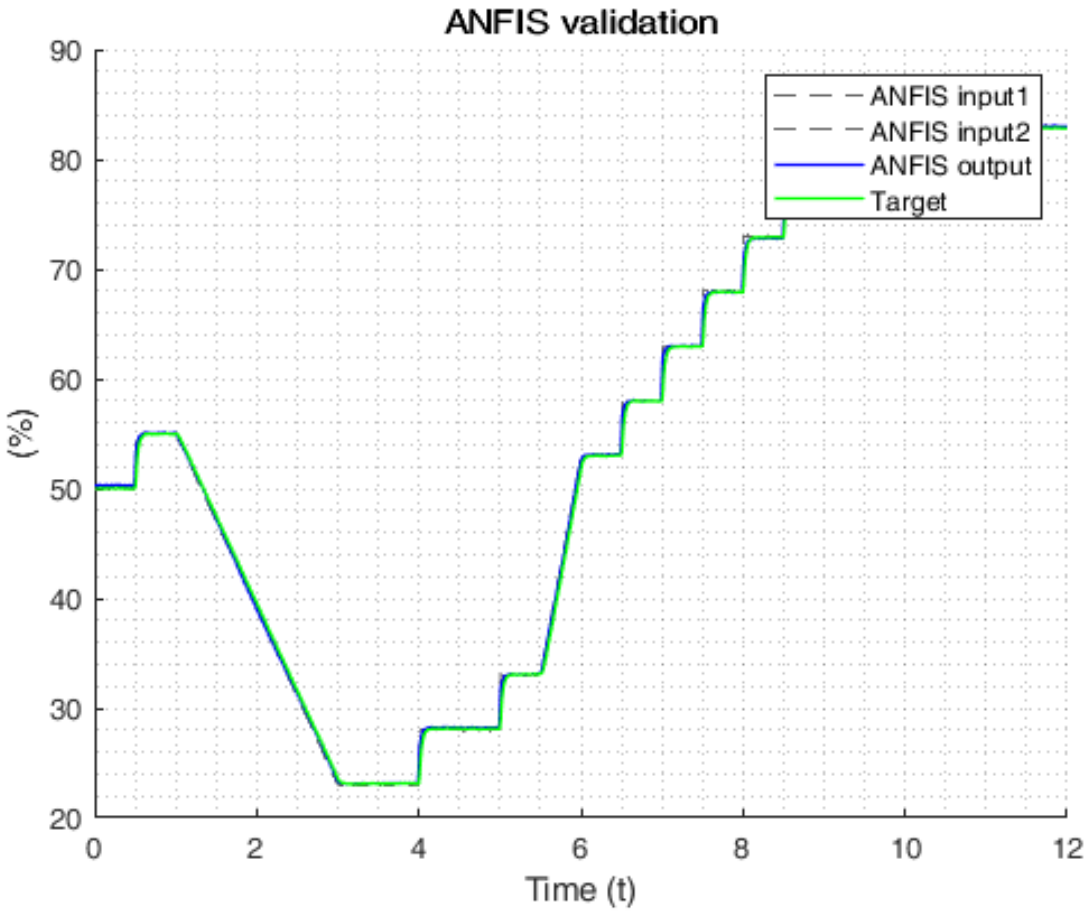


Figure 3.7: Trained ANFIS validation.

The behavior of the ANFIS output (blue line) is practically the same as the process output (green line) but **ahead** an amount equal to $T = 0.1s$. This way the ANFIS can be used as a prediction model of the process.

A 600 lines of code script was designed to train and validate an ANFIS for a SISO process. The algorithm fulfills its objective by correctly training an ANFIS and verify its behavior with a validation set. The script can be applied directly to any SISO model with an extension to MIMO models or for non-linear regression purposes. In this case its used in modelling a low scale blimp MIMO system and applied to a model-based predictive control strategy.

3.1.2 Blimp simulation

This Matlab code (*I. 6DOF blimp dynamics simulation*) was designed to study the blimp dynamic equations and its integration into the control algorithm (section 2.4). The script is organized as follows,

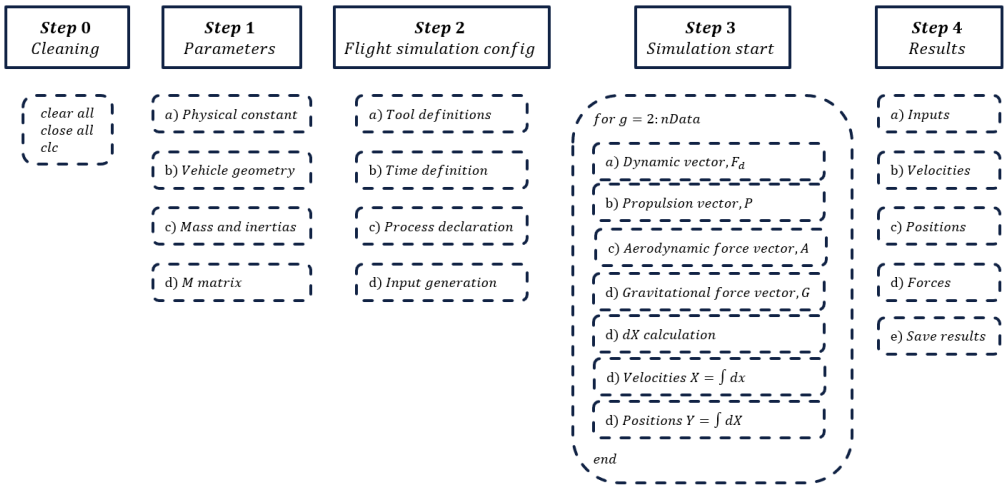


Figure 3.8: Blimp simulation Matlab algorithm.

A. Step 1: Parameter initialization

Here are defined the constants and parameters used in the simulation: physical constants, vehicle geometry and body definitions. A few are listed here while others are calculated,

Parameter	*Value*
a	0.90 m
b	0.45 m
c	0.45 m
Total mass	Vol$\times\rho_{Air}$ kg

Table 3.3: Configuration of blimp parameters.

From table 3.3, the body shape is a prolate spheroid with $a = 2 \times b = 2 \times c$ and the density of the blimp is set the same as air so the blimp's buoyancy and weight forces are equal but of opposite directions.

B. Step 2: Flight simulation configuration

Table 3.4 shows the blimp simulation parameters and figure 3.9 the input signals used,

Parameter	*Value*
t_i	10.1 s
step	0.1 s
t_f	560.0 s

Table 3.4: Configuration of blimp flight simulation.

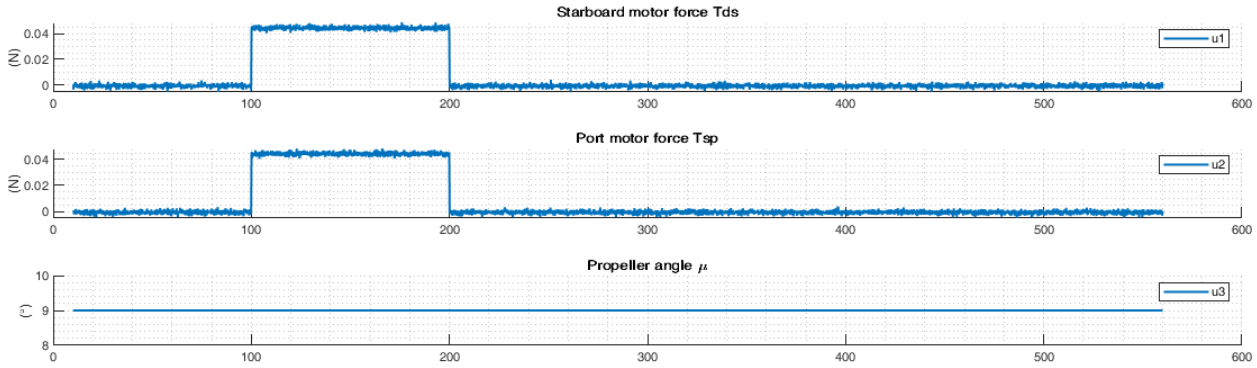


Figure 3.9: Blimp simulation inputs.

In this example inputs are kept simple for demonstration purposes. Forces are very low due to the mass of the blimp; if greater forces are used, the blimp would exhibit too much acceleration and aerodynamical forces couldn't be capable of stabilizing the blimp in reasonable times.

C. Step 4: Results

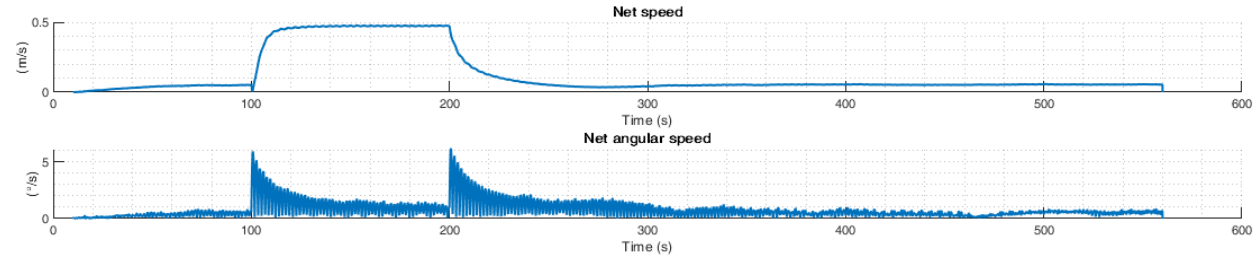


Figure 3.10: Blimp simulation net speeds.

Linear net speed does not reach 0m/s due to the added noise in T_{ds} and T_{sp}. The angular net speed increases when heavy changes in the inputs are made, this is because when a force is applied from the motors to the center of gravity, a torque is produced; and with that, angular velocity. Observations indicate that the blimp's net speed decreases over time when inputs are turned off which is a behavior expected produced by the friction of wind (aerodynamic forces), granting the simulated blimp with stability and the code with reliability.

Figure 3.11 exposes the resulting dynamics of the 6 blimp velocity states, u, v, w, p, q, r given the discussed inputs.

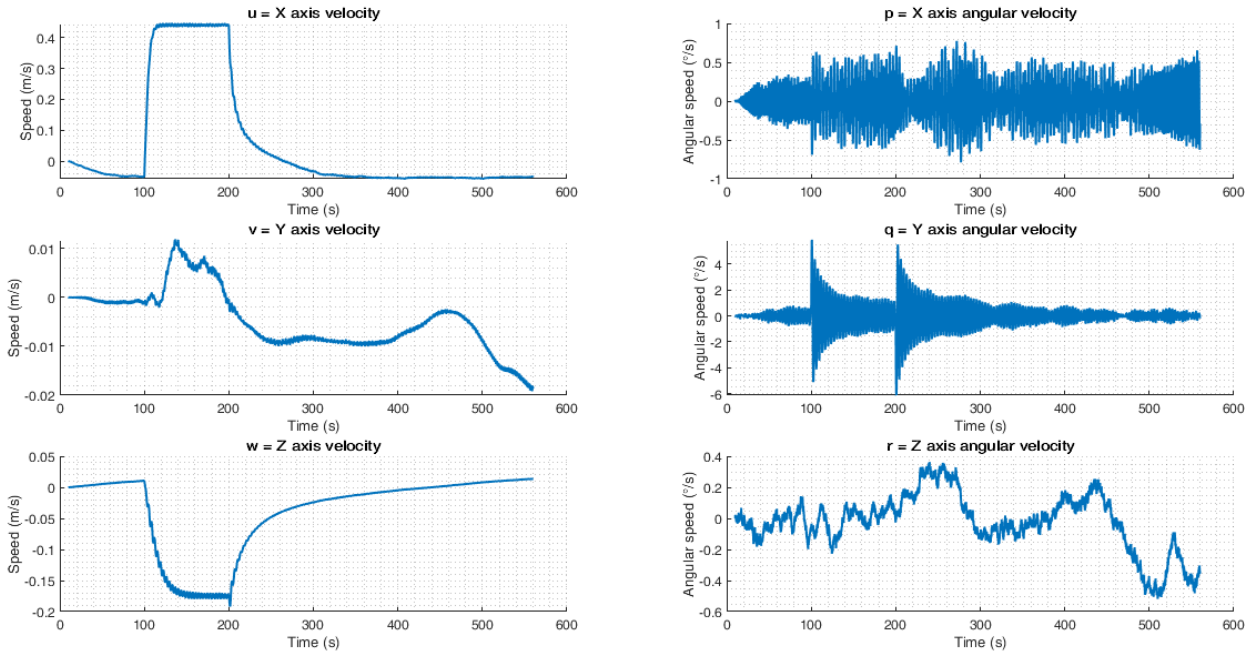


Figure 3.11: Blimp simulation velocities (X states).

Observations for each graph are listed below,

- **X axis linear velocity u** its directly related to the inputs as T_{ds} and T_{sp} are practically the same, the blimp advances in the X direction.
- **Y axis linear velocity v** is near 0m/s because there are no direct ways to produce this velocity, only due to the intrinsic non-linearities involved in F_d.
- **Z axis linear velocity w** is produced by the propeller angle and motor thrust. As the angle is positive (constant 9°), when the thrust goes to 0m/s in time = 200s, the blimp exhibits the non minimum phase characteristic: accelerating before decelerating.
- **X axis angular velocity p** and **Y axis angular velocity q** oscillate around 0°/s due to the gravitational DOWN force acting on the center of gravity which is distant to the center of volume, producing torques until the CG matches the CV in the Z axis.
- **Z axis angular velocity r** is not directly altered by the gravitational forces but by the difference between T_{ds} and T_{sp} produced by the added noise and nonlinearities involved.

These observations provide the code with trustability about the performance of the simulated blimp versus what can be expected of it in the real (physical) world. This comparison is needed for the blimp dynamics simulation to be used in the control strategy.

Figure 3.12 displays the integrals of the X states, they show the evolution of position and orientation during the simulation.

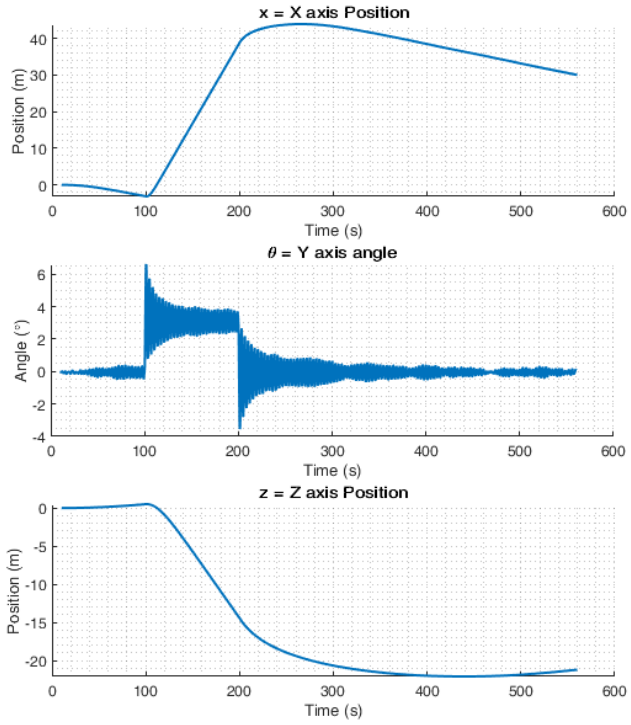


Figure 3.12: Blimp simulation positions (Y states).

States $\mathbf{x}$, $\mathbf{z}$, θ express explicitly how the blimp moved given the designed inputs: the blimp goes forward ($\mathbf{x}$) and upwards ($\mathbf{z}$), and it goes oscillating due to the body's inertia and step input (θ). Behavior expected given the speed signals, proving that the simulation script is reliable for its use in the control strategy implementation.

Figure 3.13 exhibits all forces involved in the simulation for the 3 linear forces and 3 angular forces,

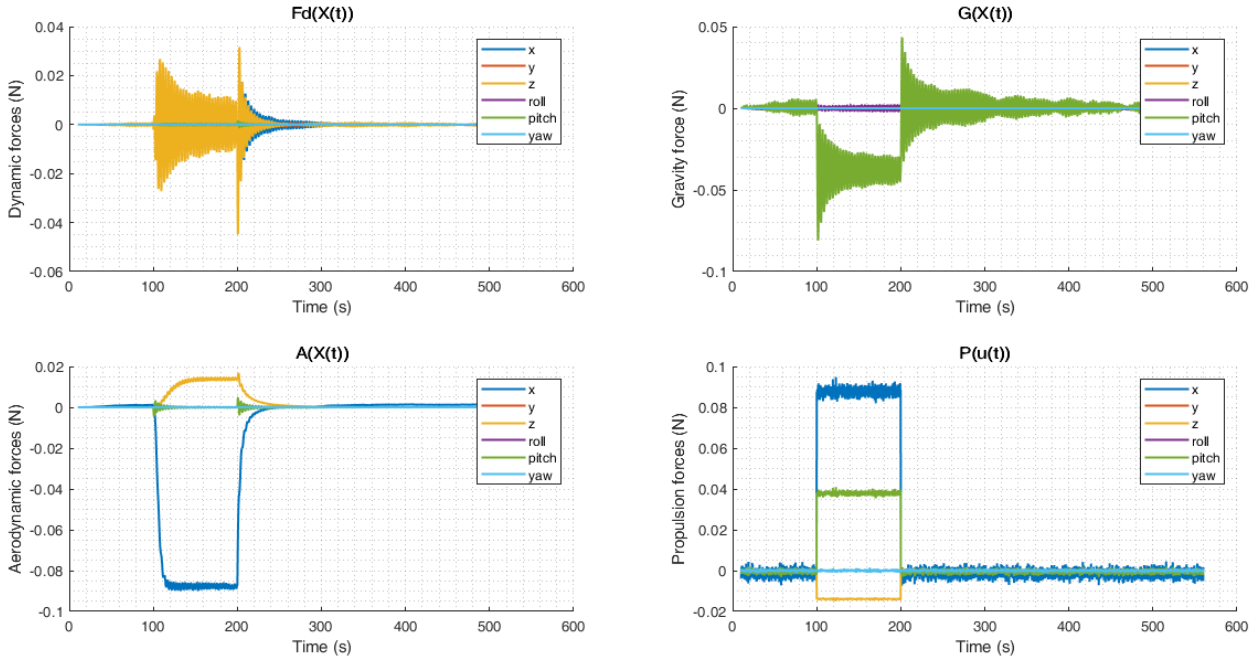


Figure 3.13: Forces participating in the blimp simulation.

The *roll* and *pitch* **gravitational forces** (G) act over the blimp when the CG is not aligned with the CV in the Z axis, proving that the oscillating angular speeds p and q are produced from this forces. **Aerodynamic forces** (A) are proportional to the corresponding velocity but in the opposite direction, they yield the blimp with stability as it eliminates energy from the process. **Propulsion forces** (P) are the input forces that make the blimp move; when set to 0N the blimp's net speed tends to go to 0m/s due to aerodynamic forces (approximately after $t = 200s$, fig. 3.10). These final observations provide the script with more robustness and trustability about the blimp dynamics simulation.

A 500 lines of code script was designed to simulate the behavior of a small scale blimp. The blimp simulation manifests a behavior similar to what is expected in the real world, granting the code with a solid background for its use in any application; in this case, the training of an ANFIS and its use in a model-based predictive control strategy.

3.1.3 Control simulation

This Matlab script (*III. Dynamic Matrix Control with ANFIS*) was designed to study the control strategy (section 2.2) for its implementation in a CUDA architecture. Its divided in 3 stages: the first stage generates training data from the blimp simulation, next a 4 input ANFIS is trained for the linear speed u (X axis linear speed), and lastly the control algorithm is applied and simulated. This script is organized as follows,

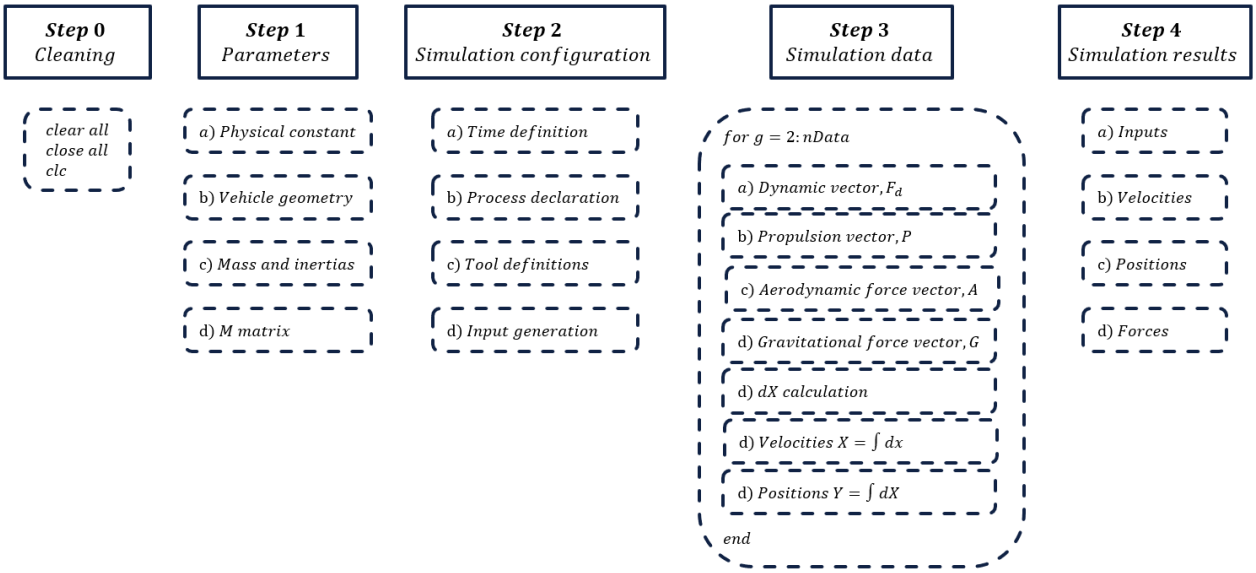


Figure 3.14: Control simulation algorithm I: training data generation.

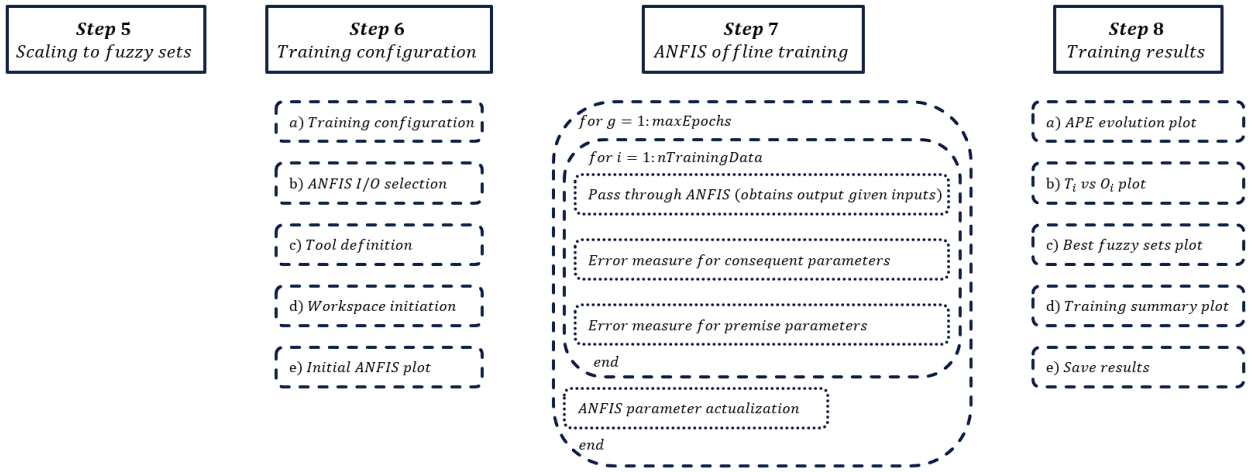


Figure 3.15: Control simulation algorithm II: ANFIS training.

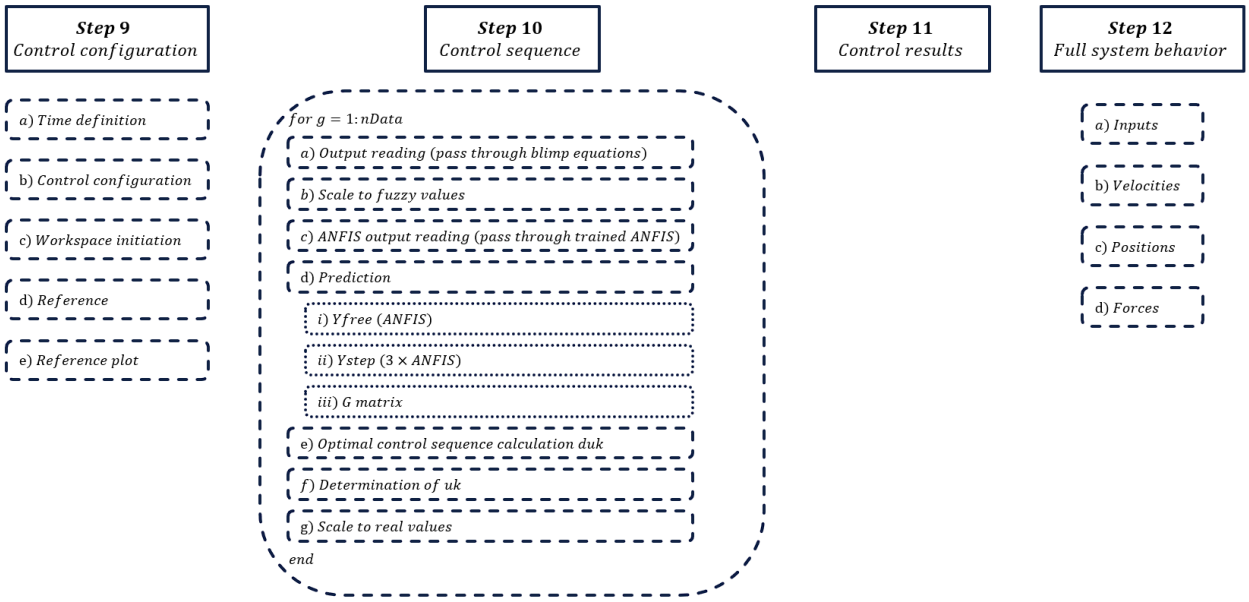


Figure 3.16: Control simulation algorithm III: Control strategy implementation.

A. Step 1: Parameter initiation

This part is dedicated to initiate the blimp body's characteristics, parameters and physical constants.

Parameter	*Value*
a	0.90 m
b	0.45 m
c	0.45 m
Total mass	Vol$\times\rho_{Air}$ kg

Table 3.5: Configuration of blimp parameters.

B. Step 2: Process definition

Time array is defined and inputs are generated. For control purposes an arrange for the input variables is made,

$$T_{ds} = \gamma \times T_{max} \tag{3.2}$$
$$T_{sp} = (1-\gamma) \times T_{max} \tag{3.3}$$
$$\mu = \mu \tag{3.4}$$

This way T_{max} controls the thrust directly and $\gamma,\ \mu$ the steering. Time configuration is shown in the table below

Parameter	*Value*
t_i	10.1 s
step	0.1 s
t_f	560.0 s

Table 3.6: Configuration of blimp flight simulation.

C. Step 4: Simulation results

Figure 3.17 shows the selected inputs and their transformed equivalents.

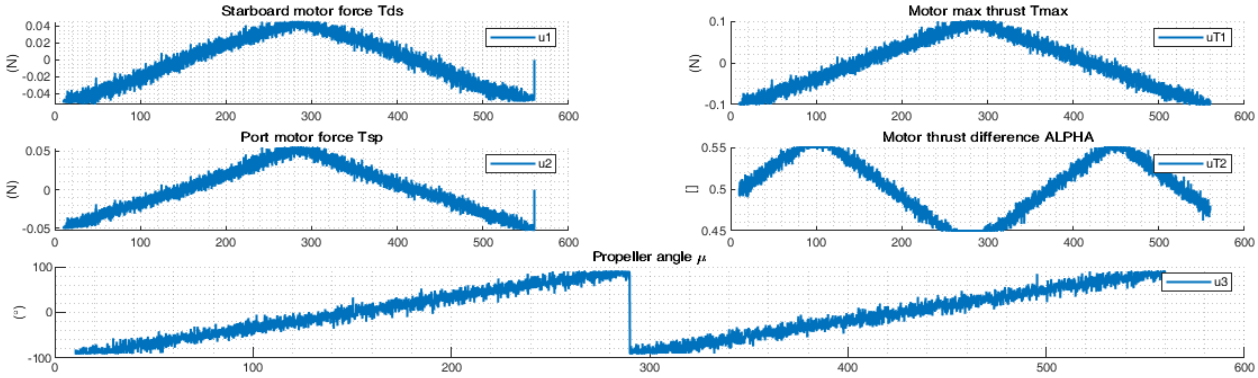


Figure 3.17: Blimp simulation inputs.

Triangular signals with added noise inputs are used as they produce data richer in information for ANFIS training as demonstrated in section 3.1.1.

Simulation results for the 6 velocity states and observations are shown below,

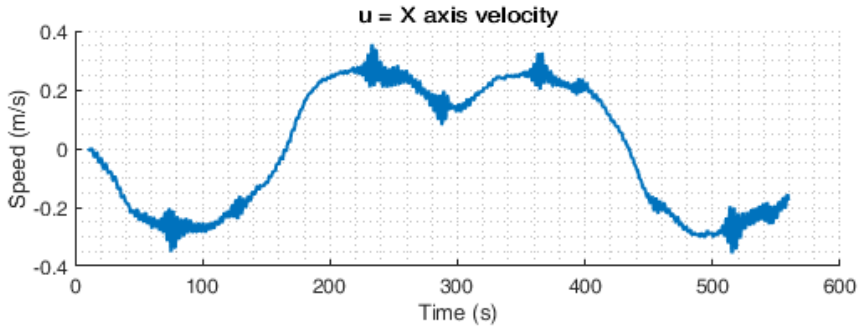


Figure 3.18: Blimp simulation X-axis velocity.

u is the variable to control so it is important that training data include a vast range of operation; in this case the signal span is [-0.3, 0.3]m/s and so is the control range.

D. Step 5: Scaling to fuzzy sets

This part scales the signals into fuzzy values [10%,90%]. The following parameters are used along with the mapping function (section 2.2.2),

Parameter	*Value*	*Parameter*	*Value*
`minFuzzy`	10%	`maxFuzzy`	90%
`minUT1`	-0.1 N	`maxUT1`	0.1 N
`minUT2`	0.45	`maxUT2`	0.55
`minUT3`	-90°	`maxUT3`	90°
`minX1`	-1.0 m/s	`maxX1`	1.0 m/s
`minX2`	-1.0 m/s	`maxX2`	1.0 m/s
`minX3`	-1.0 m/s	`maxX3`	1.0 m/s
`minX4`	-180°/s	`maxX4`	180°/s
`minX5`	-180°/s	`maxX5`	180°/s
`minX6`	-180°/s	`maxX6`	180°/s
`minY1`	-100 m	`maxY1`	100 m
`minY2`	-100 m	`maxY2`	100 m
`minY3`	-100 m	`maxY3`	100 m
`minY4`	-180°	`maxY4`	180°
`minY5`	-180°	`maxY5`	180°
`minY6`	-180°	`maxY6`	180°

Table 3.7: Scaling parameters for all involved signals.

It is important to note that *training data* for **u** extends only to [-0.3,0.3]m/s but limits are set to [-1.0,1.0]m/s. This is not fatal, it only restricts the control range to [-0.3, 0.3]m/s but the limited amount of rules are better exploited to reach better modelling. In other words, if `minX1` and `maxX1` were to be -0.3m/s and 0.3m/s respectively, the control range would also be [-0.3, 0.3]m/s but the model wouldn't be as precise as in [-1.0, 1.0]m/s because consequent MFs would be too much apart. Another way of seeing it is by increasing the amount of rules used, but it would raise the complexity of training and calculation of ANFIS.

E. Step 6: Training configuration

Configuration parameters are shown in table 3.8 and initial fuzzy MFs in figure 3.19,

Parameter	*Value*
$Epochs$	1000
$nInputs$	4
$nFuzzy$	5
$nRules$	625
In_1	$T_{max}[k-1]$
In_2	$\gamma[k-1]$
In_3	$\mu[k-1]$
In_4	$x_1[k-1]$
$Out\ (Target)$	$x_1[k]$

Table 3.8: Configuration of ANFIS blimp training.

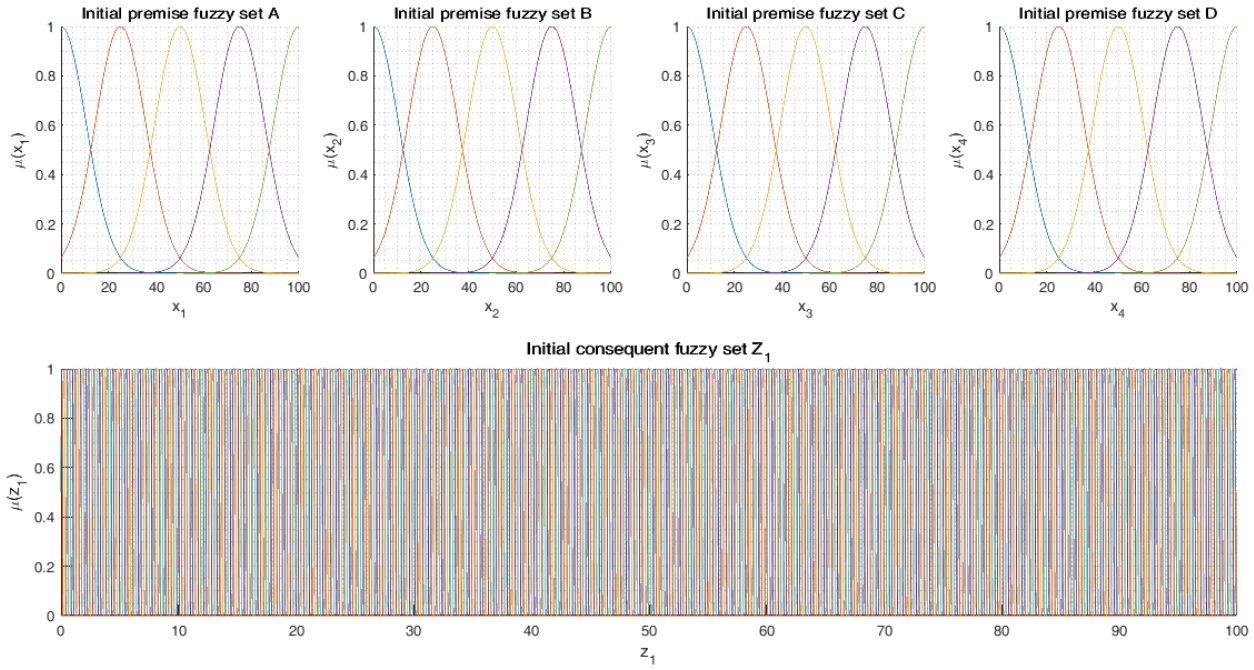


Figure 3.19: Initial fuzzy sets for blimp ANFIS training.

Naturally if more rules where to be used, the ANFIS can obtain better results but the complexity of ANFIS and its training would grow, increasing compilation times.

F. Step 8: Training results

Results of ANFIS training are shown next,

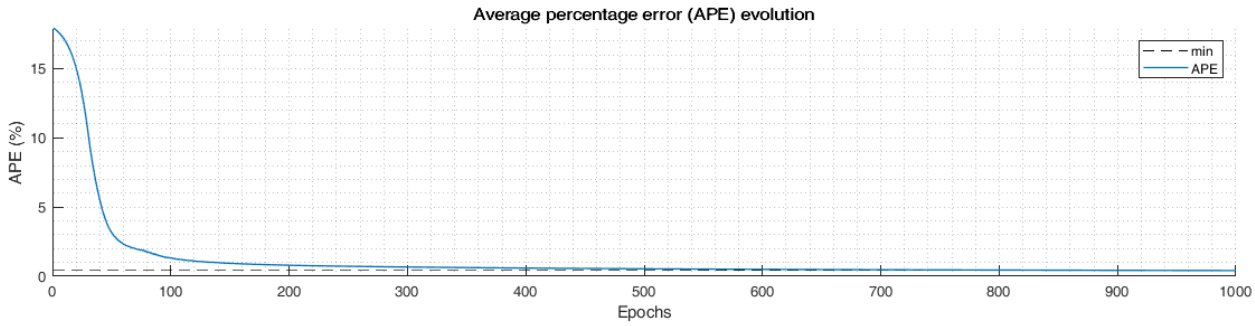


Figure 3.20: APE evolution.

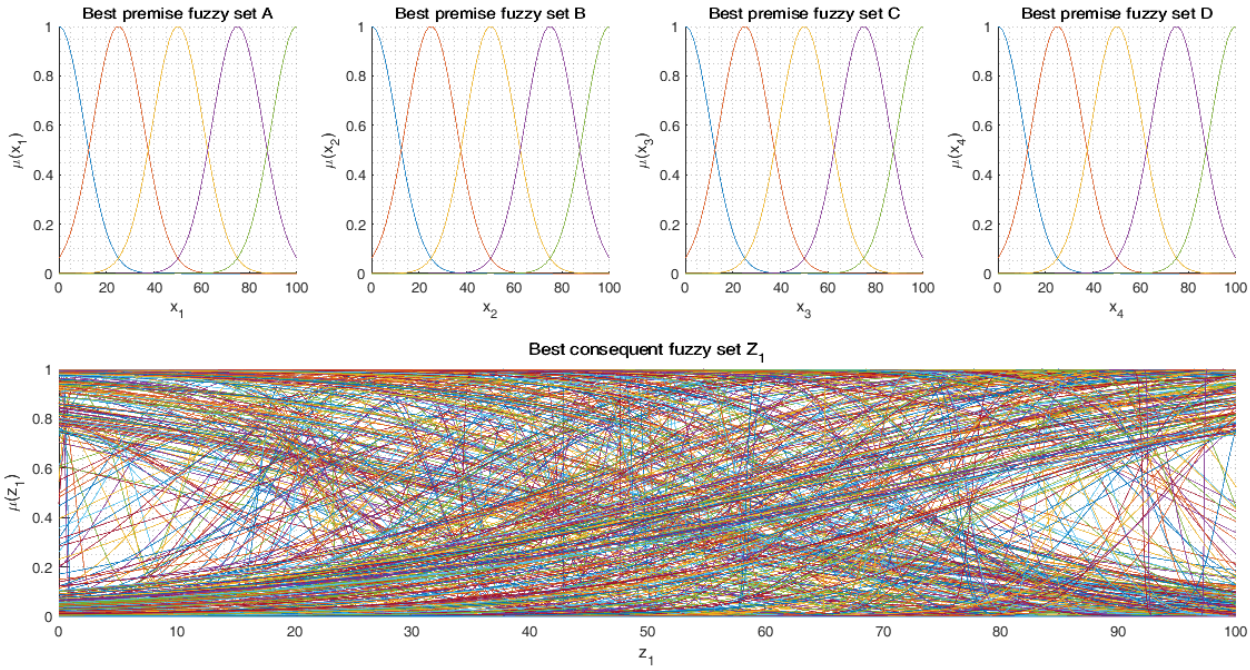


Figure 3.21: Best fuzzy sets for blimp ANFIS training.

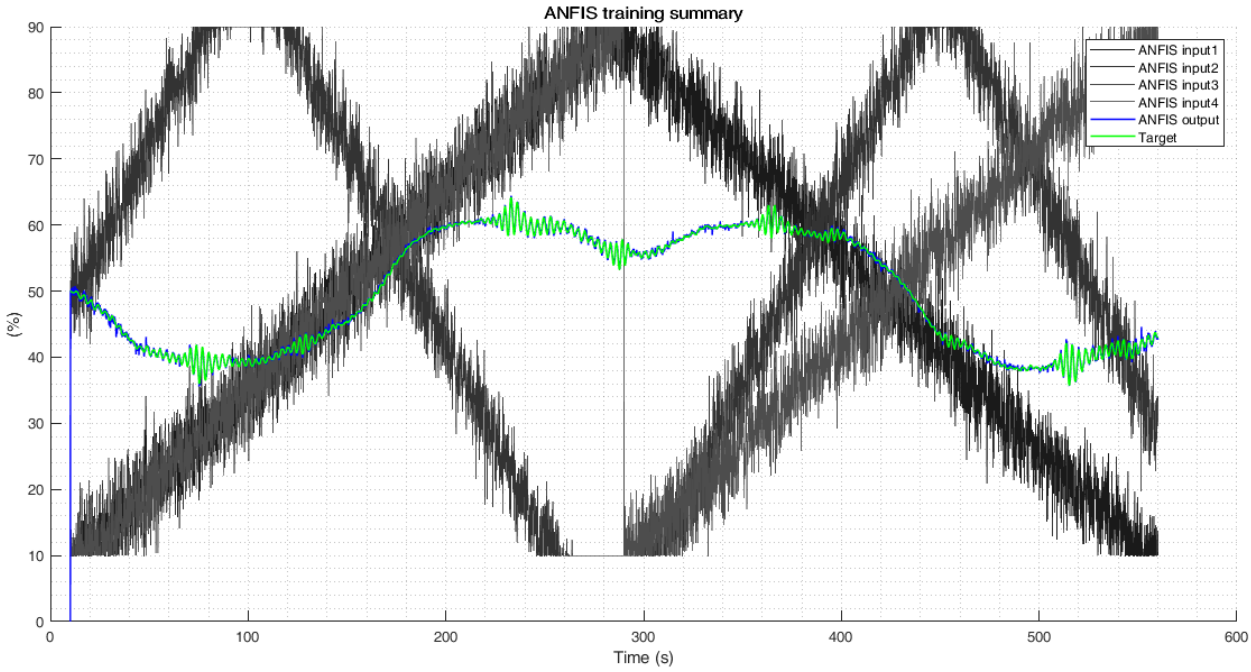


Figure 3.22: Training results for blimp ANFIS training.

Not training input MFs. As shown in figure 3.21, input functions are not trained. This is preferred because this particular control strategy measures the impact of every input for the optimization problem of the controller, and so, if input MFs are not equal some inputs may (will) inform the controller that they contribute with more (or less) weight not because they actually do in the real world, but because they have the advantage given by

the training, and the controller will grant those (or that) particular inputs with more/less (wrong) attention, not following the reference correctly.

G. Step 9: Control configuration

Configuration parameters are shown in table 3.9 and the reference and filtered reference in figure 3.23,

Parameter	*Value*
t_i	0.1
$step$	0.1
t_f	600.0
α	0.9
N_p	4
N_c	3
Q	$1\times$`eye(`N_p`)`
R	$[5, 1000, 5]\times$ `eye(`$N_c \times 3$`)`
Pu_1	6.0
Pu_2	1.0
Pu_3	2.0

Table 3.9: Configuration of the control strategy.

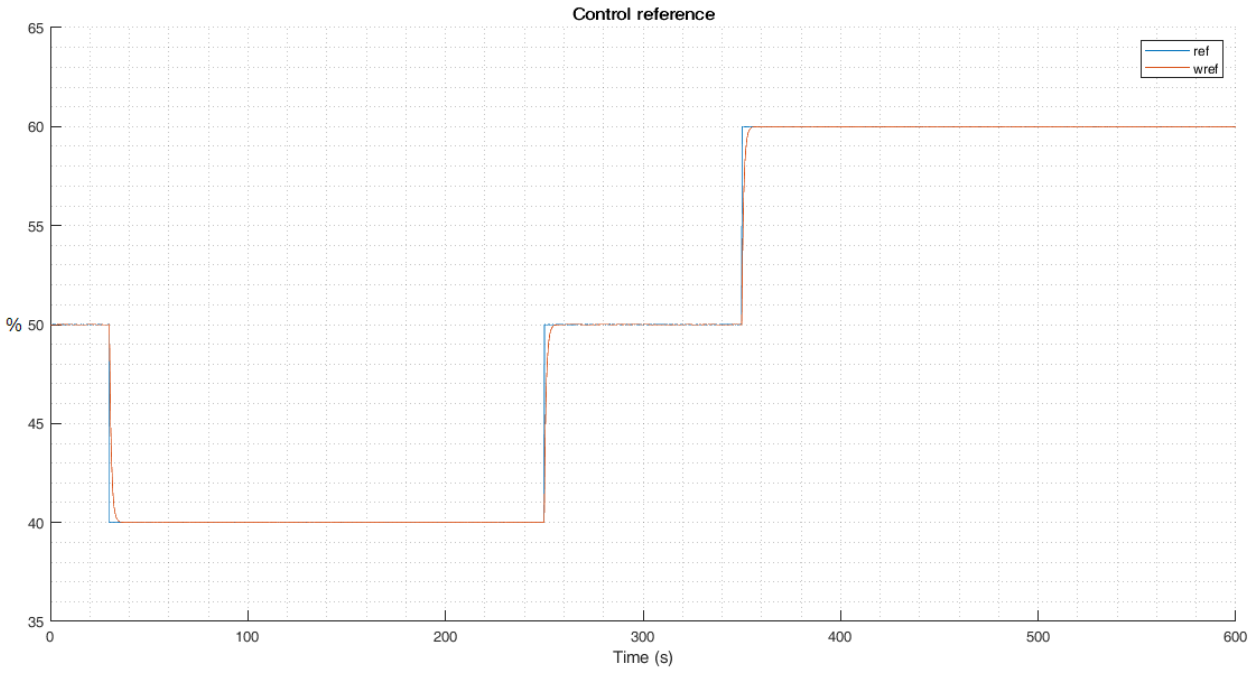


Figure 3.23: Reference and filtered reference.

Very important considerations are involved in the calculation of Ystep, related to where should the step point to and how big the step should be. In this case, the next conditions are set,

- For `YstepU1`, $du_1 = -20$.
- For `YstepU2`, if $\gamma > 50$ then $du_2 = -25$, else $du_2 = +25$.
- For `YstepU3`, if $\mu > 50$ then $du_3 = -35$, else $du_3 = +35$.

These steps are not randomly chosen, but are the ones that presented the best control behavior.

H. Step 11: Control results

Final control results (and results of this thesis) for the proposed control strategy are shown next,

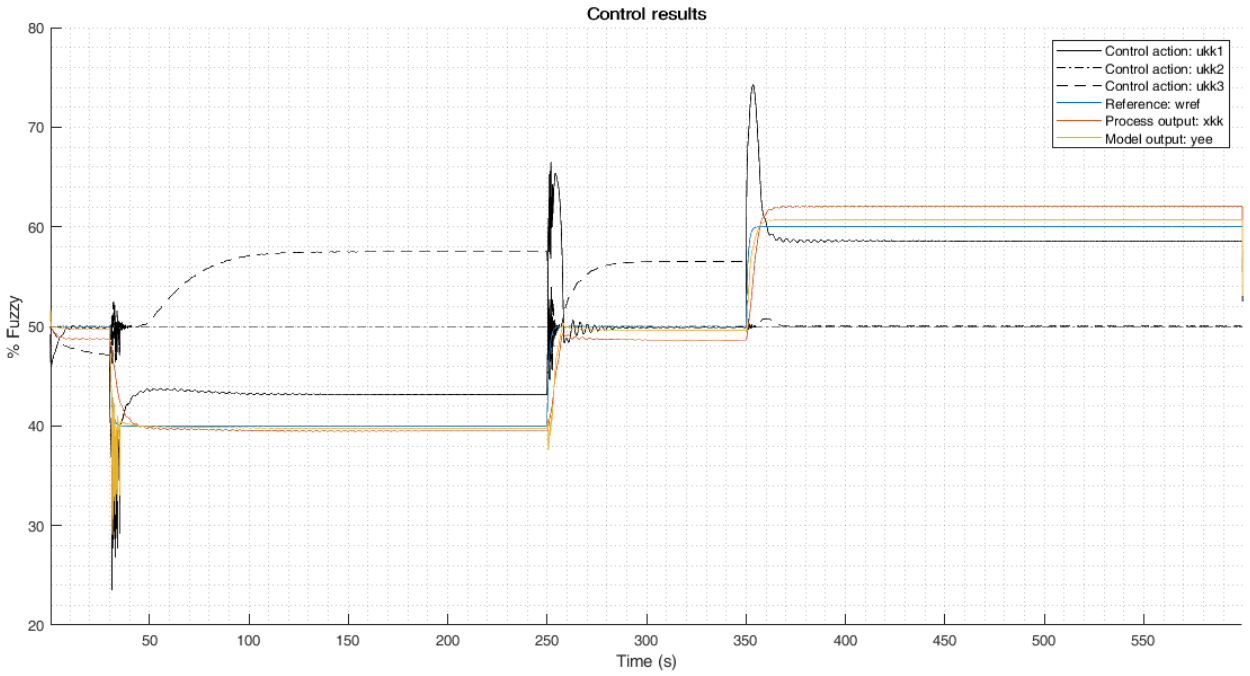


Figure 3.24: DMC+ANFIS control results over the blimp model.

It is very important to note that these results were not achieved directly from the initial proposed strategy but by fine tuning every control parameter involved and presented in step 9.

From figure 3.24 it can seen that the control strategy works well because the process output follows the reference (in the way it cans). A very important thing to note is that the 3 inputs signals are excellently selected to control the behavior of the plant: the controller chooses not to activate γ (it stays in 50%) because it doesn't need to; it activates the thrust T_{max} in a measured way and, along with the propeller angle μ the non minimum phase behavior is controlled and the blimp doesn't oscillate until it reaches the reference point (a behavior that can be expected from a set of non compensated inputs). In conclusion, the control strategy optimizes its 3 resources T_{max}, γ, μ in a balanced way to control the 3-input 1-output non-linear MIMO system under the given configuration specifications.

I. Step 12: Full system behavior

Finally for observation purposes, results for the blimp dynamics are plotted,

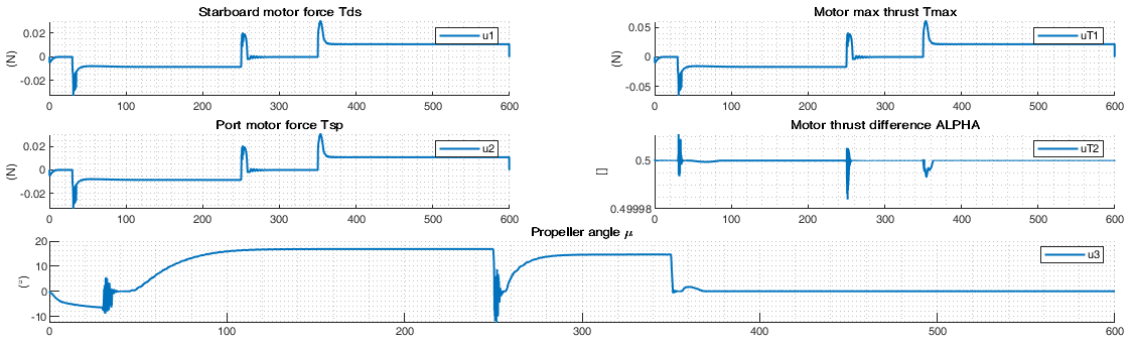


Figure 3.25: Blimp inputs as automated control actions.

Figure 3.25 shows the blimp automated inputs, it can be seen how the controller optimizes its resources to accomplish the goal of following the reference in a measured way given by the QR specifications. It is important to note that the inputs affect the blimp in a nonlinear manner, and control responses are acting very well for the operating range. As the control algorithm calculates the dynamic matrix in every step, control actions are re-weighted in every step to achieve the desired output; other basic controllers would out-measure the control actions (because of nonlinearities) and gain scheduling techniques would be required, which is not the case for this controller.

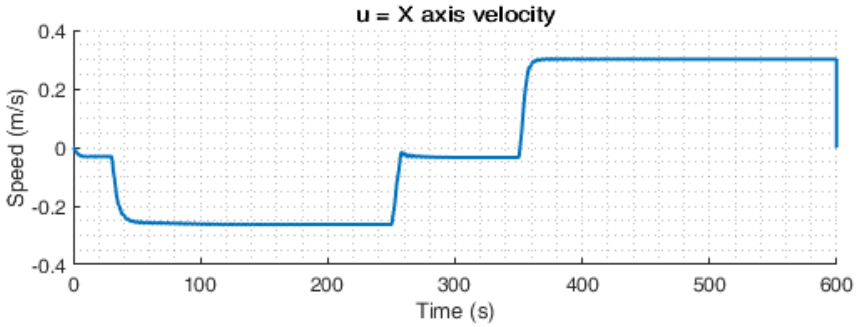


Figure 3.26: Blimp x-axis speed evolution.

Figure 3.26 shows the results of the blimp's x-axis velocity when controlled by the proposed control strategy. Non-minimum phase behaviors are reduced and the blimp's speed is softly managed (aspired behavior for aerodynamic systems) to the desired reference.

3.2 The CUDA implementation

This section discusses the results obtained from the CUDA C++ script generated for targeting higher computation speeds and with that, to extend the limitations of the proposed control strategy. The CUDA codes designed for this part of the project are based directly on the Matlab simulations and implemented in a C++ environment with Microsoft Visual Studio 2019. First the ANFIS kernel is exposed and then speed comparisons are presented.

3.2.1 ANFIS Kernel

A very large ANFIS network could have been designed (thousands of rules) but for experimental/practical issues and without the loose of generalization as computations are performed in a parallel way, a kernel for a 4 input, 5 MFs and therefore 625 rules ANFIS is designed.

There are 1024 *CUDA threads* in the one dimensional (x) *CUDA blocks* from the NVIDIA GeForce GTX 1650 GPU (section A. contains the specifications of this particular GPU), and with only 625 threads being used, a kernel employing shared memory is designed,

---

```
__global__ void ANFIS(double* aIn, double* cIn, double* aOut, double* cOut, double* O0, double* O1, double* O2, double* O3, double* O4, double* O5) {
// One 1024 thread block for 625 rules
    __shared__ double O0_shared[nInputs * nFuzzy];
    __shared__ double O1_shared[nInputs * nFuzzy];
    __shared__ double O2_shared[nRules];
    __shared__ double O3_shared[nRules];
    __shared__ double O4_shared[nRules];
    __shared__ double O5_shared;
    __shared__ double aIn_shared[nInputs * nFuzzy];
    __shared__ double cIn_shared[nInputs * nFuzzy];
    __shared__ double aOut_shared[nRules];
    __shared__ double cOut_shared[nRules];
    __shared__ double fi_shared[nRules];

    int index = threadIdx.x;
    const double dev_tol = 1e-6;
    __syncthreads();

    // Layer 1: Fuzzyfication
    if (index < nInputs * nFuzzy) {
        O0_shared[index] = O0[index % nInputs];
        aIn_shared[index] = aIn[index];
        cIn_shared[index] = cIn[index];

        O1_shared[index] = exp(-(O0_shared[index] - cIn_shared[index]) * (O0_shared[index] - cIn_shared[index]) / (aIn_shared[index] * aIn_shared[index]));

        if (O1_shared[index] < dev_tol) {
            O1_shared[index] = dev_tol;
        }
        if (O1_shared[index] > 1.0 - dev_tol) {
            O1_shared[index] = 1.0 - dev_tol;
        }
        O1[index] = O1_shared[index];

    }
    __syncthreads();

    // Layer 2: Permutation
    if (index < nRules) {
        O2_shared[index] = 1.0;
        O2_shared[index] = O2_shared[index] * O1_shared[index / (nFuzzy * nFuzzy * nFuzzy) * nInputs];
        O2_shared[index] = O2_shared[index] * O1_shared[(index / (nFuzzy * nFuzzy)) % nFuzzy * nInputs + 1];
        O2_shared[index] = O2_shared[index] * O1_shared[((index / nFuzzy) % nFuzzy) % nFuzzy * nInputs + 2];
        O2_shared[index] = O2_shared[index] * O1_shared[((index % nFuzzy) % nFuzzy) % nFuzzy * nInputs + 3];

        O2[index] = O2_shared[index];
        // O2[index] = O1_shared[index / (nFuzzy * nFuzzy * nFuzzy)*nInputs];
    }
    __syncthreads();

    // Layer 3: Normalization
    double sumW = 0.0;
    if (index < nRules) {
        for (int i = 0; i < nRules; i++) {
            sumW = sumW + O2_shared[i];
        }
        if (sqrt(sumW * sumW) < dev_tol) {
            sumW = dev_tol;
        }
        O3_shared[index] = O2_shared[index] / sumW;

        O3[index] = O3_shared[index];
    }
    __syncthreads();

    // Layer 4: Defuzzyfication
    if (index < nRules) {
        aOut_shared[index] = aOut[index];
        cOut_shared[index] = cOut[index];
        fi_shared[index] = cOut_shared[index] - aOut_shared[index] * log((1.0 / O2_shared[index]) - 1.0);
        O4_shared[index] = O3_shared[index] * fi_shared[index];
        O4[index] = O4_shared[index];
    }
```

```
    __syncthreads();

    // Layer 5: Output
    double O5_reg = 0.0;
    if (index < nRules) {
        for (int i = 0; i < nRules; i++) {
            O5_reg = O5_reg + O4_shared[i];
        }
        O5_shared = O5_reg;
        O5[0] = O5_shared;
    }
    __syncthreads();

}
```

First, shared memory is instantiated and a thread `index` is created. After this, parallel computations for each layer are performed as shown in the code above and shared memory for each node in every layer is copied into global memory for its extraction to the *host*.

3.2.2 Speed results

To measure time variables, the C++ library "`time.h`" was used. Figure 3.27 shows the results for the C++ implementation, while 3.28 the CUDA implementation.,

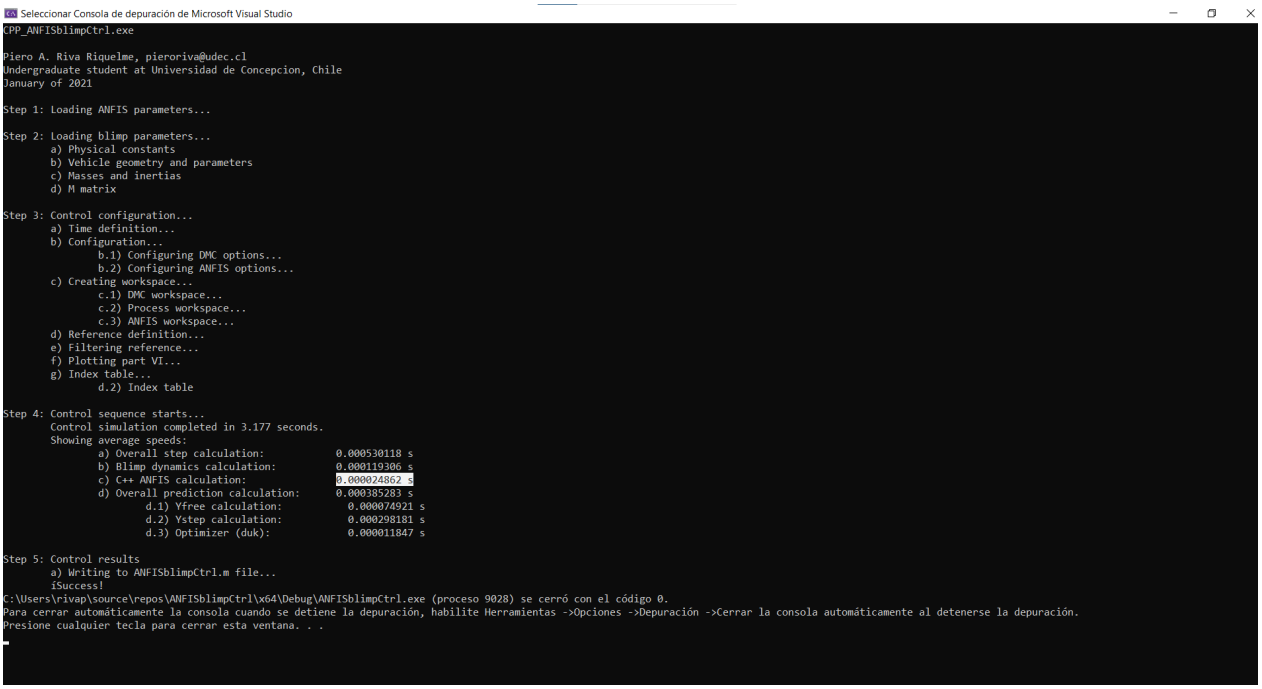


Figure 3.27: Average 625 rules ANFIS C++ speed of 0.000024862 seconds.

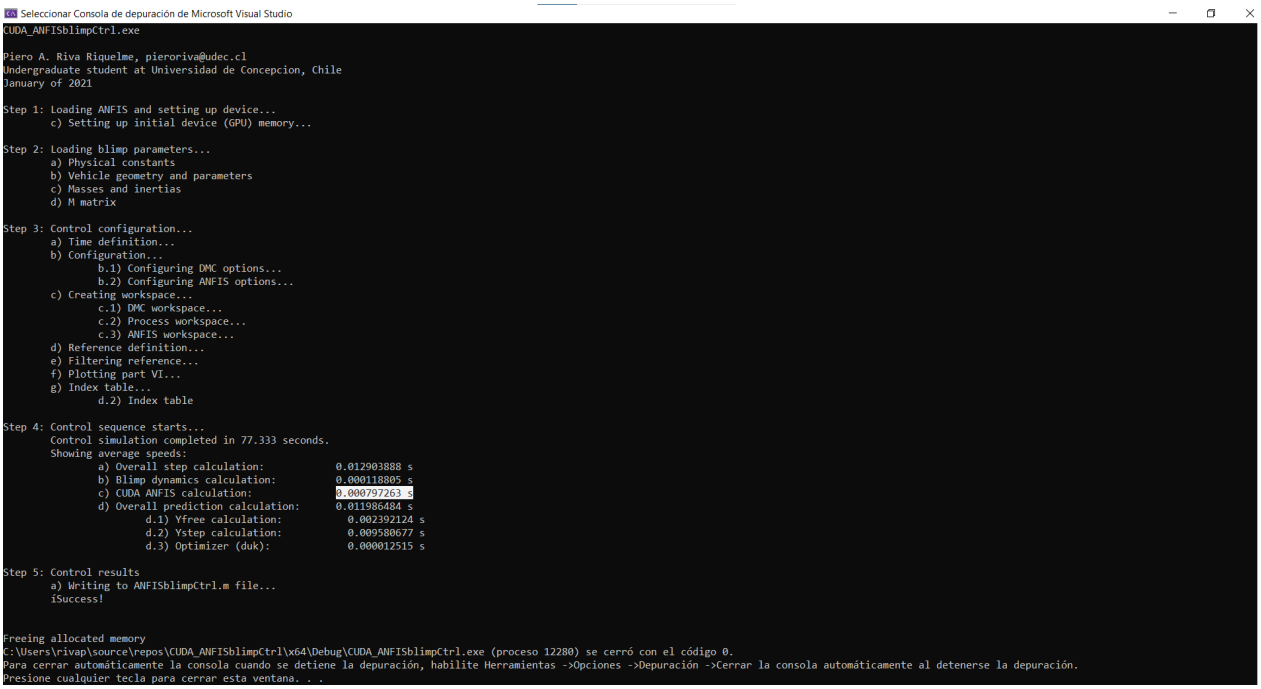


Figure 3.28: Average 625 rules ANFIS CUDA speed of 0.000797263 seconds.

Results obtained for the CUDA implementation show that the CPU computes faster the 625 rules ANFIS network than the GPU,

Device	625 rule ANFIS computation time
CPU	24.962μs
GPU	797.263μs
$\lvert t_{GPU} - t_{CPU} \rvert$	772.301μs
t_{GPU}/t_{CPU}	31.939

Table 3.10: ANFIS computation speeds for the CPU and GPU

It is important to note that these results were obtained for an AMD Ryzen 5 3550H processor (2.1 - 3.7 GHz) which is a lot to consider when compared to, for example, the Arduino Uno's ATmega328p microcontroller (20 MHz) or the NVIDIA's Jetson Nano with an ARM A57 CPU (1.43GHz) or the Allen Bradley's Micro820 PLC that "takes $0.3\mu s$ per basic instruction" [23].

Results shown in table 3.10 indicate that for a 625 rule ANFIS, the GPU and parallel programming control is not the suitable choice for the calculation of ANFIS.

CHAPTER 4

Discussion and conclusions

The aim of this work was to design and code a control strategy employing neural networks, they were of great interest due to their powerful pattern recognition ability and raised an ambition to use them in control systems. The focus was to control a complex system as to solve complex problems with complex solutions, so a non minimum phase, nonlinear MIMO system was used as the test subject.

ANFIS showed to be a captivating structure due to the adaptation of fuzzy membership functions under the deterministic gradient descent training and the "rule of thumb" pattern recognition approach. Its NARX model training implementation was coded and showed positive results only when training data was carefully selected (or filtered).

The blimp model appeared to be a good choice for control purposes but because of its complexity a great number of rules needed to be used to have a satisfactory NARX model. If ANFIS were to be used for linear or SISO systems, the control strategy could obtain better results for the same amount of resources (rules) used, or in other words, less resources could be employed to obtain the same results.

Results show that the control strategy works very well when the network is effectively trained and the controller is fine tuned under heuristic procedures, accomplishing the main goal of this work. Then, with the intention to extend the limitations of this control strategy, a GPU accelerated algorithm was designed and coded.

The parallel programming algorithm shows encouraging uses if a more complex system like biological, social (like pandemic), economic, environmental or robotic were to be control subjects, as more resources (thousands or millions of rules) would be needed to attain more efficient control results.

As any other coding work results were not obtained directly, but under an iterative process of trial and error and redesign to accomplish the main goal. This work intends to be a reference to any control engineer with the same interest in NN that can be used as support for future development.

Future directions of this work may point to code a faster CUDA C++ ANFIS training, bigger structures (millions of rules) or even different membership functions (like cubic splines) for better pattern recognition to closer NARX models and therefore superior control results.

Bibliography

[1] **H. Lamb**, "The inertia coefficients of an ellipsoid moving in fluid," *Advisory committee for aeronautics reports and memoranda, No. 623*, 1918.

[2] **L. B. Tuckerman**, "Inertia factors of ellipsoids for use in airship design," *National advisory committee for aeronautics, Report No. 210*, 1925.

[3] **S. P. Jones, J. D. DeLaurier**, "Aerodynamic estimation techniques for aerostats and airships," *J. aircraft, Vol 20, No 2 pp 120-126*, 1983.

[4] **S. B. V. Gomes**, "An investigation of the flight dynamics of airships with application to the YEZ-2A," *PhD. thesis, College of aeronautics, Cranfield University*, 1990.

[5] **J. R. Jang**, "Fuzzy controller design without domain experts," *[1992 Proceedings] IEEE International Conference on Fuzzy Systems,*, San Diego, CA, USA, 1992, pp. 289-296.

[6] **K. J. Hunt, D. Sbarbaro, R. Zbikowski, P.J. Gawthrop**, "Neural networks for control systems - a survey," *Automatica, Vol 28, Issue 6*, 1992, pp 1083-1112.

[7] **F. Gomide, A. Rocha, P. Albertos**, "Neurofuzzy Controllers," *IFAC Proceedings Volumes, Vol 25, Issue 25*, 1992, pp 13-26.

[8] **J. R. Jang**, "ANFIS: adaptive-network-based fuzzy inference system," *in IEEE Transactions on Systems, Man , and Cybernetics, Vol. 23, No. 3, pp. 665-685*, May-June 1993.

[9] **D. J. Kelly, P. D. Burton, M. A. Rahman**, "The application of aneural-fuzzy logic controller to process control," *NAFIPS/IFIS/NASA '94. Proceedings of the First International Joint Conference of The North American Fuzzy Information Processing Society Biannual Conference. The Industrial Fuzzy Control and Intellige*, San Antonio, TX, USA, 1994.

[10] **J. R. Jang**, "Inverse learning for fuzzy controller design," *Proceedings of International Conference on Control Applications*, Albany, NY, USA, 1995, pp.335-340.

[11] **J. R. Jang, CT. Sun**, "Neuro-fuzzy modeling and control," *in Proceedings of the IEEE, Vol. 83, No. 3, pp. 378-406,*, March 1995.

[12] **J. R. Jang**, "Input selection for ANFIS learning," *Proceedings of IEEE 5th International Fuzzy Systems, Vol. 2, pp. 1493-1499*, New Orleans, La, USA, 1996.

[13] **G. S. Sandhu, K. S. Rattan**, "Design of a neuro-fuzzy controller," *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Vol. 4, pp. 3170-3175*, Orlando, FL, USA, 1997.

[14] **S. B. V. Gomes, J. G. Ramos**, "Airship dynamic modeling for autonomous operation," *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), Vol. 4, pp. 3462-3467*, Leuven, Belgium, 1998.

[15] **Mueller, Joseph & Paluszek, Michael & Zhao, Yiyuan.**, "Development of an aerodynamic model and control law design for a high altitude airship," *presented at. AIAA Unmanned Unlimited Conference.*, 10.2514/6.2004-6479.

[16] **M. T. Frye, S. M. Gammon, C. Qian**, "The 6-DOF Dynamic Model and Simulation of the Tri-turbofan Remote-Controlled Airship," *2007 American Control Conference, pp. 816-821*, New York, NY, 2007.

[17] **R. Mellah, S. Guermah, R. Toumi**, "Adaptive control of bilateral teleoperation system with compensatory neural-fuzzy controllers," *Int. J. Control Autom. Syst. 15, pp. 1949 1959*, 2017.

[18] **B. L. Stevens, F. L. Lewis, E. N. Johnson**, "Aircraft control and simulation: dynamics, control design, and autonomous systems, erd edition," *Wiley and sons*, 2015.

[19] **H. Khati, R. Mellah, H. Talem**, "Neuro-fuzzy Control of aPosition-Position Teleoperation System Using FPGA," *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Miedzyzdroje, Poland, 2019.

[20] **E.F. Camacho, C. Bordons**, "Model Predictive Control 2nd edition," *Springer*, 2007.

[21] **E. Ita Essien**, "Adaptive NeuroFuzzy Inference System (ANFIS) Based Model Predictive Control (MPC) for Carbon Dioxide Reforming Of Methane (CDRM) In A Plug Flow Tubular Reactor For Hydrogen Production," *A Thesis submitted to the Faculty of Graduate Studies and Research In Partial Fulfillment of the Requirements for the Degree of Master of Applied Science in Industrial Systems Engineering*, University of Regina, Canada, 2012.

[22] **NVIDIA**, "CUDA C++ programming guide: Design guide," *PG-02829-001_v11.1*, October 2020.

[23] **Allen Bradley**, "Micro800 programmable controller family," *2080-SG001D-EN-P*, December 2013.

APPENDIX A

GeForce GTX 1650 GPU

Specifications of the NVIDIA GTX 1650 GPU are shown below,

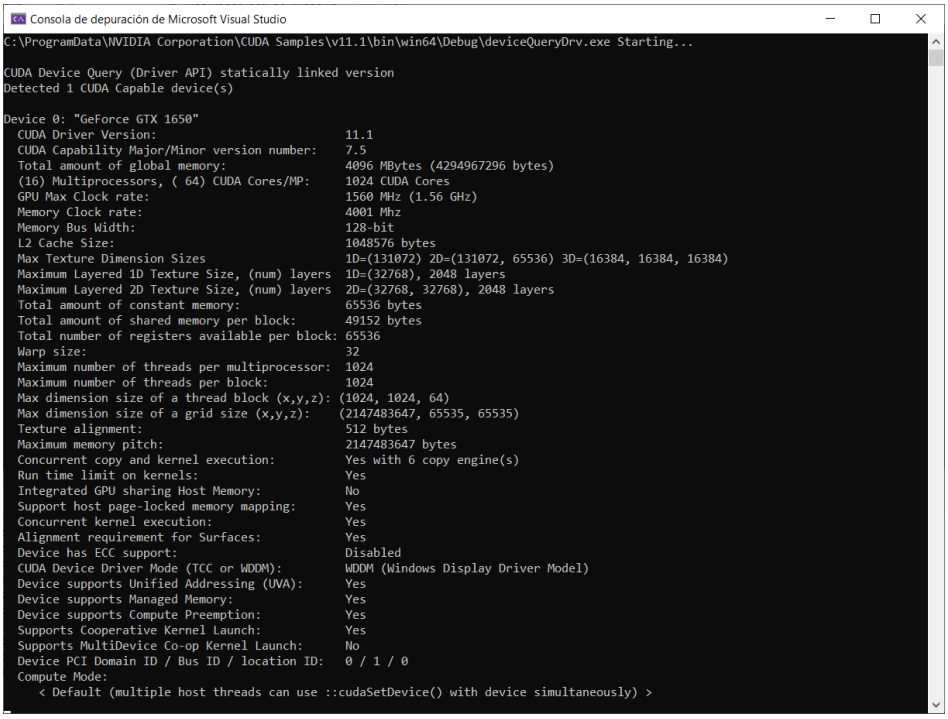


Figure A.1: CUDA specifications of the NVIDIA GeForce GTX 1650 GPU.

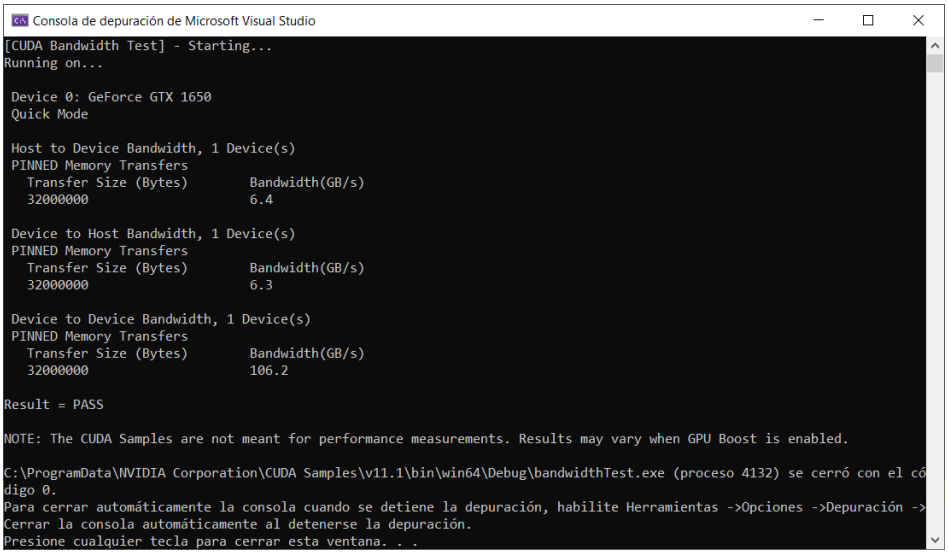


Figure A.2: CUDA device bandwidth.

Table A.1 shows remarkable characteristics,

Item	Description
Total amount of global memory	4096 MB
Total amount of constant memory	65536 Bytes
Total amount of shared memory per block	49152 Bytes
Total number of registers available per block	65536
CUDA cores	1024
Max dimension size of a thread block (x,y,z)	(1024, 1024, 64)
Max dimension size of a grid size (x,y,z)	(2147483647, 65535, 65535)
Host to Device bandwidth	6.4 GB/s
Device to Host bandwidth	6.3 GB/s

Table A.1: Specifications of the NVIDIA GTX 1650 GPU