الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière Électronique
Spécialité électronique des systèmes embarqués

Présenté par

Khemici Houssam Eddine

&

Benkhira Ahmed Imad Eddine

Estimation neuronale de la vitesse d'un moteur asynchrone embarquée sur microcontrôleur

Proposé par : **Mr. FERDJOUNI Abdellaziz.**

Année Universitaire 2024-2025

Remerciements

Qu’il nous soit d’abord permis de remercier et d’exprimer notre gratitude envers **Allah**, qui nous a donné la patience et la volonté pour que nous puissions achever ce travail.

Nous tenons à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce mémoire.

Nous remercions tout particulièrement **Mr Abdellaziz FERDJOUNI** notre encadrant, pour sa disponibilité, ses conseils précieux, son soutien et ses encouragements tout au long de ce travail. Son expertise et son accompagnement ont été d’une grande aide.

Nous remercions également les membres du jury qui ont accepté d’évaluer ce travail.

Un grand merci à nos collègues et amis pour leur soutien moral, leurs échanges enrichissants et leur bienveillance tout au long de cette aventure.

Enfin, Nous tenons à remercier nos familles pour leur amour, patience et soutien inconditionnel tout au long de notre parcours.

Dédicaces

Je dédie ce mémoire :

À **mes très chers parents**, pour leur soutien indéfectible tout au long de mon parcours scolaire, et grâce à qui j'ai pu réussir dans mes études.

A **mes chers frères.**

A **tous ma famille.**

A **tous mes amis** pour leur aide, leur amitié et leurs encouragements sans faille.

A **tous les enseignants** pour leur accompagnement**.**

A **toute personne** ayant contribué à ce travail de près ou de loin.

ملخص: يعرض هذا البحث طريقة لتقدير سرعة الدوران لمحرك غير متزامن ثلاثي الأطوار باستخدام الشبكات العصبية الاصطناعية. تم بناء النموذج اعتمادًا على بيانات تم الحصول عليها من محاكاة باستخدام برنامج MATLAB/Simulink ، ثم تم تدريبه لمحاكاة سلوك الآلة. بعد ذلك، تم تصميم تطبيق مدمج على متحكم دقيق من أجل تقييم أداء النظام في ظروف التشغيل الحقيقية. وتُظهر النتائج قابلية تطبيق هذه الطريقة وفعاليتها في التقدير بدون حساسات وضمن بيئة مدمجة.

كلمات المفاتيح: المحرك غير المتزامن، تقدير السرعة، الشبكات العصبية، المحاكاة، المتحكم الدقيق، النظام المدمج.

Résumé : Ce mémoire expose une méthode d'estimation de la vitesse rotorique d'une machine asynchrone triphasée en utilisant un réseau de neurones artificiels. Le modèle a été élaboré sur la base de données issue d'une simulation sous MATLAB/Simulink, avant d'être entraîné pour soumettre la machine au même comportement. Une application embarquée a été ensuite conçue sur microcontrôleur pour évaluer les performances du système en conditions réelles. Les résultats montrent la faisabilité et l'efficacité de ce type de méthode d'estimation sans capteur et en environnement embarqué.

Mots clés : Machine asynchrone, réseau de neurones, estimation de vitesse, simulation, implémentation embarquée.

Abstract :

This thesis presents a method for estimating the rotor speed of a three-phase induction motor using artificial neural networks. The model was developed based on data obtained from a MATLAB/Simulink simulation, and trained to reproduce the motor's behavior. An embedded application was then implemented on a microcontroller to evaluate system performance under real-world conditions. The results demonstrate the feasibility and effectiveness of this type of sensorless estimation method in embedded environments.

Keywords: Induction motor, speed estimation, neural network, simulation, microcontroller, embedded system.

Listes des acronymes et abréviations

MAS : Machine asynchrone.

f : fréquence d'alimentation.

P : Nombre de pair de pôles.

n_s : Vitesse de rotation.

g : glissement.

a, b, c : Indices correspondant aux trois phases (a), (b), (c).

s, r : Indices correspondant aux grandeurs statoriques et rotoriques.

d, q : Indices correspondant au référentiel de Park.

$[V_{abcs}][V_{abcr}]$: Vecteur tensions appliquées aux phases statoriques et rotoriques.

$[I_{abcs}][I_{abcr}]$: Vecteur courants appliquées aux phases statoriques et rotoriques.

$[Q_{abcs}][Q_{abcr}]$: Vecteur flux statoriques et rotoriques.

R_s, R_r : Résistance d'une phase statorique et rotorique.

$[M_{s\sigma}][M_{r\sigma}]$: Inductances mutuelles entre deux phases statoriques ou rotoriques.

L_s et L_r : Inductances propres statoriques et rotoriques.

l_s et l_r : Inductances mutuelles entre deux phases statoriques ou rotoriques.

l_{sr} : Inductance mutuelle maximale entre une phase statorique et rotorique.

θ_r : L'angle entre l'axe ar et d.

θ_s : L'angle entre l'axe as et d.

[P] : La matrice de transformation de Park.

$[V_{dq}]$: Vecteur tensions après la transformation de Park.

$[I_{dq}]$: Vecteur courants après la transformation de Park.

$[Q_{dq}]$: Vecteur flux après la transformation de Park.

w_s : La pulsation statorique.

w_r : La pulsation rotorique.

C_{em} : Couple électromagnétique.

C_r : Couple de charge.

Ω: Vitesse mécanique.

J : Moment d'inertie.

f_v : Coefficient de frottement.

σ : Coefficient de fuite.

T_r : Constante de temps rotorique.

T_s : Constante de temps statorique.

$x(t)$ **:** Vecteur d'état.

$u(t)$ **:** Vecteur de commande.

A : Matrice d'état.

B : Matrice d'application de la commande.

MLI : Modulation de largeur d'impulsion.

MSE : Mean Squared Error (Erreur quadratique moyenne).

R : Coefficient de régression.

MLP : Multi-Layer Perceptron (Perceptron multicouche).

RNA : Réseau de Neurones Artificiels.

ANN : Artificial Neural Network.

ESP : Espressif Systems Platform (ESP8266, ESP32, etc.).

ADC : Analog to Digital Converter.

PWM : Pulse Width Modulation.

IDE : Integrated Development Environment.

nnstart : Interface MATLAB pour l'entraînement des réseaux de neurones.

TrainIm : Levenberg-Marquardt Training Function.

Te : Temps d’échantillonnage.

Table des matières

INTRODUCTION GENERALE 1

CHAPITRE 1 LA MACHINE ASYNCHRONE TRIPHASEE 3

1.1 Introduction 3

1.2 La machine asynchrone 3
1.2.1 Définition 3
1.2.2 Constitution de la machine asynchrone 4
1.2.3 Principe de fonctionnement 10

1.3 Conclusion 12

CHAPITRE 2 MODELISATION MATHEMATIQUE ET SIMULATION DE LA MACHINE ASYNCHRONE 13

2.1 Introduction 13

2.2 Les Hypothèses Simplificatrices 14

2.3 Mise en équations de la MAS 15
2.3.1 Equations électriques 15
2.3.2 Equations magnétiques 16
2.3.3 Equations mécaniques 18

2.4 Transformation de Park 18
2.4.1 Equations électriques 20
2.4.2 Equations magnétiques 20
2.4.3 Equations mécaniques 21

2.5 Représentation d'état du modèle de la MAS 21

2.6 Représentation Mise en équation d'état 22

2.7 Mise Bloc de simulation 24

2.8 Résultats de simulation 25

2.9 Conclusion 29

CHAPITRE 3 ESTIMATION DE LA VITESSE PAR RESEAU DE NEURONES ARTIFICIELS 30

3.1 Introduction 30

3.2 Historique de réseaux de neurones 30

3.3 Réseaux de neurones artificiels 31
3.3.1 Le neurone formel 31
3.3.2 La fonction d'activation ou de seuillage 33

3.4 Architecture des réseaux de neurone artificiel 33
3.4.1 Réseau Multicouches 34

3.5 Apprentissage des réseaux de neurone 35
3.5.1 Mode supervisé 35
3.5.2 Mode non supervisé 35

3.6 Le perceptron multicouche 35
3.6.1 Architecture 36
3.6.2 L'apprentissage des réseaux MLP 36
3.6.3 Rétropropagation de l'erreur (Back propagation) 37

3.7 Préparation des données d'entraînement 39
3.7.1 Caractéristiques des deux cas de données pour l'apprentissage 40

3.8 Entrainement des réseaux de neurones 40
3.8.1 Architecture du réseau 40
3.8.2 Les étapes d'entrainement avec nnstart : 41

3.9 Conclusion 49

CHAPITRE 4SIMULATION EN CONDITION REELLE ET IMPLEMENTATION EMBARQUEE (TEMPS REEL) 50
4.2.1 Matériel utilise 51
4.2.2 Résultat obtenu 52

4.3 Implémentation embarquée 55
4.3.1 Principes d'implémentation des réseaux neuronaux en environnement embarqué 55
4.3.2 Objectif de l'implémentation 56
4.3.3 Cible matérielle utilisée 57
4.3.4 Intégration du modèle neuronal 58
4.3.5 Avantages de cette architecture embarquée 64
4.3.6 Évaluation des performances d'exécution 64

4.4 Conclusion 68

CHAPITRE 5RESULTATS ET COMPARAISON. 69

5.1 Introduction 69

5.2 Résultats et comparaison 69

5.2.1 Estimation du premier modèle par rapport à la vitesse simulée en MATLAB 70

5.2.2 Estimation du deuxième modèle par rapport à la vitesse simulée en MATLAB 72

5.2.3 Estimation du premier modèle par rapport à la vitesse mesurée 74

5.2.4 Estimation du deuxième modèle par rapport à la vitesse mesurée..... 74

5.2.5 Test du modèle embarqué avec des données réelles 75

5.2.6 Résultat de la comparaison entre le modèle embarqué et l'architecture MATLAB 78

5.2.7 Résultat embarqué comparé à la vitesse simulée dans ControlDesk 78

5.3 Conclusion 79

CONCLUSION GENERALE 80

ANNEXES 81

BIBLIOGRAPHIE 87

Liste des figures

FIGURE 1.1 : ELEMENTS DE CONSTITUTION D'UNE MACHINE ASYNCHRONE A CAGE D'ECUREUIL **[5]**. 4
FIGURE 1.2 : VUE SCHEMATIQUE EN PERSPECTIVE DU STATOR (CIRCUIT MAGNETIQUE, CONDUCTEURS D'ENCOCHES, TETE DE BOBINES). (ENROULEMENT STATORIQUE D'UNE MACHINE A 4 POLES) [6]. 6
FIGURE 1.3 : PHOTO DU STATOR D'UNE MACHINE ASYNCHRONE [3]. 6
FIGURE 1.4 : VUE D'UN ROTOR A CAGE [2]. 7
FIGURE 1.5 : VUE D'UN ROTOR BOBINE [2]. 8
FIGURE 1.6 : SCHEMA DESIGNANT LES ELEMENTS CONSTITUANT UNE MACHINE ASYNCHRONE [4]. 10
FIGURE.1.7 : PRINCIPE DE FONCTIONNEMENT D'UN MOTEUR ASYNCHRONE [8]. 11
FIGURE 2.1 : REPRESENTATION SCHEMATIQUE D'UNE MACHINE ASYNCHRONE (MAS). 15
FIGURE 2.2 : REPRESENTATION DE LA MACHINE ASYNCHRONE TRIPHASEE ET SA MACHINE BIPHASEE. 21
FIGURE 2.3 : SCHEMA BLOC DE LA MODELISATION DYNAMIQUE DE LA MAS DANS LE REPERE DQ. 24
FIGURE 2.4 : CODE MATLAB. 25
FIGURE 2.5 : TENSION STATORIQUE DANS LE REPÈRE (D,Q). 26
FIGURE 2.6 : RÉPONSE DU COUPLE DE CHARGE À UNE PERTURBATION À T = 0.5 S. 26
FIGURE 2.7 : COURANT STATORIQUE ISD ISQ. 27
FIGURE 2.8 : COURANT STATORIQUE I_{SABC}. 27
FIGURE 2.9 : VITESSE ROTORIQUE (WM). 28
FIGURE 3.1 : SCHEMA D'UN NEURONE FORMEL. 32
FIGURE 3.2 : LES FONCTIONS D'ACTIVATION LES PLUS UTILISEES. 33

FIGURE 3.3 : RESEAU MULTICOUCHE [25]. 34
FIGURE 3.4 : ARCHITECTURE DU PERCEPTRON MODEL MLP. 36
FIGURE 3.5 : PREPARER LES DONNEES D'ENTRAINEMENT. 41
FIGURE 3.6 : VALIDATION ET DONNEES D'ESSAI. 42
FIGURE 3.7 : ARCHITECTURE DU RESEAU NEURONE. 42
FIGURE 3.8 : ALGORITHME D'APPRENTISSAGE. 42
FIGURE 3.9 : RESEAU DE NEURONE. 43
FIGURE 3.10 : ARCHITECTURE DU RESEAU DE NEURONE CAS 1. 44
FIGURE 3.11 PERFORMANCE DE 1^{ER} CAS. 45
FIGURE 3.12 : REGRESSION DE 1^{ER} CAS. 46
FIGURE 3.13 ARCHITECTURE DU RESEAU DE NEURONE CAS 2. 47
FIGURE 3.14 PERFORMANCE DE 2^{EME} CAS. 48
FIGURE 3.15 : REGRESSION DE 2^{EME} CAS 48
FIGURE 4.1 : PHOTO DE LA MACHINE ASYNCHRONE. 51
FIGURE 4.2 : TENSION D'ALIMENTATION TRIPHASEE. 52
FIGURE 4.3 : CAPTEUR DE COURANTS ET DE TENSIONS. 52
FIGURE 4.4 : VUE DE L'INTERFACE CONTROLDESK. 53
FIGURE 4.5 : COURANT STATORIQUE I_{A1} I_{B1} I_{C1} 53
FIGURE 4.6 : COURANT TRANSFORMEE I_{SD} I_{SQ} 54
FIGURE 4.7 : TENSION D'ALIMENTATION V_{SD} V_{SQ}. 54
FIGURE 4.8 : VITESSE ROTORIQUE ω_M. 55
FIGURE 4.9 : MATLAB GETFUNCTION SYNTAXE 58
FIGURE 4.10 : EXEMPLE DE CONTENU DE SCRIPT GENERE AVEC MATLAB GETFUNCTION. 59
FIGURE 4.11 : LA BOUCLE DE LA NORMALISATION. 60
FIGURE 4.12 : LA BOUCLE DE COUCHE CACHEE. 60
FIGURE 4.13 : LA FONCTION EXPONENTIELLE. 61
FIGURE 4.14 : LA COUCHE DE SORTIE. 63
FIGURE 4.15: LA PARTIE DE LA DENORMALISATION. 63
FIGURE 4.16: LES DONNEES DE TEST. 65
FIGURE 4.17 : MESURE DU TEMPS D'EXECUTION. 66
FIGURE 4.18 : LES RESULTATS IMPRIMES SUR LE MONITEUR SERIE. 67
FIGURE 5.1 : LE COUPLE DE CHARGE UTILISE DANS LA SIMULATION. 70
FIGURE 5.2 : L'ARCHITECTURE DE PREMIER RESEAU. 70
FIGURE 5.3 : REPONSE EN REGIME PERMANENT (CAS 1 – SIMULATION MATLAB). 71
FIGURE 5.4 : REPONSE EN REGIME TRANSITOIRE (CAS 1 – SIMULATION MATLAB). 71
FIGURE 5.5 : L'ARCHITECTURE DE DEUXIEME RESEAU. 72
FIGURE 5.6 : REPONSE EN REGIME PERMANENT (CAS 2 – SIMULATION MATLAB). 72
FIGURE 5.7 : RÉPONSE EN RÉGIME TRANSITOIRE (CAS 2 – SIMULATION MATLAB). 73

FIGURE 5.8 : ESTIMATION DE LA VITESSE CAS 1COMPAREE A LA VITESSE REELLE. 74
FIGURE 5.9 : ESTIMATION DE LA VITESSE CAS 2 COMPAREE A LA VITESSE REELLE. 74
FIGURE 5.10 : CODE MATLAB POUR LA COMMUNICATION SÉRIE. 76
FIGURE 5.12 : COMPARAISON ENTRE ESTIMATION EMBARQUEE ET ESTIMATION MATLAB CAS1. 78
FIGURE 5.13 : COMPARAISON ENTRE ESTIMATION EMBARQUÉE ET VITESSE RÉEL. 78

Liste des tableaux

TABLEAU 3.1 : CARACTERISTIQUES DES DEUX CAS DE DONNEES POUR L'APPRENTISSAGE 40

TABLEAU 3.2 : PARAMETRES DES RESEAUX MLP UTILISES POUR LES DEUX CAS. 41
TABLEAU 3.3 : COMPARAISON DES PERFORMANCES SELON LE NOMBRE DE NEURONES DANS LA COUCHE CACHEE CAS 1..... 44
TABLEAU 3.4 : COMPARAISON DES PERFORMANCES SELON LE NOMBRE DE NEURONES DANS LA COUCHE CACHEE CAS 2..... 47
TABLEAU 3.5 : COMPARAISON DES PERFORMANCES ENTRE LES DEUX CAS. 49
TABLEAU 4.1 : CARACTERISTIQUES TECHNIQUES DU MICROCONTROLEUR ESP8266..... 57

Introduction générale

Le développement des systèmes industriels modernes impose des exigences de plus en plus élevées en matière de performance, de fiabilité et de maintenance prédictive. Dans ce contexte, les moteurs asynchrones triphasés occupent une place importante dans l'industrie, notamment en raison de leur robustesse, de leur faible coût et de leur facilité d'entretien [19]. Cependant, le contrôle sans capteur, en particulier l'estimation de la vitesse du rotor, reste un défi majeur pour le développement de systèmes embarqués efficaces [33].

Traditionnellement, la mesure de la vitesse nécessite l'utilisation de capteurs mécaniques (tachymètres, encodeurs), qui augmentent le coût du système, la taille, la maintenance et la vulnérabilité aux défauts. Une alternative moderne et efficace consiste à utiliser des réseaux de neurones artificiels, capables de modéliser un comportement non linéaire à partir de données expérimentales pour estimer des quantités qui ne sont pas directement mesurées [34] [29].

Ce mémoire s'inscrit dans cette perspective. Il propose une méthode d'estimation de la vitesse d'une machine asynchrone à l'aide d'un réseau neuronal de type MLP (perceptron multicouche), basé uniquement sur des grandeurs mesurables telles que les courants et tensions statoriques [34].

Le travail s'est articulé autour de plusieurs étapes :

Une modélisation mathématique de la machine asynchrone a d'abord été réalisée, suivie d'une simulation sous MATLAB/Simulink.

Les données simulées ont ensuite servi à l'entraînement de deux réseaux neuronaux basés sur différentes entrées : courants triphasés et composantes dq [29].

Après évaluation des performances (erreur, régression, cohérence), le modèle le plus fiable a été implémenté sur un microcontrôleur ESP8266 pour une estimation embarquée.

Une comparaison finale avec les résultats simulés et la vitesse réelle a permis de valider la solution proposée.

Ce travail démontre qu'une estimation efficace de la vitesse peut être obtenue en utilisant l'intelligence artificielle sans capteurs et que cette solution est adaptée aux systèmes embarqués légers et peu coûteux.

Chapitre 1 La Machine Asynchrone Triphasée

1.1 Introduction

Les moteurs électriques jouent un rôle crucial dans les systèmes mécaniques avancés, en particulier dans la conversion de l'énergie électrique en énergie mécanique. Parmi ces machines, les moteurs asynchrones sont largement utilisés en raison de leur robustesse, de leur construction simple et de leur faible coût. Ils sont très répandus dans les installations industrielles, domestiques et agricoles.

Dans ce chapitre, nous présentons la machine asynchrone triphasée, élément central de ce mémoire. Nous décrivons ses principaux constituants tels que le stator, le rotor, puis nous expliquons son principe de fonctionnement basé sur l'induction électromagnétique. Enfin, nous abordons des notions fondamentales comme la vitesse de synchronisme, le glissement, ainsi que le bilan de puissance, avant de conclure sur les avantages et inconvénients de cette machine.

1.2 La machine asynchrone

1.2.1 Définition

Une machine asynchrone est une machine à courant alternatif dont la vitesse du rotor et la vitesse du champ magnétique tournant ne sont pas égales. Le rotor est toujours en retard par rapport à la vitesse du champ statorique. La machine asynchrone est appelée machine à induction car l'énergie transférée du stator vers le rotor, ou inversement, se fait par induction électromagnétique **[1]**.

1.2.2 Constitution de la machine asynchrone

a Moteur Asynchrone (MAS)

La machine asynchrone, est composée d'un stator et d'un rotor réalisés en tôles d'acier au silicium, comportant des encoches dans lesquelles sont placés les enroulements. Le stator est fixe ; il contient les enroulements reliés à la source d'alimentation. Le rotor, quant à lui, est monté sur un arbre tournant. Selon que les enroulements du rotor sont accessibles de l'extérieur ou définitivement fermés sur eux-mêmes, on distingue deux types de rotors : le rotor bobiné et le rotor en cage d'écureuil. Toutefois, on admet que leur structure est électriquement équivalente à celle d'un rotor bobiné dont les enroulements sont en court-circuit. Dans ce travail, nous nous intéressons à la machine asynchrone à cage d'écureuil.

Les éléments de constitution d'une machine asynchrone à cage d'écureuil sont illustrés dans la Figure 1.1 **[3].**

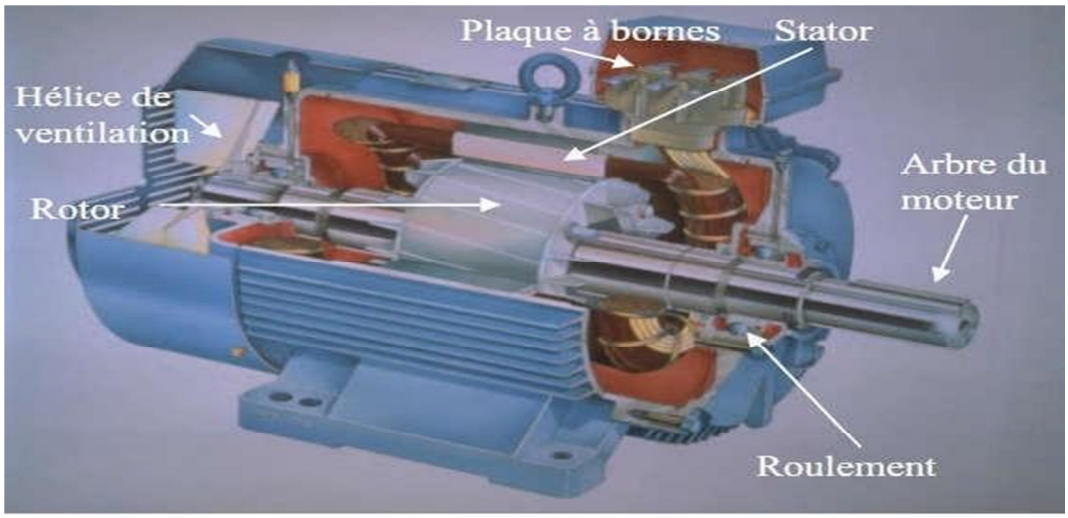

Figure 1.1 : Eléments de constitution d'une machine asynchrone à cage d'écureuil **[5]**.

La machine asynchrone à cage est composée de :

- Le stator, partie fixe de la machine où est connectée l'alimentation électrique.
- Le rotor, partie tournante qui permet de mettre en rotation la charge mécanique.

Les organes mécaniques permettant la rotation du rotor et le maintien des différents sous-ensembles **[2]**.

b Le stator

Le stator est constitué d'un enroulement triphasé bobiné, réparti dans les encoches d'un circuit magnétique. Ce dernier est formé par un empilement de fines tôles en acier, isolées entre elles pour limiter les pertes par courants de Foucault. Les tôles sont perforées d'encoches parallèles à l'axe de la machine, dans lesquelles sont logés les conducteurs de l'enroulement.

L'ensemble de ces tôles forme un anneau creux, monté à l'intérieur d'un bâti en fonte ou en acier moulé, qui assure la rigidité mécanique du stator. L'enroulement ainsi disposé permet de générer un champ magnétique tournant lorsqu'il est alimenté par un système triphasé [4].

Le stator d'un moteur asynchrone est constitué de trois enroulements couplés en étoile ou en triangle et décalés entre eux de **2π/3** [7].

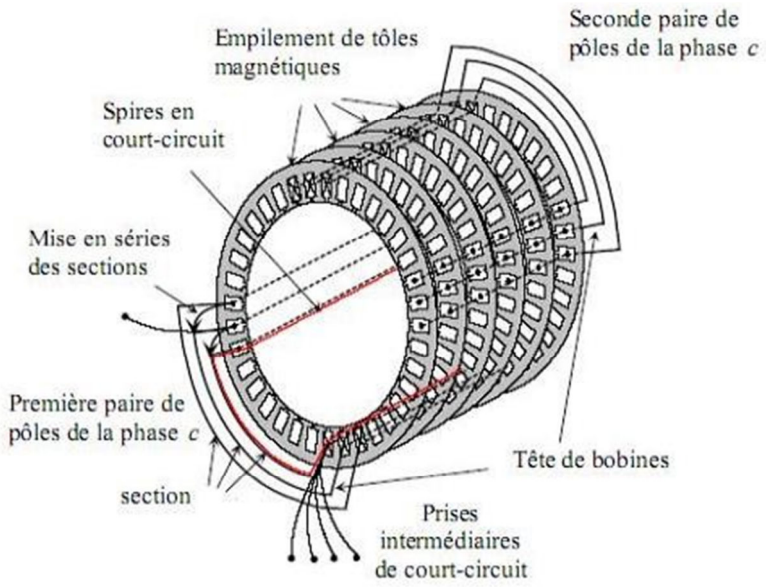

Figure 1.2 : Vue schématique en perspective du stator (circuit magnétique, conducteurs d'encoches, tête de bobines). (Enroulement statorique d'une machine a 4 pôles) [6].

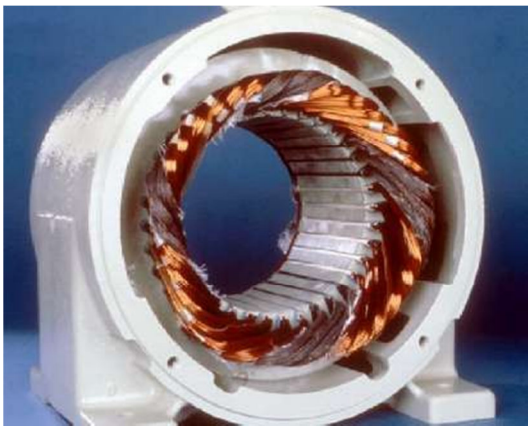

Figure 1.3 : Photo du stator d'une machine asynchrone [3].

c Le rotor

Le rotor, monté sur l'arbre du moteur, est constitué d'un noyau en fer feuilleté formé par un empilement de fines tôles métalliques. Des encoches sont usinées sur la

surface extérieure afin d'accueillir les barres conductrices. Il est séparé du stator par un entrefer très étroit, généralement compris entre 0,4 et 2 mm [4].

Il existe deux principaux types de rotors : le **rotor à cage d'écureuil et** le **rotor bobiné.**

- **Le rotor à cage d'écureuil**

  Le rotor à cage d'écureuil est constitué de barres conductrices, généralement en cuivre nu, insérées dans les encoches longitudinales du noyau rotorique. Ces barres sont reliées électriquement à leurs extrémités par deux anneaux conducteurs, formant ainsi un circuit en court-circuit. Cette disposition particulière donne au rotor son apparence caractéristique, semblable à une cage d'écureuil, d'où son nom.

  Dans les moteurs de petite et moyenne puissance, l'ensemble — barres et anneaux — est souvent réalisé en une seule pièce par coulée d'aluminium dans le noyau du rotor, ce qui simplifie la fabrication et réduit le coût de production.

Figure 1.4 : Vue d'un rotor à cage [2].

- **Le rotor à bobine**

  Le rotor bobiné comporte un enroulement triphasé, similaire à celui du stator, logé dans les encoches du noyau rotorique. Cet enroulement est constitué de

trois phases raccordées en étoile, dont les extrémités libres sont connectées à trois bagues collectrices fixées sur l'arbre du moteur.

Ces bagues permettent, à l'aide de balais, d'insérer des résistances externes en série avec chaque phase du rotor lors du démarrage. Cette configuration permet de limiter le courant de démarrage et d'augmenter le couple moteur. En régime permanent, les balais sont court-circuités, rendant les résistances inactives et ramenant le fonctionnement à celui d'un rotor à cage.

Figure 1.5 : Vue d'un rotor bobiné [2].

d L'entrefer

L'entrefer est la zone amagnétique très étroite située entre le stator et le rotor. Il constitue un vide (ou un espace rempli d'air) dont l'épaisseur est généralement de l'ordre du millimètre. En raison de sa faible taille, l'entrefer est très sensible aux irrégularités causées par la présence des encoches du stator, ce qui engendre des **harmoniques parasites**, appelées harmoniques d'encoches.

Pour atténuer ces effets indésirables, des **cales magnétiques** sont souvent placées pour fermer les encoches et maintenir en place les bobinages, réduisant ainsi les variations de perméabilité dans l'entrefer et améliorant les performances électromagnétiques de la machine [4].

e Les organes mécaniques

La carcasse constitue le support principal de la machine ; elle assure à la fois une fonction de maintien mécanique, une protection contre les agressions extérieures (poussières, humidité) et une dissipation thermique partielle.

L'arbre moteur est un organe de transmission mécanique. Il comporte une zone centrale qui soutient le rotor, et il est maintenu en position par un ou plusieurs paliers. Ces paliers assurent à la fois le guidage et la rotation libre de l'arbre. Le palier côté opposé à l'entraînement est généralement conçu comme un palier libre pour permettre les dilatations thermiques de l'arbre sans générer de contraintes mécaniques excessives.

Pour éviter la circulation de courants induits dans l'arbre — causés par des dissymétries du circuit magnétique ou des harmoniques — l'un des paliers est souvent isolé électriquement. Les paliers utilisés sont en général des roulements à billes ou à rouleaux [5].

Dans les moteurs de petite et moyenne puissance, un ventilateur de refroidissement est souvent intégré afin d'assurer une dissipation thermique efficace lors du fonctionnement [5].

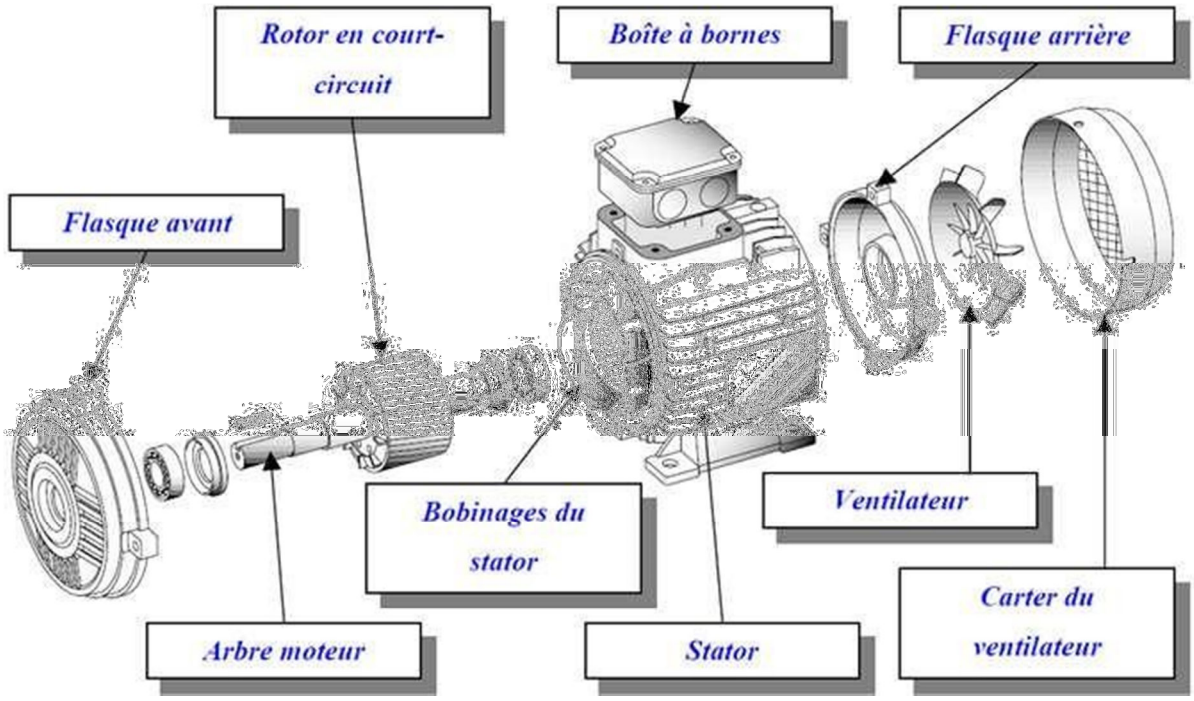

Figure 1.6 : Schéma désignant les éléments constituant une machine asynchrone [4].

1.2.3 Principe de fonctionnement

Le principe de fonctionnement de la machine asynchrone repose entièrement sur la loi de l'induction électromagnétique. Cette machine est assimilée à un transformateur à champ magnétique tournant, où le stator représente l'enroulement primaire et le rotor l'enroulement secondaire court-circuité.

Ce fonctionnement découle de l'interaction électromagnétique entre le champ tournant, créé par les courants triphasés alimentant le stator, et les courants induits dans les conducteurs du rotor, lesquels sont coupés par ce champ tournant. Lorsque le rotor tourne à une vitesse N_c différente de celle du synchronisme, l'application de la loi de Faraday à l'enroulement du rotor montre qu'une force électromotrice est induite. Cette force électromotrice, en raison du court-circuit des enroulements du rotor, génère un courant dont l'intensité est limitée par l'impédance des enroulements.

L'interaction entre ce courant et le champ magnétique tournant produit des forces électromagnétiques qui s'exercent sur les conducteurs du rotor. Le moment de ces forces par rapport à l'axe de rotation génère le couple de la machine, proportionnel à la vitesse de rotation du rotor. La vitesse de rotation du rotor est donnée par l'expression :

$$n_s = \frac{f}{p} \qquad (1.1)$$

Où : f : la fréquence d'alimentation.

p : représente le nombre de pair de pôles.

L'interaction électromagnétique des deux parties de la machine n'est possible que lorsque la vitesse du champ tournant (n_1) diffère de celle du rotor (n), c'est-à-dire lorsque $n \neq n_1$, car dans le cas contraire, ($n = n_1$), le champ serait immobile par rapporte au rotor et aucun courant ne serait induit dans l'enroulement rotorique.

Le rapport g et appelé glissement de la machine asynchrone [8].

$$g = \frac{ns - n}{ns} \qquad (1.2)$$

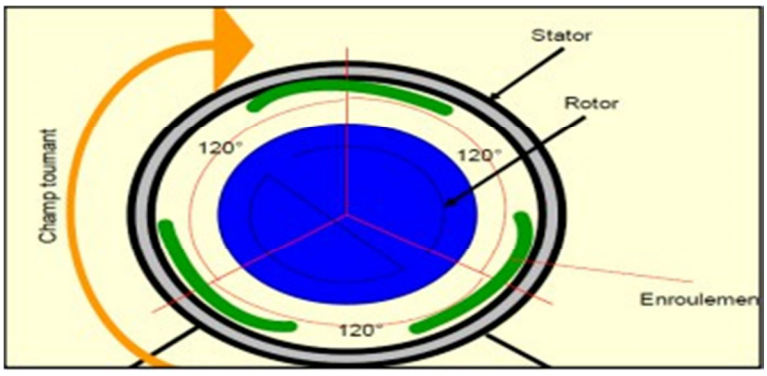

Figure.1.7 : principe de fonctionnement d'un moteur asynchrone [8].

1.3 Conclusion

Dans ce chapitre, nous avons présenté les généralités sur la machine asynchrone, en décrivant ses principaux constituants mécaniques et électromagnétiques, ainsi que son principe de fonctionnement basé sur l'induction électromagnétique et l'interaction entre le champ tournant statorique et les courants induits dans le rotor.

Cette étude préliminaire constitue une base essentielle pour aborder les chapitres suivants, en particulier le Chapitre 2, dans lequel nous développerons le modèle mathématique de la machine asynchrone triphasée et procéderons à sa modélisation sous MATLAB/Simulink.

Chapitre 2 Modélisation mathématique et simulation de la machine asynchrone

2.1 Introduction

Après avoir présenté la structure, le fonctionnement et les caractéristiques générales de la machine asynchrone dans le chapitre précédent, ce chapitre est consacré à sa modélisation mathématique et à sa simulation. La compréhension du comportement dynamique de la machine nécessite une représentation par équations électriques et mécaniques qui permettent de reproduire fidèlement son fonctionnement réel.

Dans un premier temps, nous établirons le modèle mathématique de la machine asynchrone triphasée, en utilisant les équations fondamentales dans le repère triphasé (abc), puis en les transformant dans le repère (dq) à l'aide de la transformation de Park, afin de simplifier l'analyse et la commande. La partie mécanique du modèle sera également abordée, incluant le couple électromagnétique et les principales pertes mécaniques.

Dans un second temps, nous procéderons à la modélisation sous MATLAB/Simulink, outil qui nous permettra de simuler le comportement de la machine dans différentes conditions de charge, et de générer les signaux nécessaires à l'entraînement du réseau de neurones utilisé dans le chapitre suivant.

Ce chapitre constitue ainsi la base analytique et numérique du travail, indispensable pour l'implémentation d'un estimateur de vitesse sans capteur.

2.2 Les Hypothèses Simplificatrices

Les hypothèses ci-dessous sont énoncées.

- Parfaite symétrie de construction.
- Le circuit magnétique n'est pas saturé.
- La variation des résistances des enroulements en fonction de la température est négligeable.
- L'entrefer est constant.
- Les valeurs des inductances propres et mutuelles sont indépendantes des intensités des courants.
- Les pertes fer, les pertes par hystérésis et par courants de Foucault sont nulles.
- La répartition dans l'entrefer de la force magnétomotrice et celle du flux est sinusoïdale.

La représentation schématique de la MAS dans l'espace électrique est donnée sur la Fig. (II.1), elle est munie de six enroulements [10].

Le stator de la machine est formé de trois enroulements fixes décalés de 120° dans l'espace et traversés par trois courants variables [9].

Le rotor peut être modélisé par trois enroulements identiques décalés dans l'espace de 120°, ces enroulements sont en court-circuit et la tension à leurs bornes est nulle.

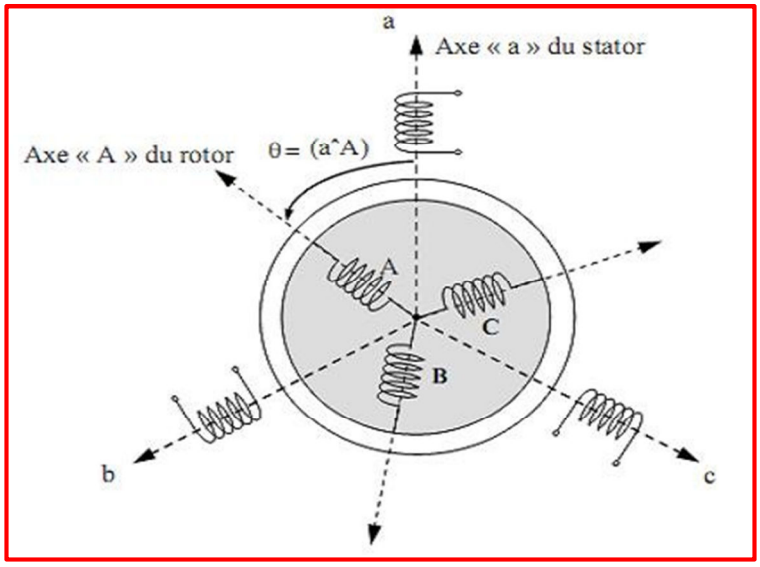

Figure 2.1 : Représentation schématique d'une machine asynchrone (MAS).

2.3 Mise en équations de la MAS

2.3.1 Equations électriques

Les six enroulements (a, b, c, A, B, C) représentés sur la figure (2.1) obéissent aux équations matricielles suivantes [11].

Au niveau du Stator :

$$[V_s] = [R_s][i_s] + \frac{d}{dt}[Q_s] \quad (2.1)$$

Avec :

$$[V_s] = \begin{bmatrix} V_{as} \\ V_{bs} \\ V_{cs} \end{bmatrix}; [i_s] = \begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix}; [Q_s] = \begin{bmatrix} Q_{as} \\ Q_{bs} \\ Q_{cs} \end{bmatrix}; [R_s] = R_s \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

Au niveau du Rotor :

$$[V_r] = [R_r][i_r] + \frac{d}{dt}[Q_r] = [0\ 0\ 0]^T \quad (2.2)$$

Avec :

$$[V_r] = \begin{bmatrix} \mathrm{V_{ar}} \\ \mathrm{V_{br}} \\ \mathrm{V_{cr}} \end{bmatrix}; [i_r] = \begin{bmatrix} \mathrm{i_{ar}} \\ \mathrm{i_{br}} \\ \mathrm{i_{cr}} \end{bmatrix}; [Q_r] = \begin{bmatrix} \mathrm{Q_{ar}} \\ \mathrm{Q_{br}} \\ \mathrm{Q_{cr}} \end{bmatrix}; [R_r] = R_r \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

Où :

R_s et R_r : Résistance d'une phase statorique et rotorique respectivement.

$[V_s]$: Vecteur tensions appliquées aux phases statoriques.

$[I_s]$: Vecteur courants appliqués aux phases statoriques.

$[Q_s]$: Vecteur flux statoriques.

$[V_r]$: Vecteur tensions appliquées aux phases rotoriques.

$[I_r]$: Vecteur courants appliqués aux phases rotoriques.

[Qr] : Vecteur flux rotoriques.

2.3.2 Equations magnétiques

Compte tenu des hypothèses de simplification exposées précédemment, les relations entre les flux et les courants de la machine asynchrone deviennent linéaires. Elles peuvent alors se formuler sous forme matricielle de la manière suivante [12] :

On a :

$$[Q_{abc}] = \begin{bmatrix} M_{sr} & L_s \\ L_r & M_{sr} \end{bmatrix} \begin{bmatrix} i_{abcs} \\ i_{abcr} \end{bmatrix} \quad (2.3)$$

Pour le stator :

$$\begin{bmatrix} \mathrm{Q_{as}} \\ \mathrm{Q_{bs}} \\ \mathrm{Q_{cs}} \end{bmatrix} = [L_s] \begin{bmatrix} \mathrm{i_{ar}} \\ \mathrm{i_{br}} \\ \mathrm{i_{cr}} \end{bmatrix} + [M_{sr}] \begin{bmatrix} \mathrm{i_{as}} \\ \mathrm{i_{bs}} \\ \mathrm{i_{cs}} \end{bmatrix} \quad (2.4)$$

Pour le rotor :

$$\begin{bmatrix} \mathrm{Q_{ar}} \\ \mathrm{Q_{br}} \\ \mathrm{Q_{cr}} \end{bmatrix} = [L_r] \begin{bmatrix} \mathrm{i_{as}} \\ \mathrm{i_{bs}} \\ \mathrm{i_{cs}} \end{bmatrix} + [M_{sr}] \begin{bmatrix} \mathrm{i_{ar}} \\ \mathrm{i_{br}} \\ \mathrm{i_{cr}} \end{bmatrix} \quad (2.5)$$

$$[L_s] = \begin{bmatrix} l_s & M_s & M_s \\ M_s & l_s & M_s \\ M_s & M_s & l_s \end{bmatrix} \quad (2.6)$$

$$[L_r] = \begin{bmatrix} l_r & M_r & M_r \\ M_r & l_r & M_r \\ M_r & M_r & l_r \end{bmatrix} \quad (2.7)$$

$$[L_{sr}] = [L_{rs}]^T = \mathrm{M} \begin{bmatrix} \cos(\theta) & \cos(\theta + \frac{2\pi}{3}) & \cos(\theta - \frac{2\pi}{3}) \\ \cos(\theta - \frac{2\pi}{3}) & \cos(\theta) & \cos(\theta + \frac{2\pi}{3}) \\ \cos(\theta + \frac{2\pi}{3}) & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta) \end{bmatrix} \quad (2.8)$$

$[L_{sr}]$: Inductance mutuelle entre le stator et le rotor.

Pour le stator :

$$[V_{abcs}] = [R_s][i_{abcs}] + \frac{d}{dt} \{[L_r][i_{abcs}] + [M_{sr}][i_{abcr}]\} \quad (2.9)$$

Pour le rotor :

$$[V_{abcr}] = [R_s][i_{abcr}] + \frac{d}{dt} \{[L_r][i_{abcr}] + [M_{sr}][i_{abcs}]\} \quad (2.10)$$

2.3.3 Equations mécaniques

L'équation mécanique de la machine et donnée par :

$$C_{em} = \mathrm{J}\frac{\mathrm{d}\Omega}{\mathrm{dt}} + C_r + f_\Omega \qquad (2.11)$$

Avec :

Cem : Le couple électromagnétique.

C_r : le couple résistant.

f : coefficient de frottement.

J : moment d'inertie du rotor [13].

2.4 Transformation de Park

La transformation de Park a pour objectif de représenter différentes machines électriques à courant alternatif selon un modèle unifié, en facilitant leur traitement mathématique. Elle permet de convertir les équations triphasées d'une machine en un système équivalent biphasé en coordonnées tournantes, ce qui simplifie significativement l'analyse et la commande [14].

Cette transformation, parfois appelée transformation d'axes, consiste à appliquer une rotation mathématique sur les enroulements physiques de la machine. Elle aboutit à un système d'enroulements fictifs mais équivalents du point de vue électrique et magnétique, dans un repère tournant. L'un des avantages majeurs de cette méthode est de rendre les inductances mutuelles constantes et indépendantes de l'angle de rotation, ce qui simplifie les équations du modèle [14].

- **Choix du repère (référentiel)**

L'isotropie de la machine asynchrone (symétrie de sa structure électromagnétique) permet d'écrire ses équations dans différents référentiels orthogonaux, en utilisant les composantes de Park. Le choix du repère a un impact direct sur la complexité des équations et leur utilisation en simulation ou commande.

Les trois référentiels les plus courants sont [14] :

- **Référentiel statorique :**

Immobile par rapport au stator, noté (α – β) $\rightarrow \omega = 0$

- **Référentiel rotorique :**

Tournant avec le rotor, noté (x - y) $\rightarrow \omega = \omega_r$

- **Référentiel synchrone (champ tournant) :**

Tournant avec le champ statorique, noté (d - q) $\rightarrow \omega = \omega_s$

La **matrice normalisée** de la transformation de Park est donnée par [15] :

$$[P] = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) & \cos(\theta - 2\frac{\pi}{3}) & \cos(\theta + 2\frac{\pi}{3}) \\ -\sin(\theta) & -\sin(\theta - 2\frac{\pi}{3}) & -\sin(\theta + 2\frac{\pi}{3}) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad (2.12)$$

Puisque cette matrice est orthogonale, on en déduit que sa matrice inverse est la même que sa matrice transposée :

$$[P]^{-1} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & \frac{1}{\sqrt{2}} \\ \cos(\theta + 2\frac{\pi}{3}) & -\sin(\theta - 2\frac{\pi}{3}) & \frac{1}{\sqrt{2}} \\ \cos(\theta + 4\frac{\pi}{3}) & -\sin(\theta - 4\frac{\pi}{3}) & \frac{1}{\sqrt{2}} \end{bmatrix} \qquad (2.13)$$

2.4.1 Equations électriques

On représente les tensions dans le repère de Park par l'équation suivante :

$$V_{sd} = R_s i_{sd} + \frac{d}{dt} Q_{sd} - \omega_s Q_{sq} \qquad (2.14)$$

$$V_{sq} = R_s i_{sq} + \frac{d}{dt} Q_{sq} - \omega_s Q_{sd} \qquad (2.15)$$

$$0 = R_r i_{rd} + \frac{d}{dt} Q_{rd} - (\omega_s - \omega_r) Q_{rq} \qquad (2.16)$$

$$0 = R_r i_{rq} + \frac{d}{dt} Q_{rq} - (\omega_s - \omega_r) Q_{rd} \qquad (2.17)$$

2.4.2 Equations magnétiques

$$Q_{sd} = L_s i_{sd} + M i_{rd} \qquad (2.18)$$

$$Q_{sq} = L_s i_{sq} + M i_{rq} \qquad (2.19)$$

$$Q_{rd} = L_r i_{rd} + M i_{sd} \qquad (2.20)$$

$$Q_{rq} = L_r i_{rq} + M i_{sq} \qquad (2.21)$$

Avec :

$$L_s = l_s - M \quad , \quad L_r = l_r - M \quad , \quad M = \frac{3}{2} M_0$$

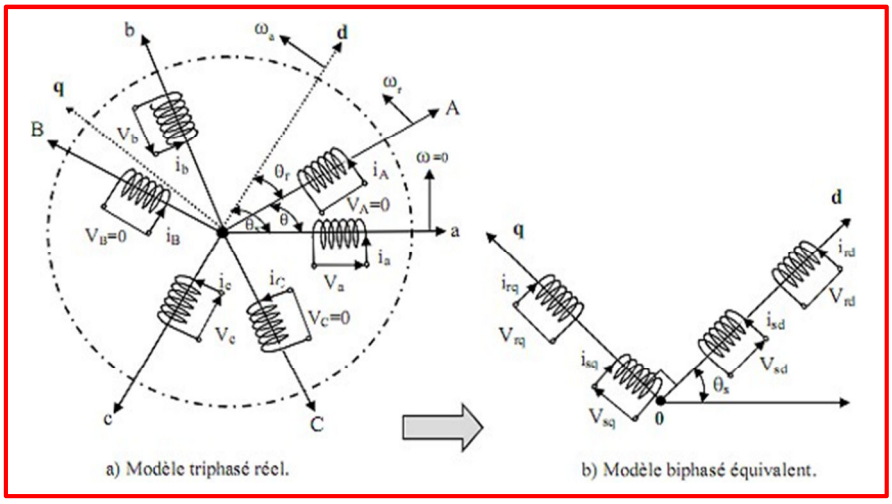

Figure 2.2 : Représentation de la machine asynchrone triphasée et sa machine biphasée.

2.4.3 Equations mécaniques

$$Cem = P\frac{M}{L_r}(Q_{rd}i_{sq} - Q_{rq}i_{sd}) \tag{2.22}$$

2.5 Représentation d'état du modèle de la MAS

La forme générale de l'équation d'état s'écrit de la façon suivante :

$$[\dot{X}] = [A][X] + [B][U] \tag{2.23}$$

Avec :

$$[X] = [i_{sd}i_{sq}Q_{rd}Q_{rq}]^{T} \tag{2.24}$$

$$[U] = [V_{sq}V_{sd}]^{T} \tag{2.25}$$

2.6 Représentation Mise en équation d'état

Les phénomènes transitoires dans la machine asynchrone peuvent être analysés à l'aide d'un modèle généralisé exprimé dans le repère fixe lié au stator, noté (α, β). Ce choix de référentiel permet d'étudier la dynamique de la machine sans introduire de rotation dans le système de coordonnées, ce qui simplifie certaines simulations de réponse transitoire [17].

Dans ce cadre, les tensions statoriques ($V_{s\alpha}$,$V_{s\beta}$) sont considérées comme grandeurs de commande, tandis que les courants statoriques ($i_{s\alpha}$,$i_{s\beta}$), les flux rotoriques ($Q_{r\alpha}$,$Q_{r\beta}$) et la vitesse mécanique $\boldsymbol{\Omega}_r$ constituent les variables d'état du système. Le couple résistant est quant à lui modélisé comme une perturbation extérieure [16].

On obtient ainsi un système d'équations différentielles qui décrit le comportement électromagnétique et mécanique de la machine dans ce repère fixe (α,β), en prenant en compte les couplages entre les différentes grandeurs dynamiques [18].

$$\begin{cases} V_{s\alpha} = R_s i_{s\alpha} + \frac{d}{dt} Q_{s\alpha} \\ V_{s\beta} = R_s i_{s\beta} + \frac{d}{dt} Q_{s\beta} \\ V_{r\alpha} = R_r i_{r\alpha} + \frac{d}{dt} Q_{r\alpha} + \omega_r Q_{r\beta} \\ V_{r\beta} = R_r i_{r\beta} + \frac{d}{dt} Q_{r\beta} + \omega_r Q_{r\alpha} \end{cases} \tag{2.26}$$

En substituant le système (2.26) dans (2.27), on aboutit à :

$$\begin{bmatrix} V_{s\alpha} \\ V_{s\beta} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -r & 0 & 0 & 0 \\ 0 & -r & 0 & 0 \\ \frac{M}{T_r} & 0 & \frac{-1}{T_r} & -\omega_r \\ 0 & \frac{M}{T_r} & \omega_r & \frac{-1}{T_r} \end{bmatrix} \begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \\ Q_{r\alpha} \\ Q_{r\beta} \end{bmatrix} = \begin{bmatrix} \sigma l_s & 0 & \frac{M}{l_r} & 0 \\ 0 & \sigma l_s & 0 & \frac{M}{l_r} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \\ Q_{r\alpha} \\ Q_{r\beta} \end{bmatrix} \tag{2.27}$$

On cherche à obtenir un système d'équation écrit sous forme d'équation d'état le modèle sera de la forme :

$$[\dot{X}] = [A][?] + [B][U] \quad (2.28)$$

Talque :

$[X] = [i_{s\alpha} i_{s\beta} Q_{r\alpha} Q_{r\beta}]^T$: Vecteur d'état.

[A] = Matrice d'évolution d'état du système.

[B] = Matrice de commande du système.

$[U] = [V_{s\alpha} V_{s\beta}]^T$: Vecteur de commande.

Apres tous les calculs, on trouve :

$$[A] = \begin{bmatrix} -(\frac{1}{\sigma T_s} + \frac{M^2}{l_r T_r}) & 0 & \frac{M}{\sigma l_s l_r T_r} & \frac{M}{\sigma l_s l_r} w_r \\ 0 & -(\frac{1}{\sigma T_s} + \frac{M^2}{l_r T_r}) & -\frac{M}{\sigma l_s l_r}\omega_r & \frac{M}{\sigma l_s l_r T_r} \\ \frac{M}{T_r} & 0 & -\frac{1}{Tr} & \omega_r \\ 0 & \frac{M}{T_r} & -\omega_r & -\frac{1}{T_r} \end{bmatrix} \quad (2.29)$$

$$[B] = \begin{bmatrix} \frac{1}{\sigma l_s} & 0 \\ 0 & \frac{1}{\sigma l_s} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (2.30)$$

Avec :

Le coefficient de la fuite totale :

$$\sigma = 1 - \frac{M^2}{l_s l_r} \quad (2.31)$$

Constante de temps statorique :

$$T_s = \frac{l_s}{R_s} \quad (2.32)$$

Constante de temps rotorique :

$$T_r = \frac{l_r}{R_r} \quad (2.33)$$

2.7 Mise sous forme de blocs de simulation

Dans le but d'analyser le comportement dynamique de la machine asynchrone triphasée, un modèle a été élaboré sous MATLAB/Simulink, en se basant sur ses équations électriques, magnétiques et mécaniques.

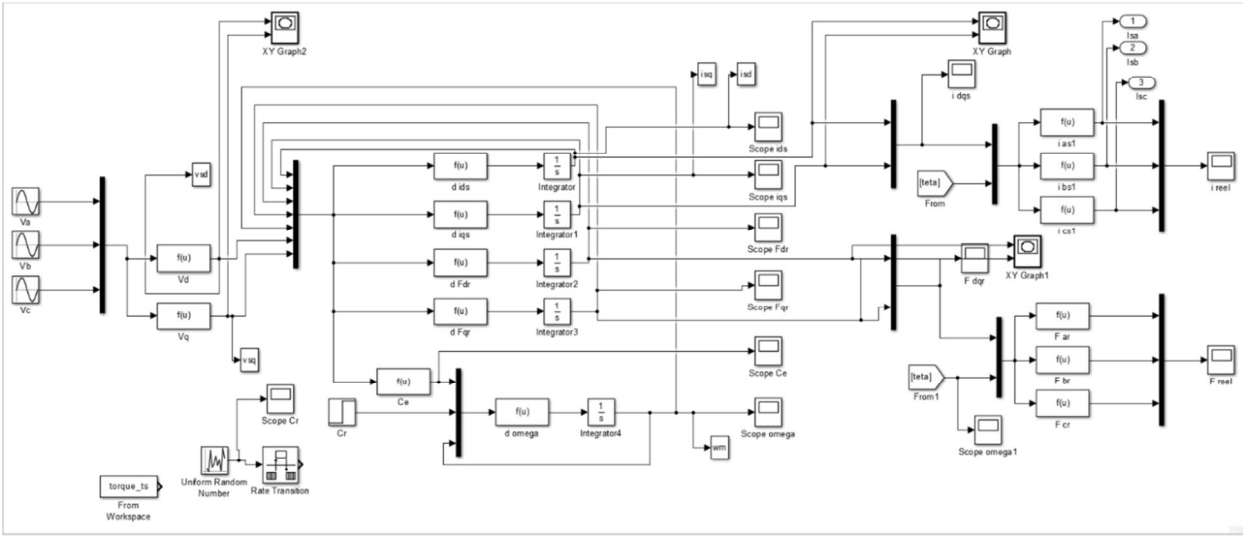

Figure 2.3 : Schéma bloc de la modélisation dynamique de la MAS dans le repère dq.

Cette figure représente la modélisation mathématique complète de la machine asynchrone dans le repère tournant (d–q). On y retrouve :

- Les entrées de tension triphasées V_a V_b V_c.
- La transformation en composantes V_{ds} V_{qs}.
- Les équations différentielles des courants statoriques et flux rotoriques.
- Le calcul du couple électromagnétique C_e et de la vitesse ω_r.
- La transformation inverse pour obtenir les courants i_a i_b i_c et flux réels.

Avec : couple de charge 0.5 et temps de simulation 1ms.

- **Code Matlab pour la définition des variables utilisées dans le schéma Simulink:**

```
clear all
clc
Rr=13.6324; Rs=13.3072;
Lr=0.678; Ls=0.678; Lm=0.638;
J=0.00177007; fv=0.000643777; p=2; fs=0;

sig=1-(Lm^2)/(Ls*Lr);
a1=Rs/(sig*Ls)+Rr*(Lm^2)/(sig*Ls*(Lr^2));
a2=Lm*Rr/(sig*Ls*Lr^2);
a3=(Lm/(sig*Ls*Lr))*p;
b=1/(sig*Ls);
aa1=Rr*Lm/Lr;
bb1=Rr/Lr;
alph1=p*Lm/Lr;
alph2=1/j;
alph3=fv/j;
```

Figure 2.4 : Code Matlab.

2.8 Résultats de simulation

Après la mise en place du modèle Simulink, une série de simulations a été effectuée pour observer l'évolution des grandeurs électriques et mécaniques de la machine. Les signaux obtenus, tels que les courants, tensions et vitesses.

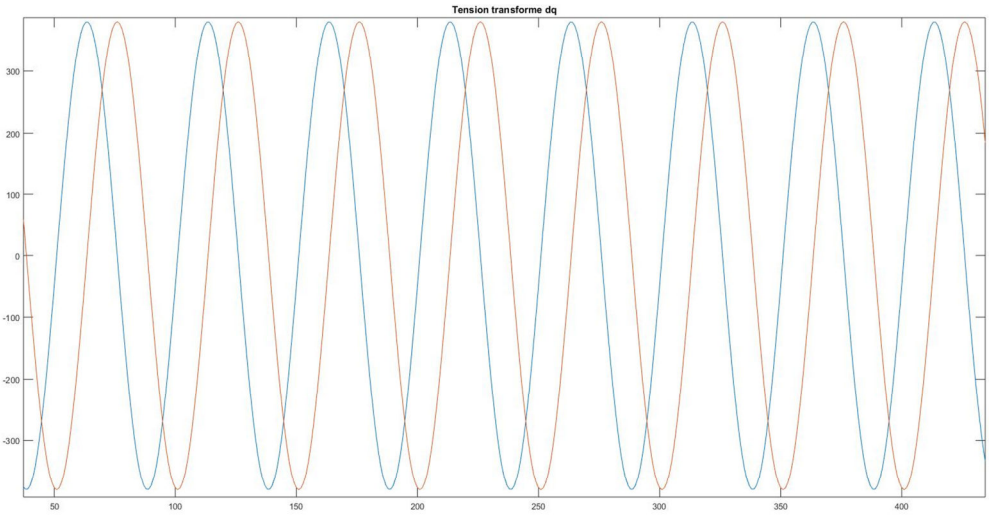

Figure 2.5 : Tension statorique dans le repère (d,q).

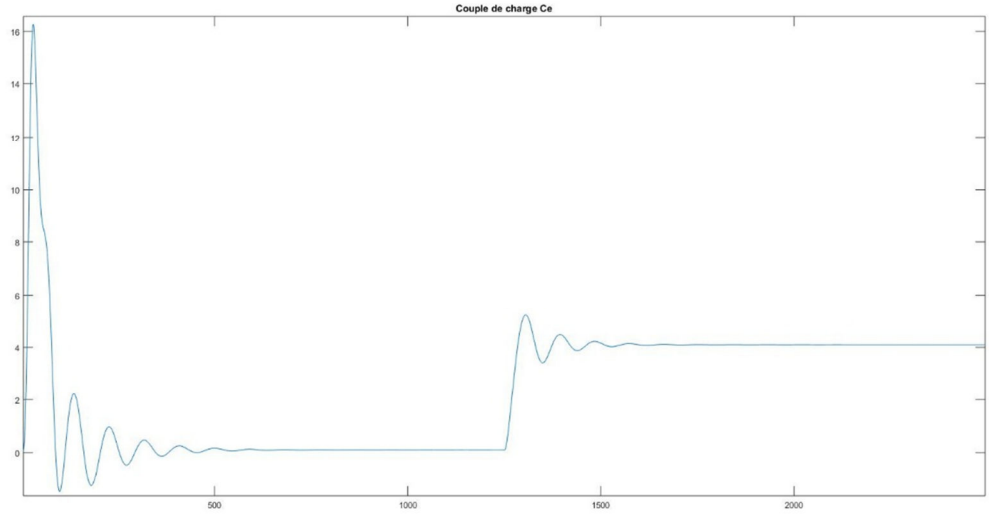

Figure 2.6 : Réponse du couple de charge à une perturbation à t = 0.5 s.

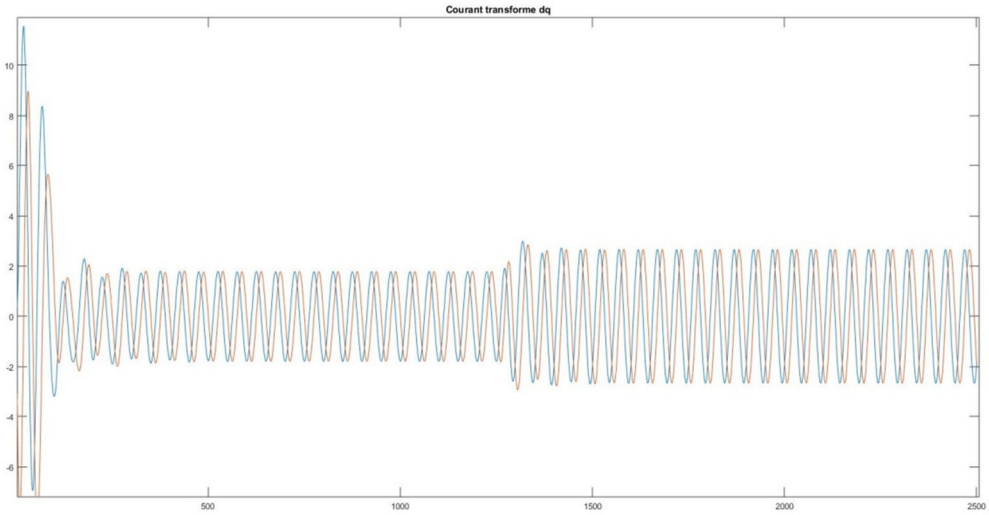

Figure 2.7 : Courant statorique Isd Isq.

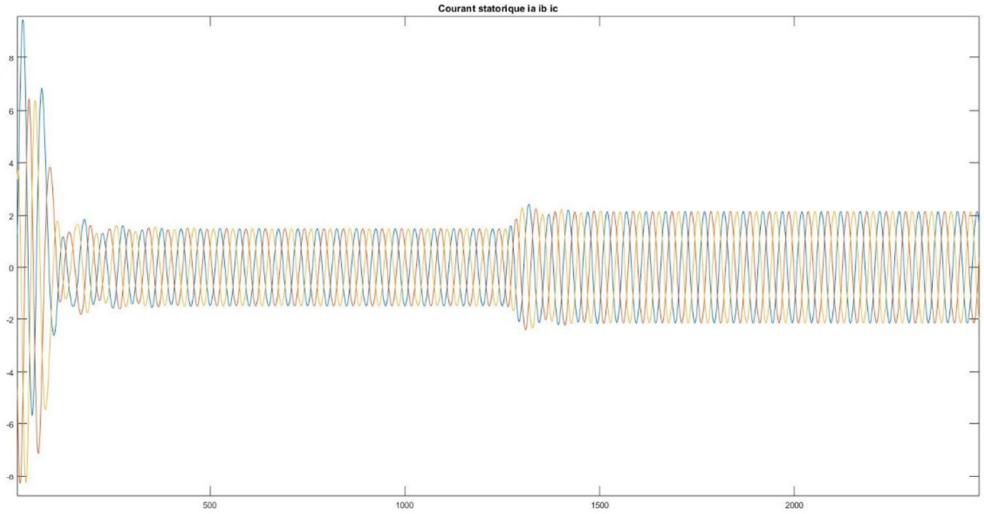

Figure 2.8 : Courant statorique I_{sabc}.

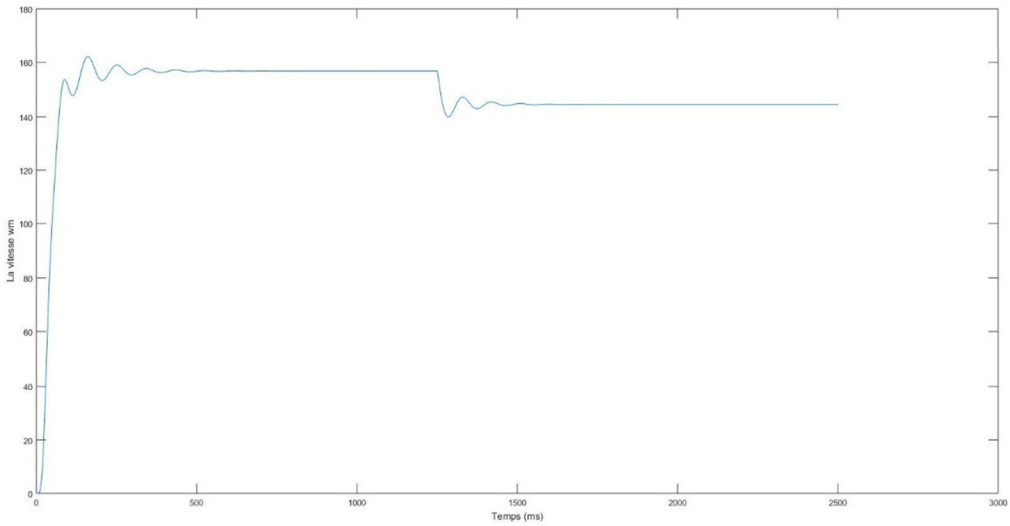

Figure 2.9 : Vitesse rotorique (wm).

- **Interprétation de résultats**

Les figures obtenues à l'issue de la simulation permettent d'analyser le comportement dynamique de la machine asynchrone face à une variation de couple de charge.

La figure 2.4 présente la tension statorique dans le repère (d,q), à savoir les composantes V_{sd} et V_{sq}. Ces tensions varient de manière sinusoïdale, ce qui reflète l'évolution dynamique de la commande et son adaptation au comportement de la machine.

La figure 2.5 montre l'évolution du couple de charge appliqué à la machine. On y observe un saut brusque vers t=0.5s, représentant une perturbation volontaire du système. Ce changement permet d'étudier la réponse dynamique de la machine.

Les courants statoriques i_{sd} et i_{sq}, représentés dans la figure II.6, suivent une évolution cohérente avec celle du couple. En effet, le couple électromagnétique développé par la

machine est directement lié à ces composantes de courant, notamment i_{sq}. Ainsi, on remarque une élévation de ces courants au moment de l'augmentation du couple, confirmant que le courant est proportionnel au couple moteur.

La figure 2.7 illustre les composantes triphasées du courant statorique $i_a i_b i_c$. Leur forme sinusoïdale équilibrée en régime permanent confirme le bon fonctionnement de la machine. On note une hausse de l'amplitude à partir de t=0.5s, synchronisée avec l'augmentation du couple, ce qui valide le lien entre charge et intensité de courant.

Enfin, la figure 2.8 présente la vitesse rotorique ω_r Après une phase transitoire rapide, la vitesse se stabilise avant de chuter légèrement au moment du changement de couple. Cette baisse est attendue : lorsqu'un couple résistant augmente, la machine ralentit momentanément avant de retrouver un nouvel équilibre.

2.9 Conclusion

Dans ce chapitre, nous avons développé la modélisation mathématique complète de la machine asynchrone triphasée, en passant du repère triphasé (abc) au repère (dq) via la transformation de Park. Cette modélisation a permis de décrire précisément les dynamiques électriques et mécaniques de la machine. Les simulations réalisées sous MATLAB/Simulink ont confirmé la validité du modèle en reproduisant fidèlement le comportement de la machine face à des variations de charge. Ces résultats constituent une base essentielle pour l'estimation de la vitesse sans capteur par réseau de neurones, objet du chapitre suivant [19].

Chapitre 3 Estimation de la vitesse par réseau de neurones artificiels

3.1 Introduction

Estimer la vitesse d'un moteur asynchrone sans utiliser de capteur mécanique présente des avantages en termes de coût et de fiabilité pour l'entretien du moteur électrique en question. Les réseaux de neurones artificiels se révèlent être un choix efficace pour cette estimation basée sur des données électriques grâce à leur capacité à apprendre des relations complexes et non-linéaires.

Ce chapitre exposé la méthode suivie pour développer le réseau neuronal et préparer les données avant de l'entraîner sous MATLAB et d'évaluer ses performances.

3.2 Historique de réseaux de neurones

Le concept de réseaux neuronaux n'est pas nouveau ; l'idée initiale était de concevoir un système capable de modéliser la bio physiologie du cerveau. Ce modèle tente d'expliquer le fonctionnement et les mécanismes du cerveau [20].

En 1943, McCulloch et Pitts ont donné leur nom à un modèle de neurone biologique (un neurone au comportement binaire). Ils furent les premiers à démontrer que des réseaux neuronaux formels simples pouvaient effectuer des fonctions logiques, Cependant, ce modèle a rapidement montré ses limites, notamment son incapacité à résoudre le problème du XOR, qui n'est pas linéairement séparable. En 1957, Rosenblatt a développé le modèle du Perceptron. Il a construit le premier neuro-

ordinateur basé sur ce modèle et l'a appliqué au domaine de la reconnaissance de formes [21].

En 1960, Widrow, un spécialiste du contrôle, a créé le modèle Adaline (élément linéaire adaptatif). Dans sa structure, ce modèle ressemble au Perceptron, mais il utilise une loi d'apprentissage différente. C'est à partir de là que l'algorithme de rétropropagation du gradient, largement utilisé aujourd'hui avec les perceptrons multicouches, a ses racines. Les réseaux de type Adaline sont encore utilisés de nos jours pour certaines applications spécifiques. Widrow a également fondé l'une des premières entreprises commercialisant des neuro-ordinateurs et des neuro-composants à cette époque.

En 1985, l'algorithme de rétropropagation du gradient est apparu. Il s'agit d'un algorithme d'apprentissage adapté aux réseaux neuronaux multicouches (également appelés perceptrons multicouches). Sa découverte, faite indépendamment par trois groupes de chercheurs, montre que « l'idée était dans l'air ». Grâce à cette découverte, il devient possible de réaliser une fonction d'entrée/sortie non linéaire dans un réseau en décomposant cette fonction en une série d'étapes linéairement séparables. Aujourd'hui, les réseaux multicouches avec rétropropagation du gradient demeurent le modèle le plus étudié et le plus exploité en termes d'applications.

3.3 Réseaux de neurones artificiels

Les réseaux neuronaux artificiels sont des réseaux fortement interconnectés de processeurs simples fonctionnant en parallèle. Chaque processeur calcule une sortie spécifique à partir des données qu'il reçoit. Toute structure hiérarchique de réseaux constitue, bien entendu, un réseau.

3.3.1 Le neurone formel

Le premier modèle du neurone formel a été présenté par McCulloch et Pitts (Figure 2.1). De manière générale, un neurone formel est un élément de traitement qui effectue

une sommation pondérée de *n* entrées x1,…,xn. Si cette somme dépasse un certain seuil (fonction de seuillage F), le neurone est activé et transmet une réponse dont la valeur correspond à celle de son activation. Si le neurone n'est pas activé, il ne transmet rien [22].

$$y_i = f\left(\sum_{i=1}^{n} w_{ij} x_j\right) \quad (3.1)$$

w_{ij} : Coefficient synaptique ou poids associe à la i^{eme} entrée du neurone j.

Parfois, il y a un terme additionnel θ_j appelé biais, est ajouté et représente le seuil interne du neurone. Ce biais est considéré comme un poids supplémentaire θ_{0j} associe à une entrée constante égale à 1. L'expression (3.1) devient donc :

$$S_i = \sum_{i=1}^{n} (w_i x_j + \theta_j) \quad (3.2)$$

$$y_i = f(S_i) \quad (3.3)$$

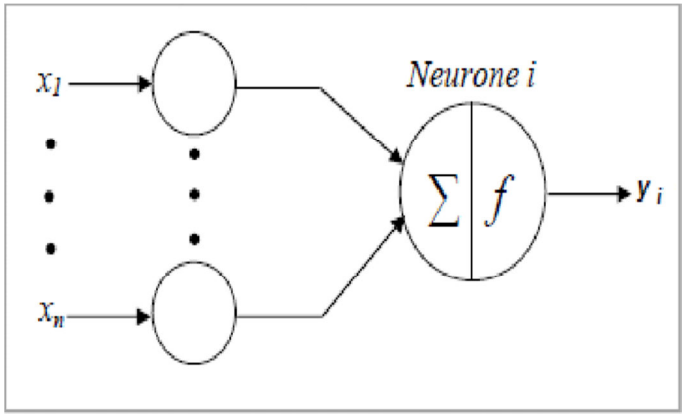

Figure 3.1 : Schéma d'un neurone formel.

Les réseaux de neurones artificiels (RNA) sont des modèles mathématiques inspirés du fonctionnement du cerveau humain. Chaque neurone artificiel reçoit un signal d'entrée, applique un poids, ajoute un biais et transmet une sortie via une fonction d'activation. Ce processus est similaire à la transmission synaptique dans les cerveaux biologiques [23].

3.3.2 La fonction d'activation ou de seuillage

Elle permet d'introduire de la **non-linéarité**, essentielle pour modéliser des relations complexes. Les plus utilisées sont :

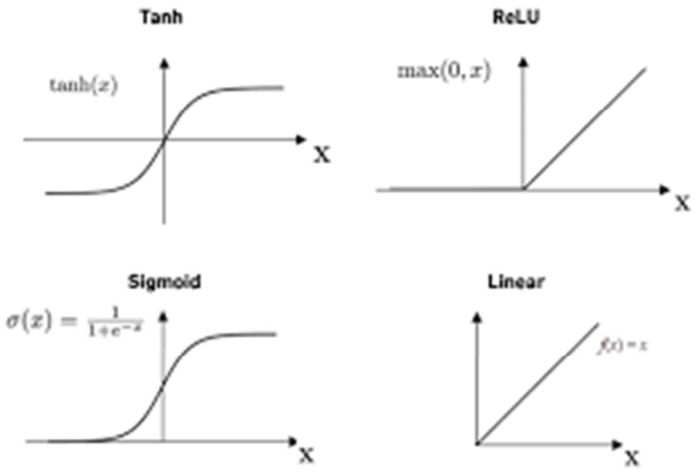

Figure 3.2 : Les fonctions d'activation les plus utilisées.

Parmi les fonctions d'activation classiques, seule la fonction sigmoïde tangente est décrite ici car elle a été employée dans notre architecture MLP.

- **Sigmoïde tangente :**

$$tansig(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \qquad (3.4)$$

- **Linear** :

$$f(x) = \mathrm{x} \qquad (3.5)$$

3.4 Architecture des réseaux de neurone artificiel

Les associations entre les neurones qui composent l'organisation décrivent la « topologie » du réseau. Elles sont extrêmement variées, le nombre d'associations étant

considérable. Cette topologie révèle une certaine cohérence dans le fonctionnement des neurones.

L'architecture la plus courante est le **réseau multicouche (MLP – Multilayer Perceptron)**.

3.4.1 Réseau Multicouches

Les RNA sont organisées en couches, il n'y a pas d'associations entre les neurones d'une même couche, et les associations ne se font qu'avec les neurones des couches inférieures. Chaque neurone d'une couche est associé à tous les neurones d'une autre couche [24].

Typiquement appelé :

- **Couche d'entrée** : l'ensemble des neurones d'entrées.
- **Couche cachées** : les couches intermédiaires n'ayant aucun contact avec l'extérieur.
- **Couche de sortie** : l'ensemble des neurones de sorties.

En général, la couche d'entrée est une couche passive. Ses neurones n'effectuent aucun traitement.

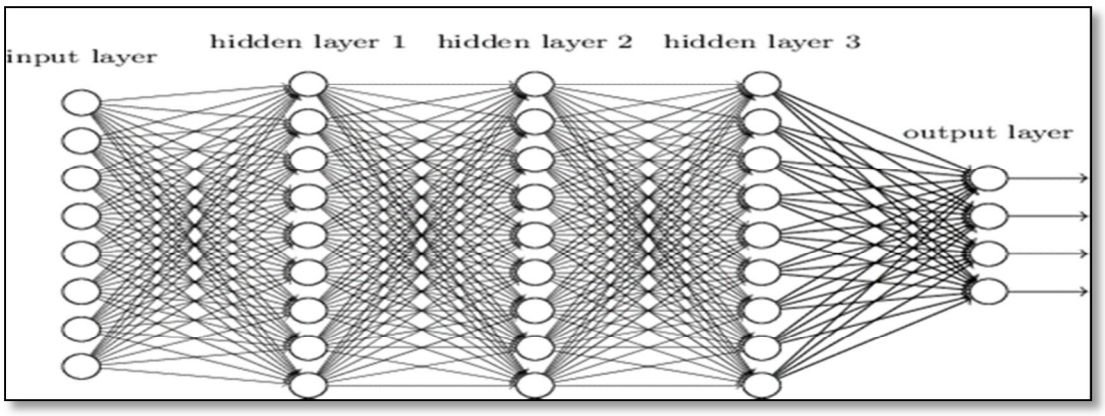

Figure 3.3 : réseau multicouche [25].

3.5 Apprentissage des réseaux de neurone

Pour un système neuronal, l'apprentissage peut être considéré comme la question de l'amélioration des poids synaptiques à l'intérieur du système, afin de réussir la tâche qu'on lui demande d'accomplir. L'apprentissage est l'élément le plus important des systèmes neuronaux et peut prendre de nombreuses formes [26].

3.5.1 Mode supervisé

Dans ce type d'apprentissage, le réseau de neurones s'entraîne à partir de paires entrée-sortie connues. Il compare la sortie qu'il produit avec la sortie attendue (cible), puis ajuste ses poids afin de minimiser l'erreur.
Ce mode est utilisé lorsqu'on dispose de données annotées, comme c'est le cas dans notre projet.

3.5.2 Mode non supervisé

Dans l'apprentissage non supervisé, aucune sortie cible n'est fournie. Le réseau apprend à détecter des structures ou des régularités dans les données d'entrée, comme des groupes, des clusters ou des corrélations, sans supervision externe.
Ce mode est utilisé notamment pour la classification automatique ou la réduction de dimension.

3.6 Le perceptron multicouche

Un réseau neuronal multicouche, ou MLP (Multi-layer Perceptron), est composé de plusieurs couches de perceptrons reliées entre elles, généralement divisées en une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Ce type de perceptron, diffère du perceptron de Rosenblatt principalement en ce qu'il utilise une fonction d'activation continue (généralement une sigmoïde) plutôt que la fonction Heaviside utilisée à l'origine.

Dans un MLP, chaque neurone d'une couche reçoit des signaux de tous les neurones de la couche précédente (entièrement connectés) et transmet sa sortie à

tous les neurones de la couche suivante. Il convient de noter que les neurones d'une même couche ne sont pas interconnectés entre eux, ce qui garantit une propagation unidirectionnelle des signaux [27].

3.6.1 Architecture

Un réseau multicouche contient [23] :

- Une couche d'entrée : qui sert à représenter les entrées sans faire de traitement sur ces dernières.
- Une ou plusieurs couches cachées.
- Une couche de sortie : en contact avec le monde extérieur.

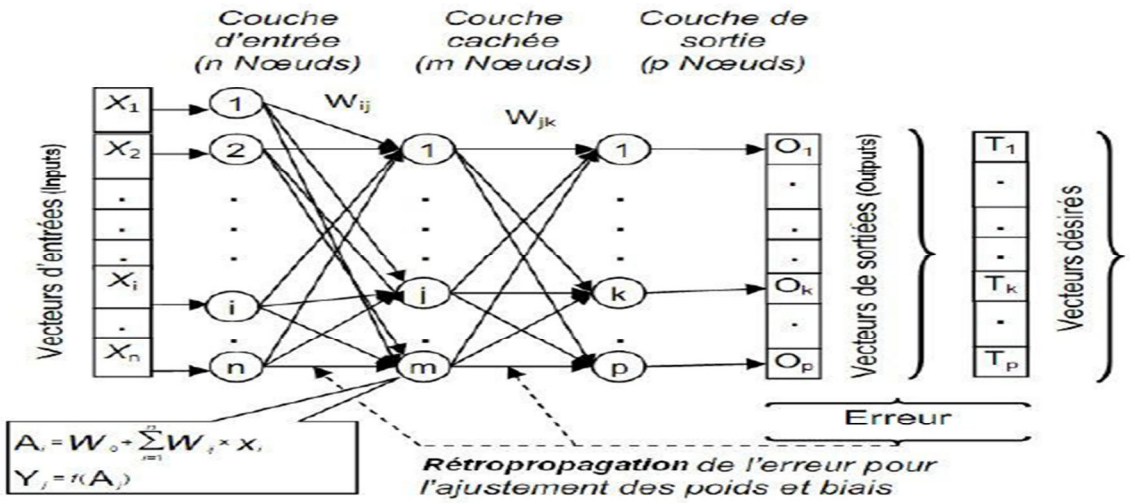

Figure 3.4 : Architecture du perceptron model MLP.

3.6.2 L'apprentissage des réseaux MLP

Pendant de nombreuses années, les réseaux neuronaux artificiels étaient limités à des structures à une seule couche, en raison de l'absence de méthode efficace pour ajuster les poids dans les couches cachées. Le principal obstacle résidait dans la

difficulté à calculer l'erreur associée aux neurones de ces couches intermédiaires, ce qui empêchait le développement de véritables modèles multicouches.

Ce problème a été résolu dans les années 1980 grâce à la découverte de l'algorithme de rétropropagation de l'erreur. Cette méthode a marqué un tournant majeur, en permettant l'apprentissage supervisé des réseaux multicouches (MLP) en propageant l'erreur de la couche de sortie vers les couches cachées [28].

Le calcul est basé sur l'**algorithme de la descente du gradient**, qui repose sur la **différentiation** de la fonction d'erreur. Cette méthode permet de minimiser l'erreur entre la sortie réelle et la sortie désirée, en ajustant les poids du réseau couche par couche.

La méthode comprend les étapes suivant :

- Le passage en avant des données d'entrée vers la sortie.
- Calculer l'erreur par rapport à l'estimation prévue.
- Rétro-propager cette erreur à travers le réseau, en suivant les règles de la détermination en chaîne.
- Réviser les poids en déplaçant les coefficients dans l'axe inverse de la pente de l'erreur.

Cet algorithme permet ainsi à un MLP de modifier ses paramètres internes (poids et biais) afin de minimiser l'erreur globale et d'apprendre une relation non linéaire entre les variables d'entrée et de sortie [26].

3.6.3 Rétropropagation de l'erreur (Back propagation)

L'algorithme de rétropropagation repose sur le principe de la descente de gradient. Pour chaque exemple d'apprentissage, une propagation avant est effectuée à travers le réseau, suivie du calcul de l'erreur en sortie. Cette erreur est ensuite rétropropagée vers les couches précédentes afin de déterminer les gradients. Ces gradients sont

utilisés pour mettre à jour les poids synaptiques du réseau et ainsi réduire progressivement l'erreur globale [26].

Propagation directe (Forward pass)

$$a_j = \sum_i w_{ij} \cdot x_j + b_j \quad (3.6)$$

$$y_j = f(a_j) \quad (3.7)$$

Calcul de l'erreur en sortie (Erreur quadratique)

$$E = \frac{1}{2} \sum_j (d_j - y_j)^2 \quad (3.8)$$

Descente de gradient (Gradient Descent)

Cette méthode ajuste les poids dans la direction négative du gradient de l'erreur [29] :

$$w_{new} = w_{old} - \tau \cdot \frac{\partial E}{\partial w}$$

(3.9)

τ : est le taux d'apprentissage.

$\frac{\partial E}{\partial w}$: est le gradient de l'erreur.

Algorithme de Levenberg–Marquardt (LM)

L'algorithme de Levenberg-Marquardt est une méthode d'optimisation numérique non linéaire utilisée pour accélérer l'apprentissage supervisé des réseaux de neurones, en particulier des réseaux multicouches de type MLP [30].

Contrairement à la descente de gradient classique, l'algorithme LM combine les avantages de :

- La méthode de **Gauss-Newton** (rapide mais parfois instable),
- La **descente de gradient** (stable mais lente).

$$\Delta\omega = -(J^T J + \mu I)^{-1} J^T e \quad (3.10)$$

$\Delta\omega$: Correction des poids.

J : La **jacobienne** de la fonction d'erreur par rapport aux poids.

μ : Paramètre d'amortissement (adaptatif).

I : Matrice identité.

e : Vecteur des erreurs ($e_i = d_i - y_i$).

3.7 Préparation des données d'entraînement

Les données utilisées pour l'apprentissage du réseau de neurones ont été extraites à partir de la simulation MATLAB/Simulink du modèle de la machine asynchrone développé au chapitre 2 **figure 2.3**.

Les courants statoriques : i_a i_b i_c

Les courants statoriques dq : i_{sd} i_{sq}

Les tensions d'alimentation : V_{sd} V_{sq} .

La vitesse rotorique : ω_m.

Dans cette étude, on a développé 2 cas d'entrainement du réseau de neurone.

Toutes les données ont été normalisées dans l'intervalle [−1,1] afin de faciliter l'apprentissage. Elles ont ensuite été organisées sous forme de matrices.

3.7.1 Caractéristiques des deux cas de données pour l'apprentissage

Tableau 3.1 : Caractéristiques des deux cas de données pour l'apprentissage

Caractéristique	1^{er} Cas	2^{eme} Cas
Temps de début	0.0s	0.0s
Temps de fin	200s	100s
Pas d'échantillonnage Te	0.0004s	0.0005s
Entrées	$[i_a , i_b , i_c]$	$[i_{sd} , i_{sq} , V_{sd} , V_{sq}]$
Sortie	ω_m	ω_m
Taille des entrées	3×500001	4×200001
Taille des sorties	1×500001	1×200001

3.8 Entrainement des réseaux de neurones

La préparation a été effectuée dans MATLAB à l'aide de la boite à outils RNN. L'objectif est de permettre à l'arrangement neuronal de mémoriser l'évaluation de la vitesse du rotor à partir de diverses combinaisons d'entrées électriques.

3.8.1 Architecture du réseau

Le réseau utilisé est de type perceptron multicouche (MLP), dont les paramètres sont donnés par le **tableau 3.2** :

Tableau 3.2 : Paramètres des réseaux MLP utilisés pour les deux cas.

Paramètre	1^{er} Cas	2^{eme} Cas
Nombre d'entrée	3	4
Nombre de couche cachée	1	1
Neurone dans la couche cachée	20	15
Nombre de neurone en sortie	1(vitesse estimée ω_m)	1(vitesse estimée ω_m)
Fonction d'activation (cachée)	Sigmoïde tangente (tansig)	Sigmoïde tangente (tansig)
Fonction d'activation (sortie)	Linéaire (pureline)	Linéaire (pureline)
Algorithme d'apprentissage	Levenberg-Marquardt (trainlm)	Levenberg-Marquardt (trainlm)

3.8.2 Les étapes d'entrainement avec nnstart :

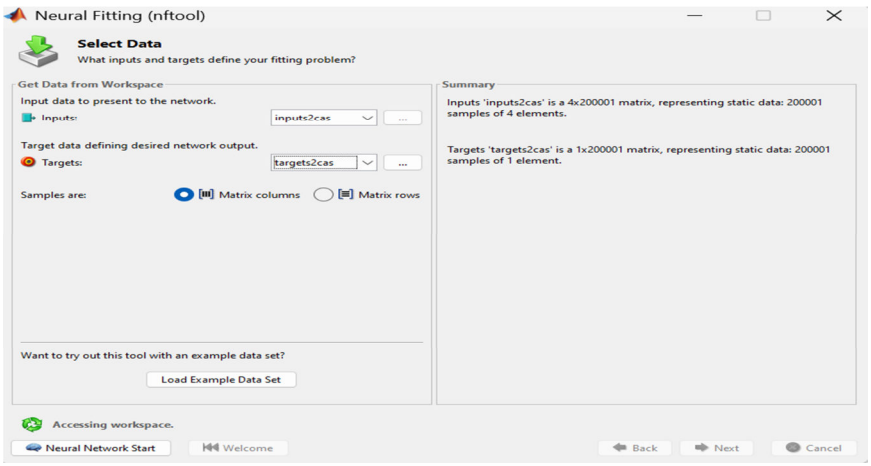

Figure 3.5 : Préparer les données d'entrainement.

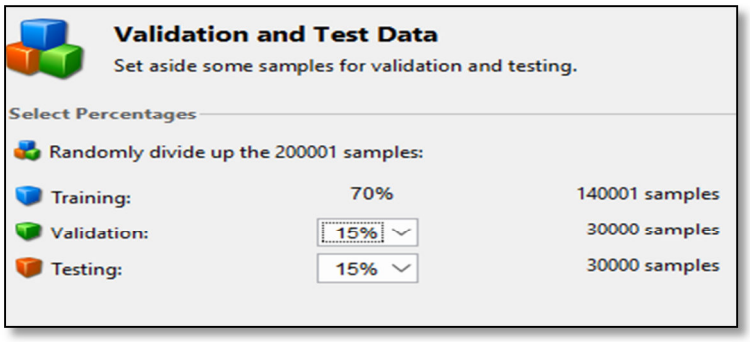

Figure 3.6 : Validation et données d'essai.

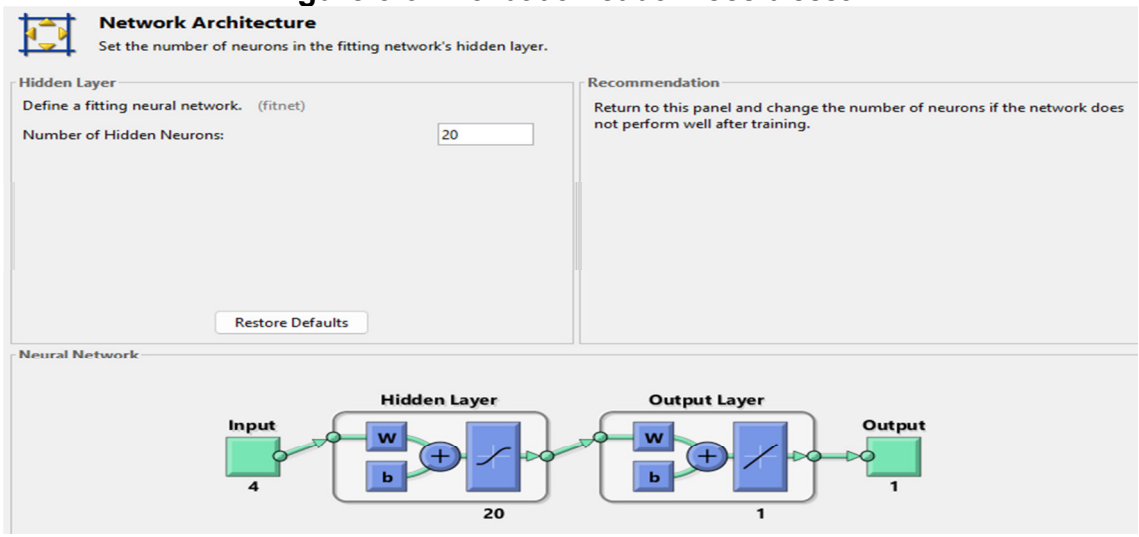

Figure 3.7 : Architecture du réseau neurone

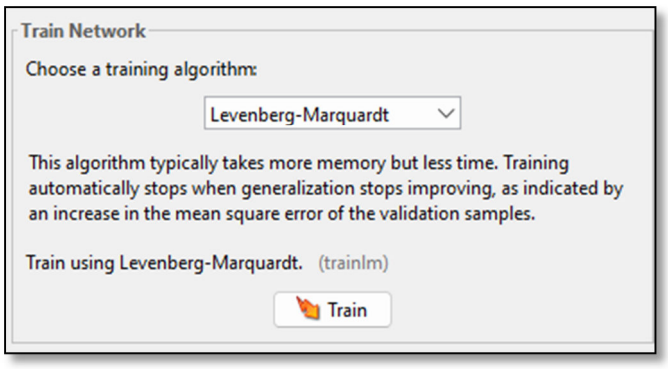

Figure 3.8 : Algorithme d'apprentissage.

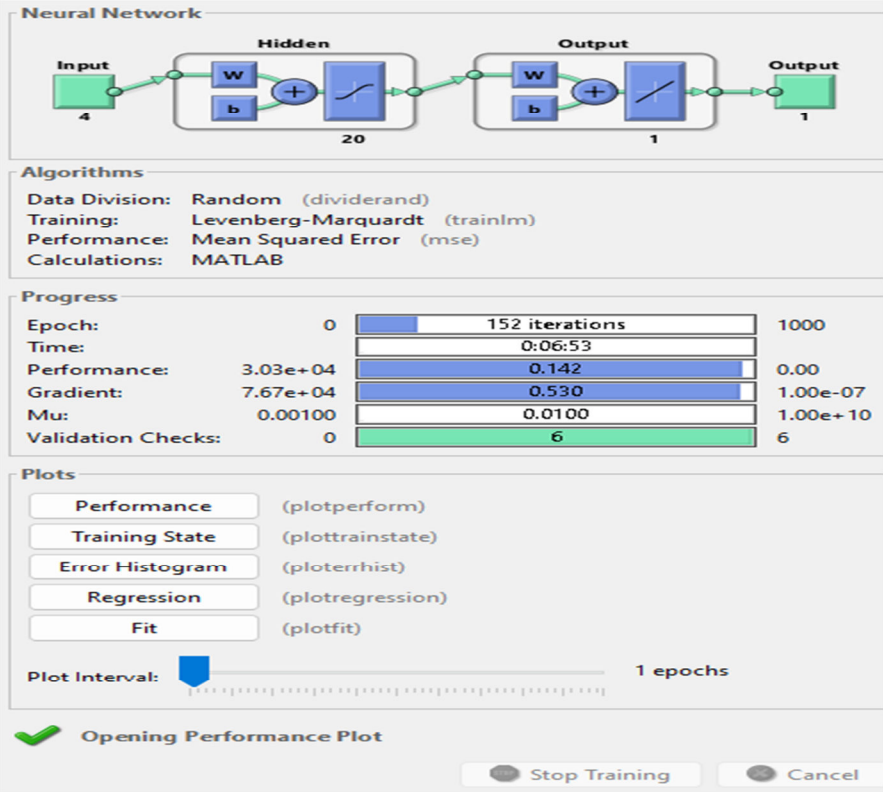

Figure 3.9 : Réseau de neurone.

a 1er Cas : Entrainement avec (i_a i_b i_c)

Afin de déterminer la structure la plus performante, plusieurs tailles de couches couvertes ont été essayées : 10, 15 et 20 neurones. Chaque arrangement a été évalué en termes de vitesse de fusion, d'exactitude d'estimation (MSE) et de qualité de régression (R).

Tableau 3.3 : Comparaison des performances selon le nombre de neurones dans la couche cachée Cas 1.

Nombre de neurones	Erreur quadratique Moyenne (MSE)	Coefficient de régression (R)
10	1.5716	0.98897
15	1.3859	0.98961
20	1.2361	0.99114

Comme le montre le tableau ci-dessus, l'augmentation du nombre de neurones dans la couche cachée permet d'améliorer continuellement l'exécution. La configuration avec 20 neurones a donné la meilleure précision (MSE = 1.2361) et une excellente régression (R = 0,99114). Il a donc été adopté comme modèle ultime pour le reste de l'étude.

- **Architecture :**

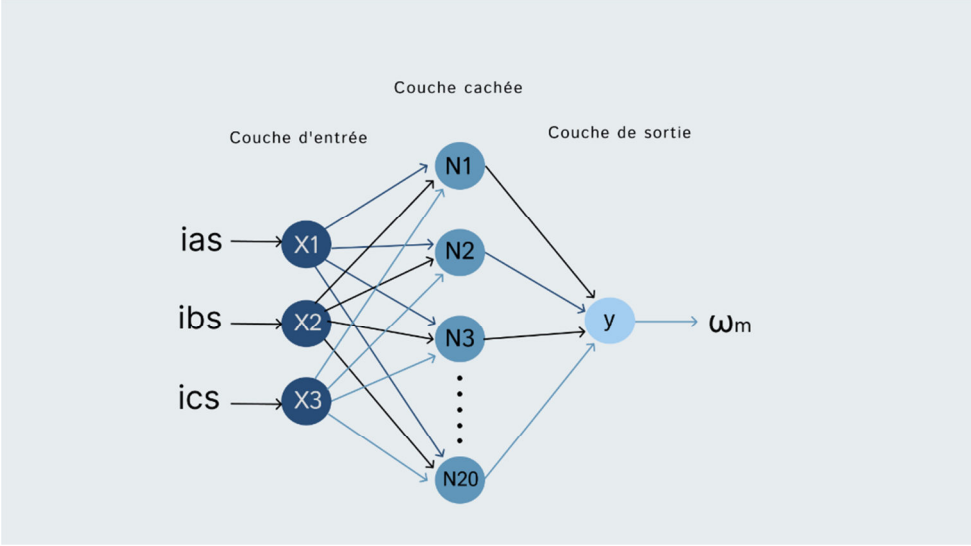

Figure 3.10 : Architecture du réseau de neurone Cas 1.

- **Performance :**

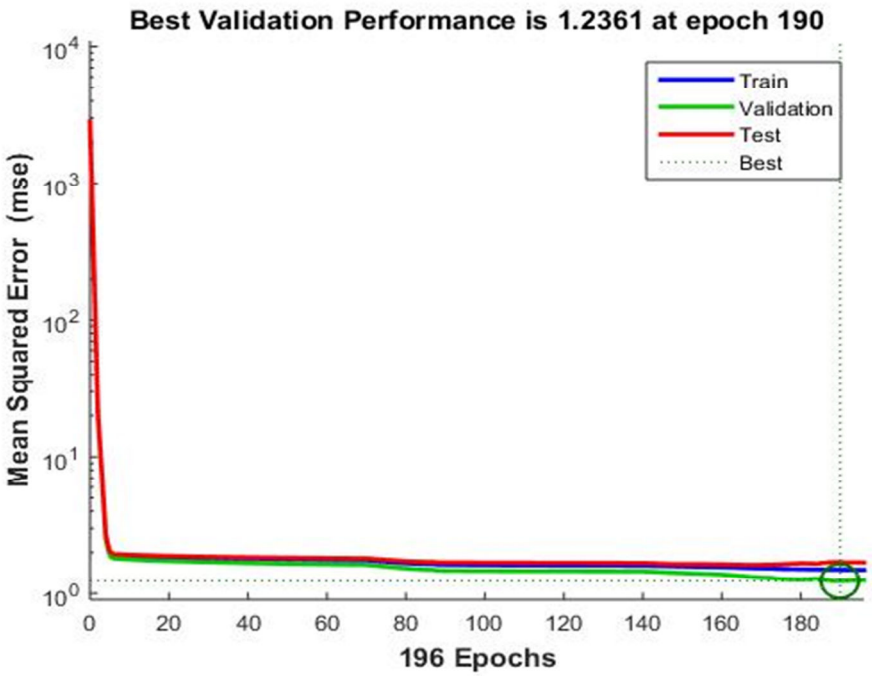

Figure 3.11 Performance de 1er cas.

- **Régression :**

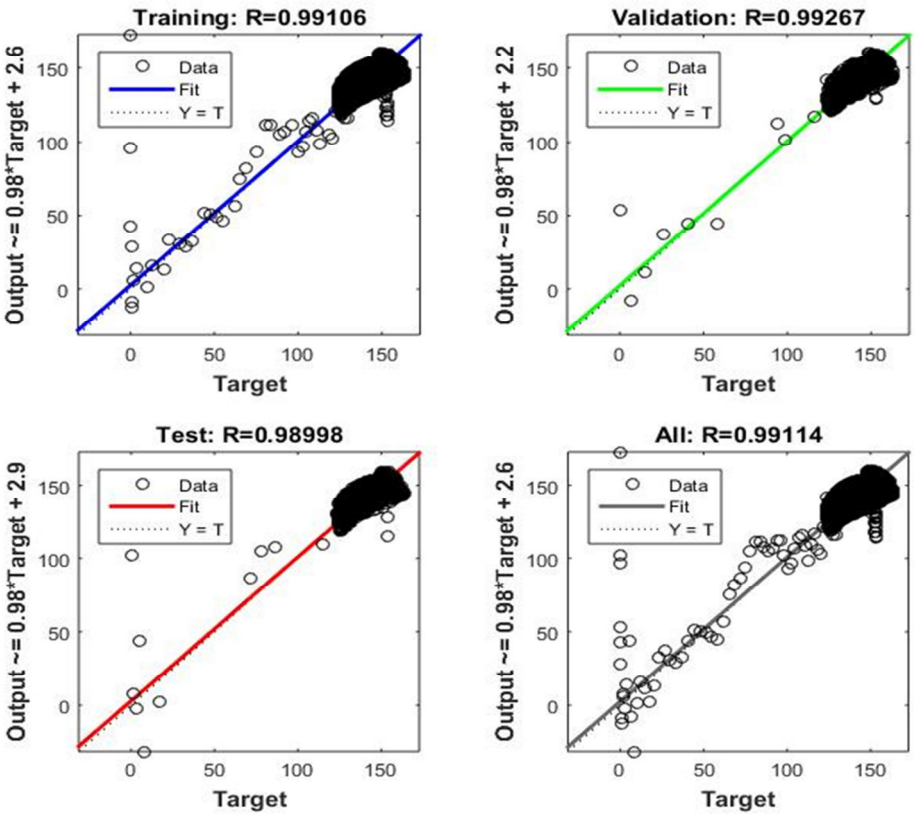

Figure 3.12 : Régression de 1er cas.

b 2ème cas : Entrainement avec (i_{sd} i_{sq} v_{sd} v_{sq})

Afin de déterminer la structure la plus performante, plusieurs tailles de couches cachées ont été essayées : 10, 15 neurones. Chaque arrangement a été évalué en termes de vitesse de fusion, d'exactitude d'estimation (MSE) et de qualité de régression (R).

Tableau 3.4 : Comparaison des performances selon le nombre de neurones dans la couche cachée Cas 2.

Nombre de neurones	Erreur quadratique Moyenne (MSE)	Coefficient de régression (R)
10	0.46995	0.99718
15	0.35702	0.99791

Comme le montre le tableau ci-dessus, l'augmentation du nombre de neurones dans la couche cachée permet d'améliorer continuellement l'exécution. La configuration avec 15 neurones a donné la meilleure précision (MSE = 0.357) et une excellente régression (R = 0,9979). Il a donc été adopté comme modèle ultime pour le reste de l'étude.

- **Architecture :**

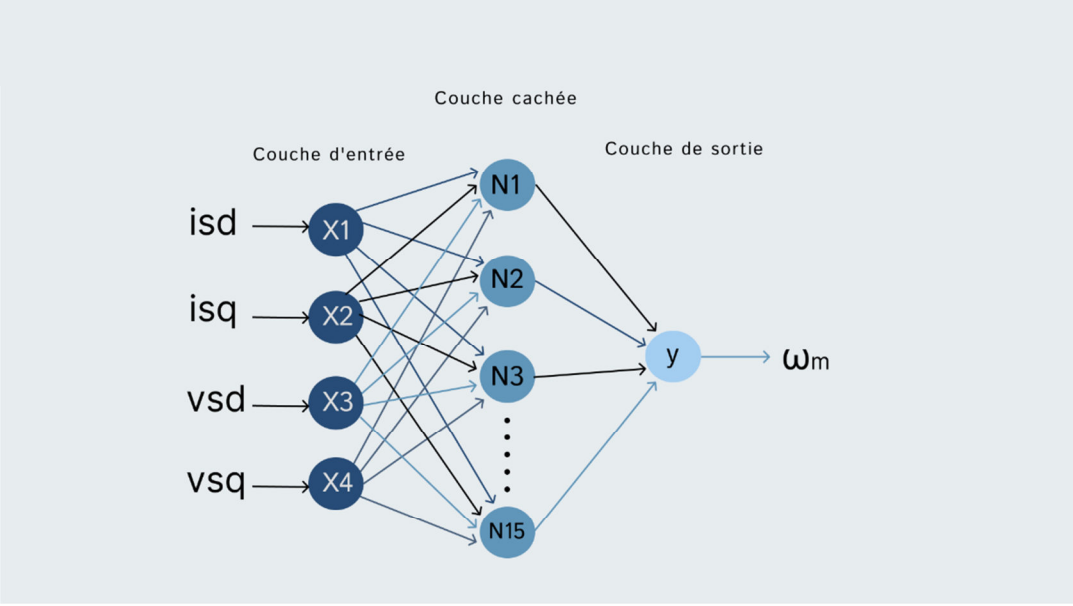

Figure 3.13 Architecture du réseau de neurone Cas 2.

- **Performance :**

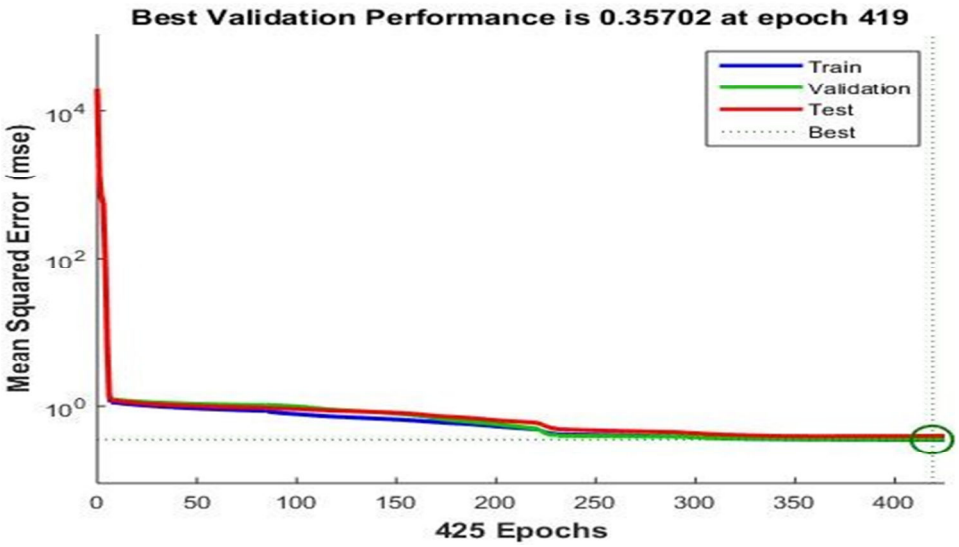

Figure 3.14 Performance de 2eme cas.

- **Régression :**

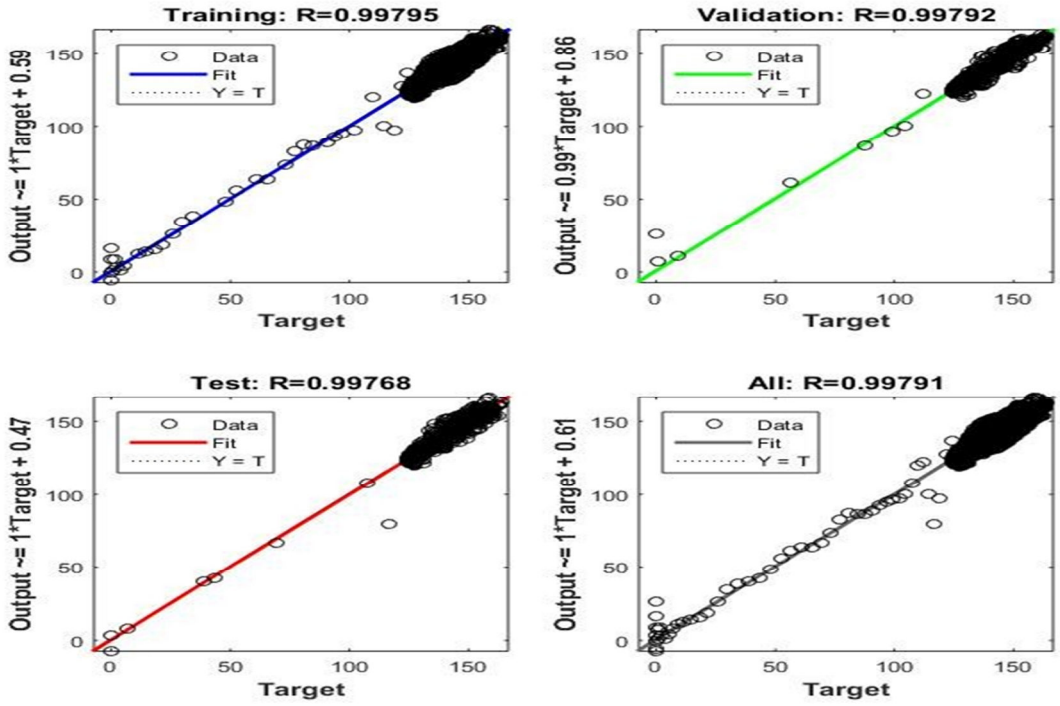

Figure 3.15 : Régression de 2eme cas

Tableau 3.5 : Comparaison des performances entre les deux cas.

Critère	Modèle 1	Modèle 2
Nombre d'entrée	3	4
Erreur quadratique Moyenne (MSE)	1.2361	0.35702
Coefficient de régression (R)	0.99114	0.99791

3.9 Conclusion

Dans ce chapitre, nous avons créé deux cas de RNA pour l'estimation de la vitesse du rotor.

Après avoir comparé plusieurs modèles, l'architecture MLP avec une couche cachée de 20 neurones a été choisie comme le meilleur candidat dans chaque cas.

Il apparaît que le modèle basé sur les courants triphasés est le mieux adapté à la réalité, car il donne une évaluation sans aucun écart critique par rapport à la vitesse réelle. L'approbation finale de ce choix sera présentée en détail au chapitre 5, par le biais d'une comparaison entre la simulation et l'estimation embarquée.

Chapitre 4 Simulation en condition réelle et implémentation embarquée (temps réel)

4.2 Introduction

Ce chapitre présente tout d'abord l'utilisation d'un banc d'essai expérimental pour la collecte des données réelles qui sont appliquées pour les phases d'apprentissage et de test du réseau de neurones pour l'estimation de la vitesse rotorique de la machine asynchrone. Par la suite, l'approche utilisée pour insérer sur un microcontrôleur, l'architecture adoptée pour le RNA, est présentée. L'objectif est de comparer l'exécution de l'architecture du RNA évaluée dans un environnement simulé et dans une exécution implantée en temps réel.

4.2 Simulation en conditions réelles

Dans cette partie, nous présentons le fonctionnement de la machine asynchrone dans un environnement réel à l'aide d'un banc d'essai expérimental. Cette partie a pour but de générer des signaux réels de courants, tensions et vitesse, qui seront utilisés pour comparer les performances du modèle embarqué développé dans la suite du chapitre.

Le travail a été réalisée à l'aide d'une maquette réelle de machine asynchrone, connectée à une plateforme de supervision et d'acquisition, à travers un système de prototypage rapide formé par dSpace/Controldesk.

4.2.1 Matériel utilisé

Dans cette partie, le matériel utilisé dans le banc d'essai et présenté.

- **La machine asynchrone :**

  C'est une MAS rebobinée avec des points de contact pour effectuer des tests de court-circuit de spires (Figure4.1). Les paramètres de cette MAS sont donnés par l'annexe1. Elle est reliée à un frein à poudre qui permet de simuler des couples résistants.

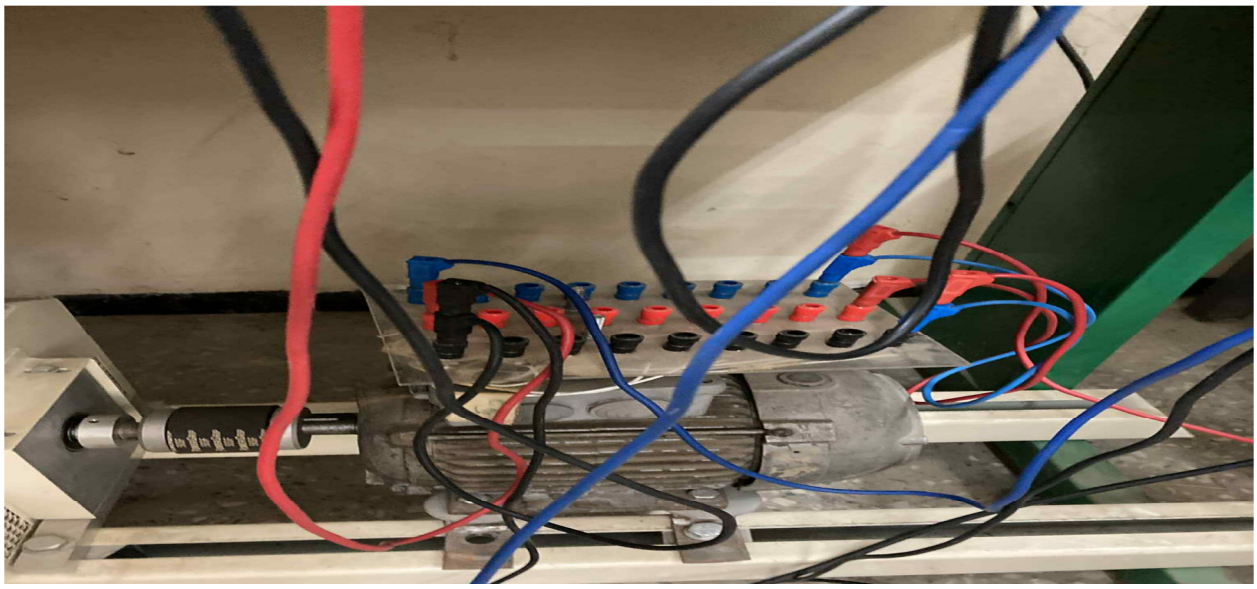

Figure 4.1 : Photo de la machine asynchrone.

- **Tension d'alimentation :**

  La source de tension triphasée est un autotransformateur triphasé variable (Figure4.2).

Figure 4.2 : Tension d'alimentation triphasée.

- **Capteur de courants et tensions :**

  La figure4.3 montre les capteurs de courant et de tension.

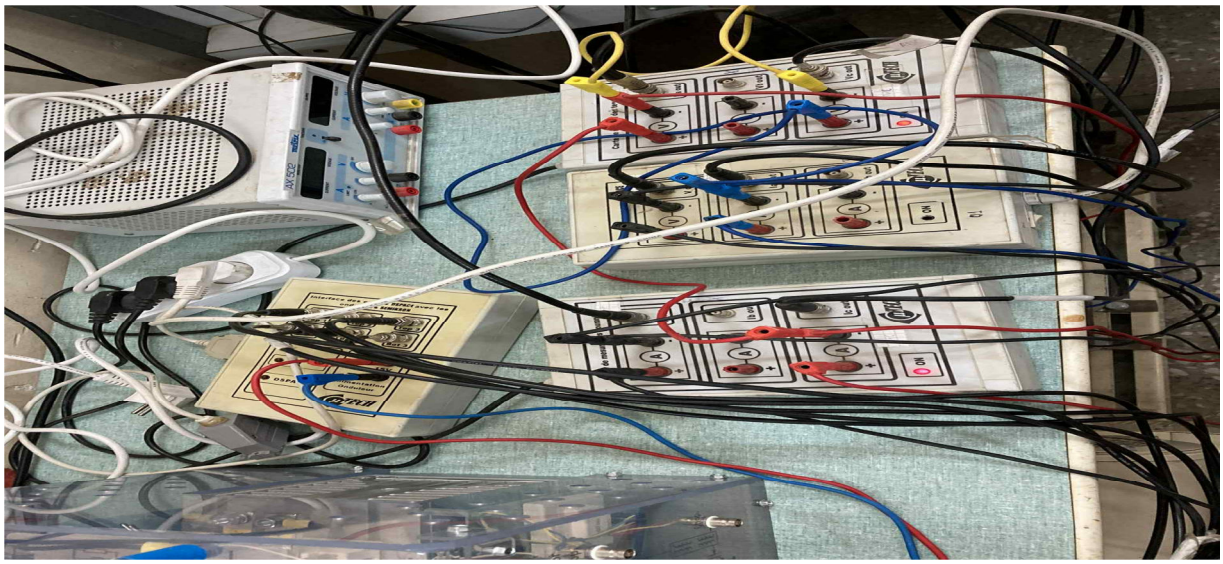

Figure 4.3 : Capteur de courants et de tensions.

4.2.2 Résultats obtenus

La simulation en conditions réelles a été réalisée à l'aide d'un banc de test permettant l'acquisition des grandeurs électriques en temps réel et leur visualisation via

l'interface **ControlDesk**. Ce système permet de réaliser un fonctionnement réel avec possibilité de variation de charge et d'autres paramètres. La figure4.5 illustre une acquisition en temps réel à l'aide de dSpace1104 et ControlDesk.

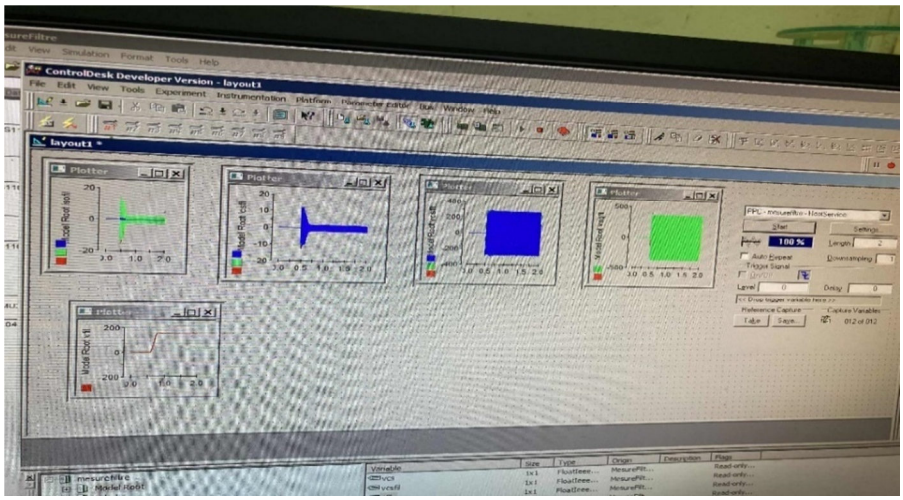

Figure 4.4 : Vue de l'interface ControlDesk.

Les figures (4.5, 4.6, 4.7) ci-dessous illustrent les différents signaux mesurés.

- Les courants statoriques :

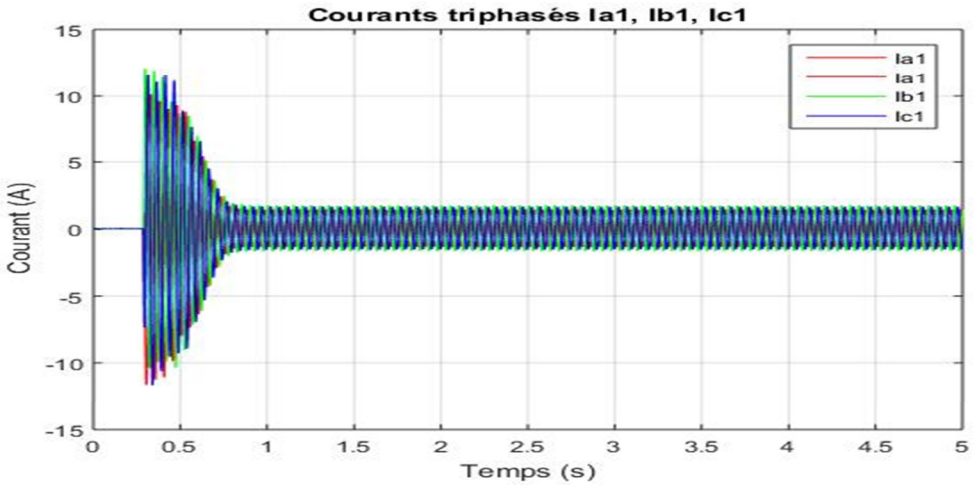

Figure 4.5 : Courant statorique i_{a1} i_{b1} i_{c1}.

- Les courants statorique (dq) :

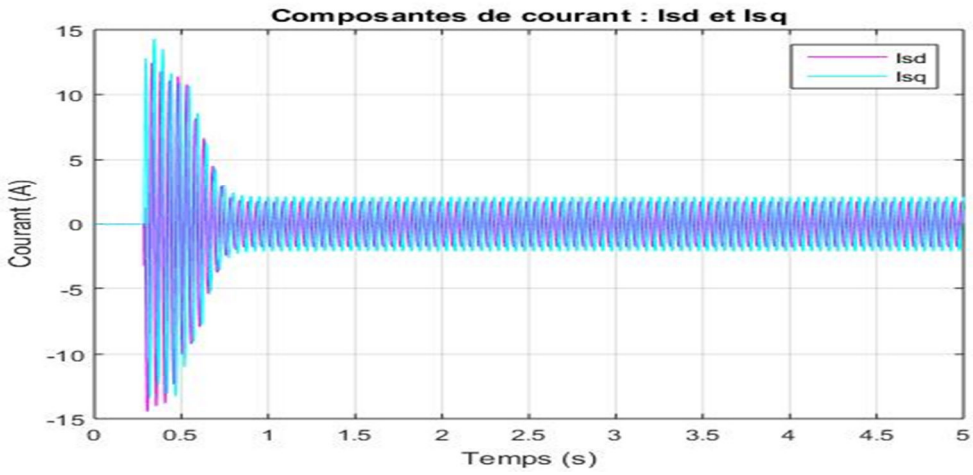

Figure 4.6 : Courant transformée i_{sd} i_{sq}.

- Tension d'alimentation (dq) :

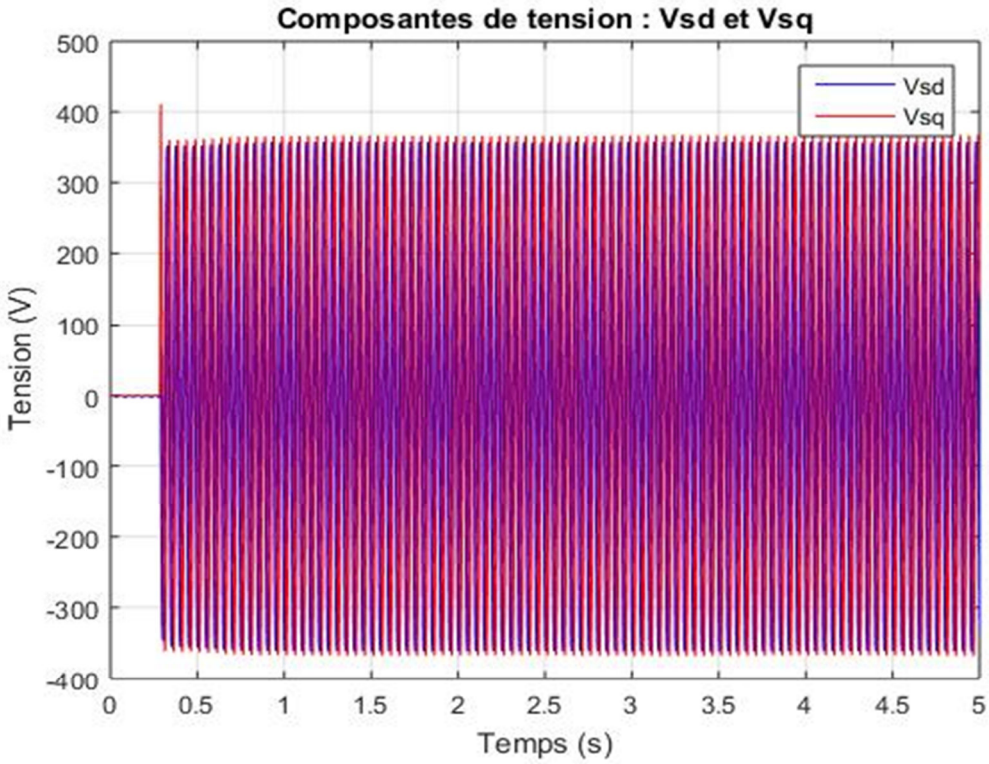

Figure 4.7 : Tension d'alimentation V_{sd} V_{sq}.

- La vitesse rotorique (ω_m) :

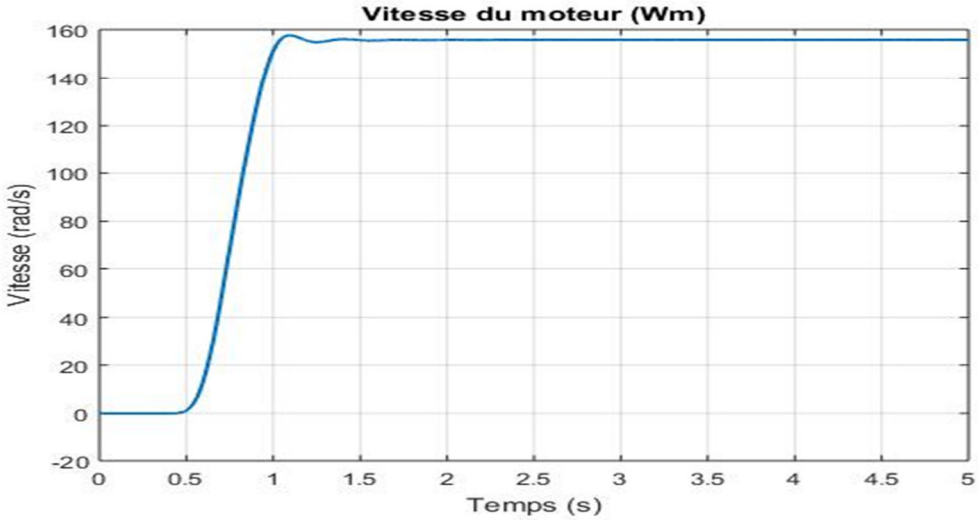

Figure 4.8 : Vitesse rotorique ω_m.

Ces signaux, mesurés en conditions réelles, serviront de base de comparaison avec les résultats obtenus via l'implémentation embarquée du modèle neuronal, présentée dans la section suivante.

4.3 Implémentation embarquée

L'implémentation embarquée des réseaux de neurones constitue une étape cruciale pour la validation en conditions réelles d'un modèle de traitement ou d'estimation. Elle permet de démontrer la faisabilité de l'exécution du modèle sur une cible matérielle contrainte, en termes de ressources mémoire, de puissance de calcul et de consommation énergétique. Cette phase représente souvent un compromis entre précision, efficacité, et contraintes matérielles.

4.3.1 Principes d'implémentation des réseaux neuronaux en environnement embarqué

Avant l'implantation d'un modèle de réseau neuronal dans un microcontrôleur, plusieurs stratégies peuvent être envisagées afin de garantir son efficacité sur le plan matériel :

- **Quantification des poids** : elle consiste à convertir les poids flottants du modèle en représentations à faible précision (souvent 8 ou 16 bits) pour réduire l'utilisation de la mémoire et améliorer la vitesse d'exécution, notamment sur des microcontrôleurs à architecture fixe.
- **Compression du modèle** : des techniques comme le **pruning** ou le **knowledge distillation** peuvent être utilisées pour réduire la taille du réseau sans altérer significativement ses performances.
- **Optimisation topologique** : cela inclut la simplification de l'architecture du réseau, le choix de fonctions d'activation simples à implémenter (comme **tanh**, **ReLU**, ou fonctions approchées), et la minimisation du nombre de couches et de neurones.

Toutefois, dans certains cas, lorsque le modèle initial est déjà compact et optimisé, ces étapes peuvent être jugées inutiles. Cela est notamment vrai pour les réseaux de petite taille qui peuvent être embarqués directement sans altération ni transformation.

4.3.2 Objectif de l'implémentation

L'objectif principal de cette phase est de valider le comportement du réseau neuronal dans un environnement embarqué, en intégrant le modèle d'estimation de vitesse directement sur un microcontrôleur de type **ESP8266**.

Dans notre cas, l'architecture du réseau neuronal choisi est particulièrement simple : elle se compose d'une seule couche cachée contenant 20 neurones, avec une fonction d'activation de type **tanh** (**tansig** pour Matlab). Cette structure légère ne nécessite ni compression, ni quantification, ni simplification supplémentaire pour être exécutée efficacement sur le microcontrôleur ciblé. L'intégration directe est ainsi rendue possible sans compromettre la précision ou la rapidité du modèle.

Étant donné que l'acquisition en temps réel des grandeurs physiques (notamment les courants I_a, I_b et I_c) n'a pas encore été mise en place à ce stade, les valeurs d'entrée

ont été injectées manuellement dans le système afin de tester le comportement de l'algorithme embarqué.

Dans les sections suivantes, des mesures précises de la vitesse d'exécution seront présentées afin de démontrer les performances du modèle intégré, mettant en évidence sa capacité à fonctionner en temps réel avec une latence très faible.

4.3.3 Cible matérielle utilisée

Le modèle a été implanté sur un ESP8266, est un microcontrôleur 32 bits à faible coût et à faible consommation, développé par **Espressif Systems**, intégrant une connectivité Wi-Fi. Il est largement utilisé dans les projets embarqués et IoT pour sa compacité, sa facilité de programmation via Arduino IDE, et sa capacité à exécuter des algorithmes simples en temps réel [33].

Tableau 4.1 : Caractéristiques techniques du microcontrôleur ESP8266.

Caractéristique	**Valeur**
Modèle	ESP8266 (NodeMCU)
Processeur	Tensilica L106 32-bit RISC
Fréquence d'horloge	80MHz (jusqu'à 160 MHz)
Mémoire RAM	80 Ko
Mémoire Flash	Jusqu'à 4 Mo
Tension d'alimentation	3.3V
Nombre d'entrées analogiques (ADC)	1 canal 10 bits
Interfaces de communication	UART, SPI, I2C, Wi-Fi
Environment de development	Arduino IDE / MicroPython / Lua
Languages supports	C / C++ / Python (via MicroPython)

4.3.4 Intégration du modèle neuronal

L'intégration du modèle neuronal dans un microcontrôleur nécessite une transposition efficace de l'architecture développée dans un environnement de haut niveau (MATLAB) vers un langage bas niveau adapté aux systèmes embarqués (comme le C++ dans l'environnement Arduino). Dans cette section, nous décrivons en détail le processus de transformation du réseau de neurones, initialement entraîné sous MATLAB, vers un code exécutable sur microcontrôleur.

a Génération du modèle à l'aide de MATLAB

Après l'entraînement du réseau de neurones dans MATLAB à l'aide de la Toolbox *Neural Network*, la fonction **genFunction** a été utilisée pour générer automatiquement une fonction MATLAB autonome contenant les poids, les biais, et les opérations de calcul du réseau. Cette fonction est structurée de manière à faciliter son interprétation manuelle et sa transcription vers un langage embarqué.

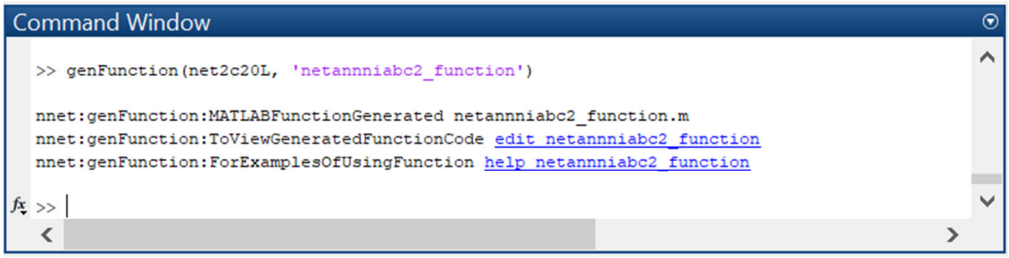

Figure 4.9 : MATLAB getFunction syntaxe.

Cette fonction génère une routine structurée comprenant les étapes de normalisation des entrées, propagation directe dans la couche cachée (avec tansig), propagation vers la couche de sortie (linéaire), puis dénormalisation de la sortie.

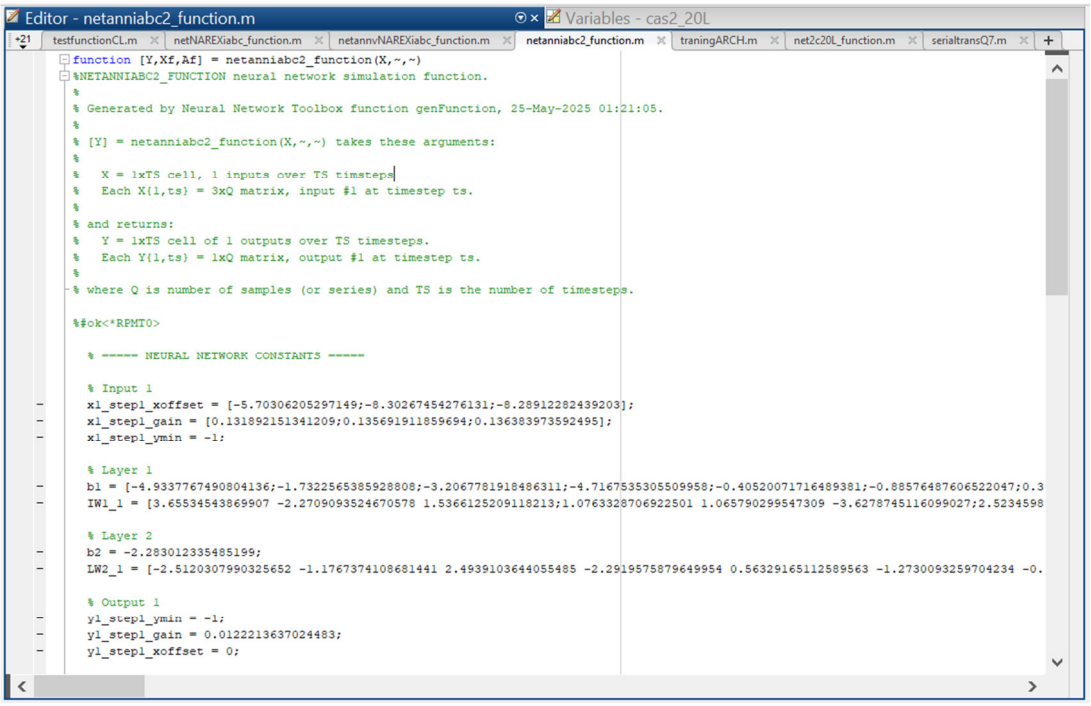

Figure 4.10 : exemple de contenu de script généré avec MATLAB getFunction.

b Normalisation des entrées

Dans MATLAB, les réseaux de neurones utilisent souvent une normalisation des données pour améliorer la stabilité numérique et la performance du réseau. Dans notre cas, les entrées sont normalisées selon la formule :

$$\boldsymbol{x_{norm,i}} = (\boldsymbol{x_i} - \boldsymbol{x_{offset,i}}) \times \boldsymbol{gain_i} + \boldsymbol{ymin} \qquad (4.1)$$

avec:

- $\boldsymbol{x_i}$: la i-ème composante de l'entrée brute,
- $\boldsymbol{x_{norm,i}}$: le décalage pour centrer les données,
- $\boldsymbol{gain_i}$: le facteur d'échelle,
- $\boldsymbol{ymin} = \boldsymbol{-1}$: la borne inférieure standard.

Cette opération est réalisée par la boucle suivante dans le code Arduino C++ :

```
16
17    float X_norm[3];
18    for (int i = 0; i < 3; i++) {
19      X_norm[i] = (input[i] - x_offset[i]) * gain[i] + ymin;
20    }
21
```

Figure 4.11 : la boucle de la normalisation.

c Propagation dans la couche cachée

La première couche du réseau comprend 20 neurones, chacun appliquant une fonction d'activation **tansig**. La sortie d'un neurone caché est donnée par :

$$a_j^{(1)} = tansig(\sum_{i=0}^{3} w_{j.i}^{(1)} \times \boldsymbol{x_{norm,i}} + b_j^{(1)}) \qquad (4.2)$$

Où :

- $w_{j.i}^{(1)}$: poids entre l'entrée iii et le neurone caché *j*.
- $b_j^{(1)}$: biais du neurone *j*.
- $\boldsymbol{x_{norm,i}}$: i -ème entrée normalisée.

En C++, ceci est codé de manière explicite :

```
50
51    float a1[20];
52    for (int i = 0; i < 20; i++) {
53      float sum = b1[i];
54      for (int j = 0; j < 3; j++) {
55        sum += IW1[i][j] * X_norm[j];
56      }
57      float e = Exp(-2.0f * sum);
58      a1[i] = 2.0f / (1.0f + e) - 1.0f;  // tansig
59    }
60
```

Figure 4.12 : la boucle de couche cachée.

d Approximation rapide de la fonction d'activation « tansig »

La fonction d'activation utilisée dans la couche cachée du réseau neuronal est la fonction **tansig**, une forme modifiée de la tangente hyperbolique, définie par :

$$tansig(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \qquad (4.3)$$

Cette fonction est largement utilisée dans les réseaux de neurones pour sa dérivabilité continue et sa sortie bornée dans l'intervalle $[-\mathbf{1}, \mathbf{1}]$ ce qui est idéal pour stabiliser l'apprentissage et le comportement du modèle.

Le calcul direct de l'exponentielle via la fonction standard $\boldsymbol{exp}()$ est coûteux en termes de cycles processeur et d'utilisation mémoire sur des microcontrôleurs embarqués comme l'ESP8266, Pour éviter l'utilisation coûteuse de la fonction $\boldsymbol{exp}()$ native, une approximation personnalisée a été utilisée pour calculer $\boldsymbol{e^x}$:

```
1
2   float Exp(float x) {
3     x = 1.0f + x / 256.0f;
4     x *= x; x *= x; x *= x; x *= x;
5     x *= x; x *= x; x *= x; x *= x;
6     return x;
7   }
8
```

Figure 4.13 : la fonction exponentielle.

L'astuce mathématique derrière cette fonction repose sur une **approximation par exponentiation rapide**, en utilisant l'équation suivante :

$$e^x \approx \left(1 + \frac{x}{n}\right)^n \qquad \text{lorsque } n \to \infty \qquad (4.4)$$

Ceci est connu comme la définition limite de l'exponentielle, issue du développement de **Taylor** :

$$e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \qquad (4.5)$$

Dans notre implémentation :

- On choisit n=256, donc :

$$e^x \approx \left(1+\frac{x}{n}\right)^{256} \qquad (4.6)$$

- Mais plutôt que d'effectuer 256 multiplications, on exploite l'équation :

$$\left(1+\frac{x}{n}\right)^{256} = \left[\left(1+\frac{x}{n}\right)^2\right]^8 \qquad (4.7)$$

D.1 VALIDITE DE L'APPROXIMATION

Cette approximation est suffisamment précise dans la plage restreinte des entrées du modèle, c'est-à-dire $x \in [-1,\ 1]$, car :

- Les entrées sont normalisées à l'avance via une transformation linéaire,
- Les valeurs de $-2x$ utilisées dans e^{-2x} sont donc également contenues dans une plage restreinte,
- Dans cette plage, la courbe de e^x est bien approximée par la formule limite, sans écarts importants.

e Couche de sortie

La couche de sortie est une couche linéaire avec un seul neurone. La sortie brute (avant dénormalisation) est donnée par :

$$a^{(2)} = \sum_{j=1}^{20} w_j^{(2)} \times a_j^{(1)} + b_j^{(2)} \qquad (4.8)$$

Et est directement codée comme suit :

```
71
72    float a2 = b2;
73    for (int i = 0; i < 20; i++) {
74      a2 += LW2[i] * a1[i];
75    }
76
```

Figure 4.14 : la couche de sortie.

f Dénormalisation de la sortie

La sortie du réseau est ensuite transformée dans l'échelle originale à l'aide de la formule inverse de la normalisation :

$$y = \frac{a^{(2)} - y_{min}}{y_{gain}} + y_{offset} \qquad (4.9)$$

En C++, ceci est codé de manière explicite :

```
76
77    // ===== Denormalisation =====
78    const float y_gain = 0.01222136;
79    const float y_offset = 0.0;
80    const float y_min = -1.0;
81    float y = ((a2 - y_min) / y_gain) + y_offset;
82
```

Figure 4.15: la partie de la dénormalisation.

Cette opération permet d'obtenir la valeur estimée de la vitesse du moteur en tr/min

4.3.5 Avantages de cette architecture embarquée

L'architecture retenue est particulièrement adaptée à une implémentation embarquée sans compression ni quantification, pour les raisons suivantes :

- Elle contient seulement une couche cachée avec 20 neurones, ce qui reste léger en termes de calcul.
- Les fonctions mathématiques utilisées sont simples et peuvent être optimisées manuellement.
- Le modèle a été directement transcrit sans perte de performance.
- La vitesse d'exécution est très rapide, comme démontré dans la section suivante (*cf. section 4.3.6*).

4.3.6 Évaluation des performances d'exécution

L'évaluation des performances d'exécution du réseau de neurones embarqué a été réalisée à l'aide d'un microcontrôleur ESP8266, avec un accent particulier sur le **temps d'exécution moyen** nécessaire pour traiter une entrée par le réseau.

a Structure des entrées

Pour simuler un environnement réel de fonctionnement, des vecteurs d'entrée ont été définis manuellement dans le tableau **inputData** 3x10. Chaque vecteur contient trois grandeurs physiques représentatives de l'état du moteur — à savoir :

- i_a : composante de phase A du courant statorique,
- i_b : composante de phase B du courant statorique,
- i_c : composante de phase C du courant statorique.

Exemple d'entrée :

```
86
87  // les donnee de test
88  float inputData[10][3] = {
89    {0.4427, 0.3112, 0.7280},
90    {0.1067, 0.5285, 0.9037},
91    {0.9619, 0.1656, 0.5738},
92    {0.0046, 0.6020, 0.5440},
93    {0.7749, 0.2630, 0.5150},
94    {0.8173, 0.6541, 0.4660},
95    {2.1555, 1.191, 0.96447},
96    {-1.6607, -0.36475, 2.0254},
97    {-1.6607, 0.1067, 0.1948},
98    {-1.4746, -0.62859, 2.1032}
99  };
100
```

Figure 4.16: les données de test.

Ces valeurs sont saisies manuellement pour tester des scénarios spécifiques, permettant une validation ciblée du modèle dans des conditions réalistes. Ce mode de saisie est utile en phase de test mais ne convient pas à une utilisation temps réel finalisée, où les données doivent être acquises dynamiquement depuis des capteurs.

b Mesure du temps d'exécution

Pour mesurer précisément le temps requis pour une exécution complète du réseau (de la normalisation d'entrée jusqu'à la sortie dénormalisée), le code utilise les instructions suivantes :

```
104
105  uint32_t startCycle =0;
106  uint32_t CycleCount;
107  uint32_t elapsed = 0;
108  uint32_t totalduration = 0;
109    for (int i = 0; i < 10; i++) {
110      startCycle = ESP.getCycleCount();  // obtenir le numéro de cycle avant de démarrer l'exécution
111      float result = runNN(inputData[i]); // démarrage l'exécution de la fonction
112      CycleCount = ESP.getCycleCount() - startCycle; // Calculer le nombre de cycles utilisés pour l'exécution
113      float temp = CycleCount / (float)ESP.getCpuFreqMHz(); // Calculer le temp de exécution en us
114      Serial.print("Output "); Serial.print(i); Serial.print(": ");
115      Serial.println(result, 4);
116      Serial.print("1 cycle time (us): "); Serial.println(temp, 2);
117      totalduration = totalduration + temp;  // Calculer le temp total pour 10 exécutions
118    }
119
120    Serial.print("Average time per run (us): ");
121    Serial.println(totalduration / 10); //Calculer le temp moyenne pour 1 exécution
122  }
123
```

Figure 4.17 : Mesure du temps d'exécution.

- **ESP.getCycleCount()** retourne le nombre de cycles d'horloge CPU depuis le démarrage du microcontrôleur.
- La différence entre les cycles avant et après l'exécution donne une mesure très précise du nombre de cycles consommés.
- Cette valeur est ensuite convertie en microsecondes **(µs)** selon la formule suivante :

$$Temps\ d'exécution = \frac{N_{cycles}}{f_{CPU}} \qquad (4.10)$$

Où :

- N_{cycles} : est le nombre total de cycles effectués pendant l'exécution du réseau,
- f_{CPU} : est la fréquence du processeur en MHz (par exemple, 80 MHz),
- Le résultat donne le temps d'exécution réel en microsecondes.

Cette méthode est bien adaptée aux tests de performance embarqués, car elle permet une évaluation fine et non intrusive du temps réel de calcul. C'est un aspect

crucial dans les systèmes embarqués temps réel, où le respect des délais de réponse est impératif.

- **Exemple de sortie**

L'exécution imprime dans le moniteur série la sortie du réseau et le temps d'exécution associé, comme illustré ci-dessous :

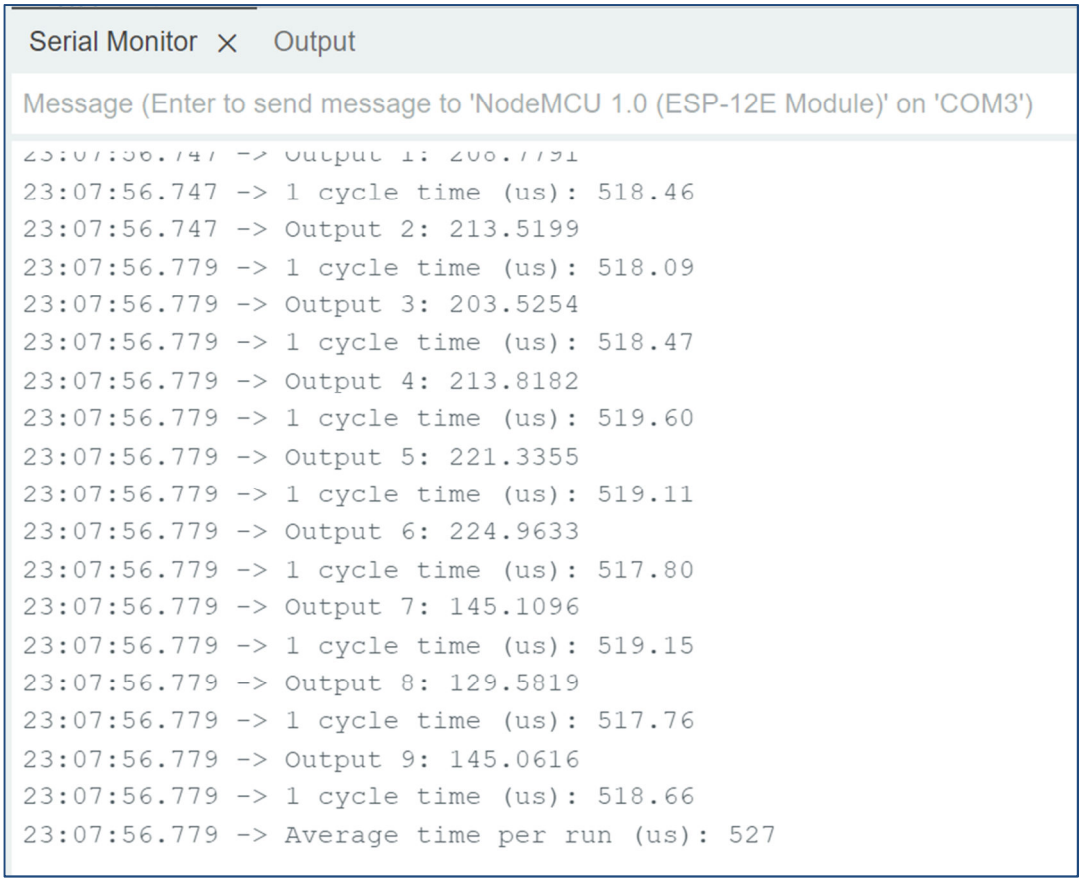

Figure 4.18 : les résultats imprimés sur le moniteur série.

La **figure 4.18** montre les résultats imprimés sur le moniteur série. On observe que chaque exécution du réseau de neurones prend environ **518 à 520 µs**, avec un **temps moyen de 527 µs** par cycle. Cette vitesse d'exécution est considérée comme **très bonne** pour les systèmes embarqués en temps réel, notamment dans les applications de contrôle de moteurs, où la fréquence de mise à jour est souvent de 1 kHz (soit une

période de 1000 µs). Le réseau neuronal est donc suffisamment rapide pour fonctionner en ligne dans une boucle de commande sans ralentir le système.

4.4 Conclusion

Ce chapitre a présenté l'intégration du modèle neuronal d'estimation de vitesse sur un microcontrôleur ESP8266. Grâce à la simplicité du réseau (une seule couche cachée), l'implémentation directe a été possible sans optimisation supplémentaire. Les tests effectués avec des données simulées ont montré un temps d'exécution moyen de 527 µs, confirmant la compatibilité du modèle avec les exigences d'un système temps réel. Ces résultats valident la pertinence de la solution pour des applications embarquées de commande de MAS sans capteur.

Chapitre 5 Résultats et comparaison.

5.1 Introduction

Ce chapitre présente les résultats obtenus à travers les différentes étapes du projet. L'objectif est de comparer les estimations de la vitesse rotorique fournies par les réseaux de neurones dans différents contextes : en simulation MATLAB et en implémentation embarquée. Quatre comparaisons principales sont abordées afin d'évaluer la précision, la cohérence temporelle et la robustesse des estimations par rapport à la vitesse réelle simulée.

5.2 Résultats et comparaison

Dans cette partie, nous présentons et analysons les résultats obtenus à travers les différentes étapes de développement du système d'estimation. L'analyse commence par les résultats issus des simulations MATLAB, qui constituent une référence pour l'évaluation initiale du modèle. Afin de faciliter la compréhension des courbes et de contextualiser les réponses du système, nous introduisons tout d'abord le couple de charge utilisé dans les tests avec la simulation **Figure 5.1**. Cette étape permet d'interpréter correctement le comportement du modèle, d'évaluer la qualité de l'estimation, et de comparer objectivement les sorties du réseau neuronal à la vitesse simulée.

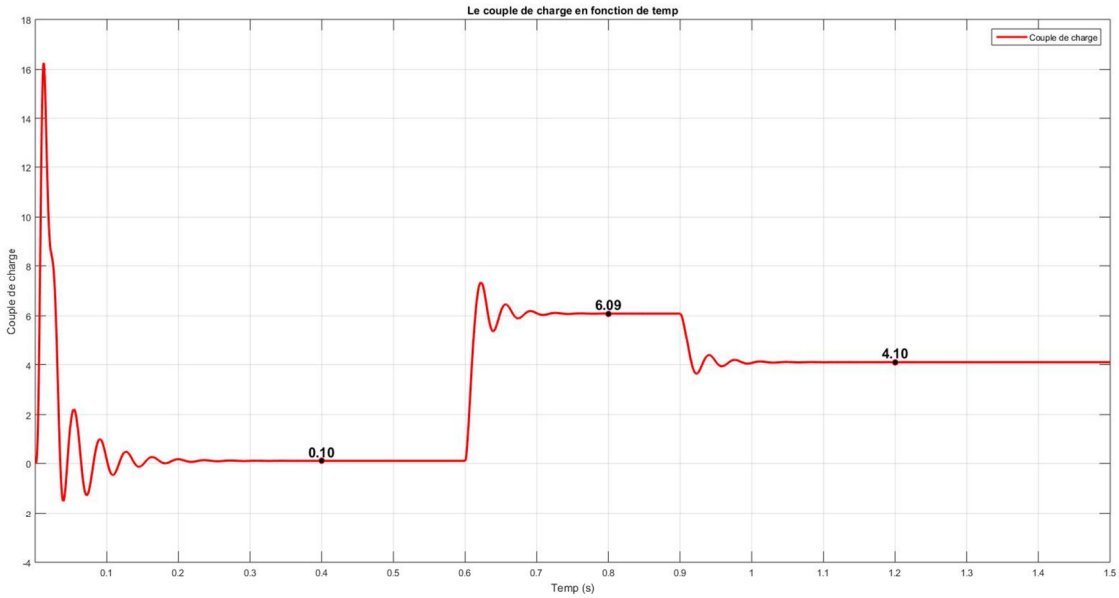

Figure 5.1 : Le couple de charge utilisé dans la simulation.

5.2.1 Estimation du premier modèle par rapport à la vitesse simulée en MATLAB

Cette section présente la comparaison entre la vitesse estimée par le premier modèle neuronal et la vitesse simulée sous MATLAB. La **figure 5.2** rappelle l'architecture du modèle utilisé dans cette étape.

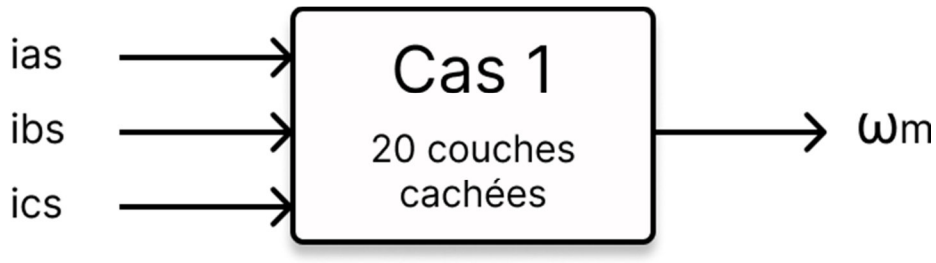

Figure 5.2 : l'architecture de premier réseau.

- **En régime permanent :**

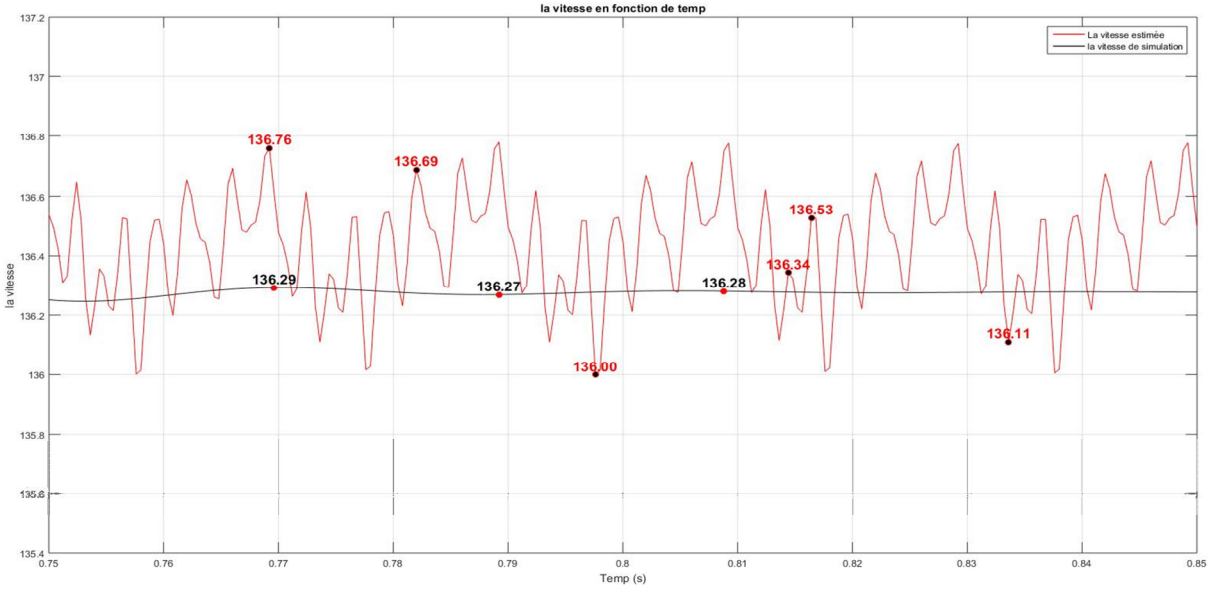

Figure 5.3 : Réponse en régime permanent (Cas 1 – simulation MATLAB).

- **En régime transitoire :**

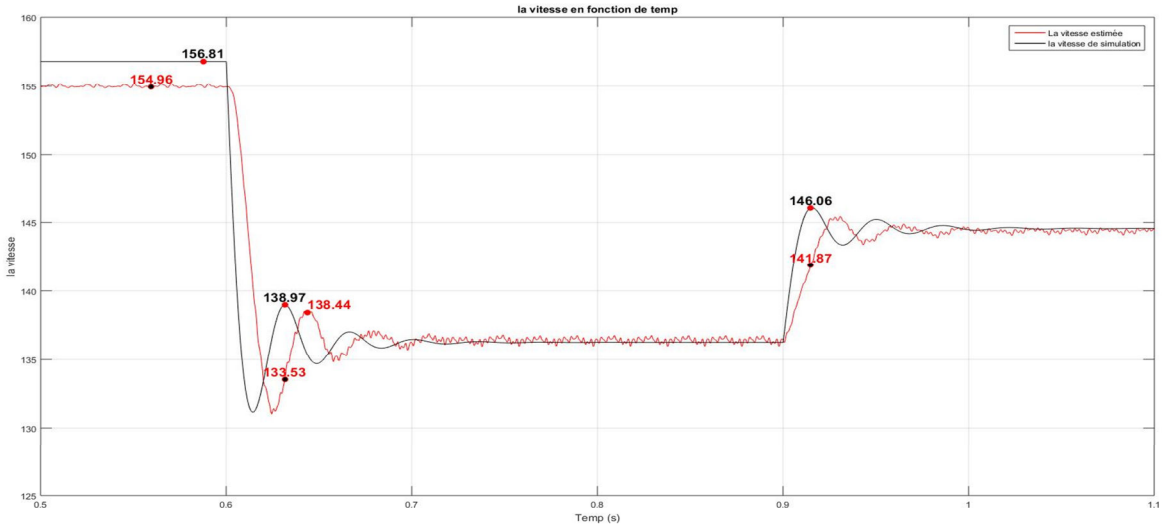

Figure 5.4 : Réponse en régime transitoire (Cas 1 – simulation MATLAB).

5.2.2 Estimation du deuxième modèle par rapport à la vitesse simulée en MATLAB

Cette section compare la vitesse estimée par le deuxième modèle neuronal à la vitesse obtenue par simulation sous MATLAB. La **figure 5.5** illustre brièvement la structure du second modèle utilisé pour cette comparaison.

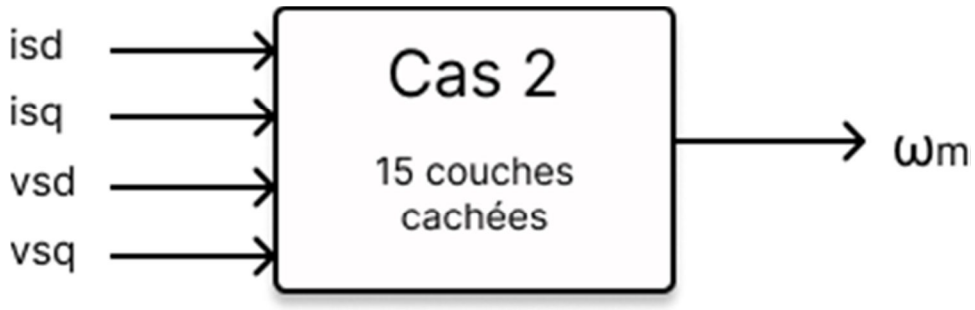

Figure 5.5 : l'architecture de deuxième réseau.

- **En régime permanent :**

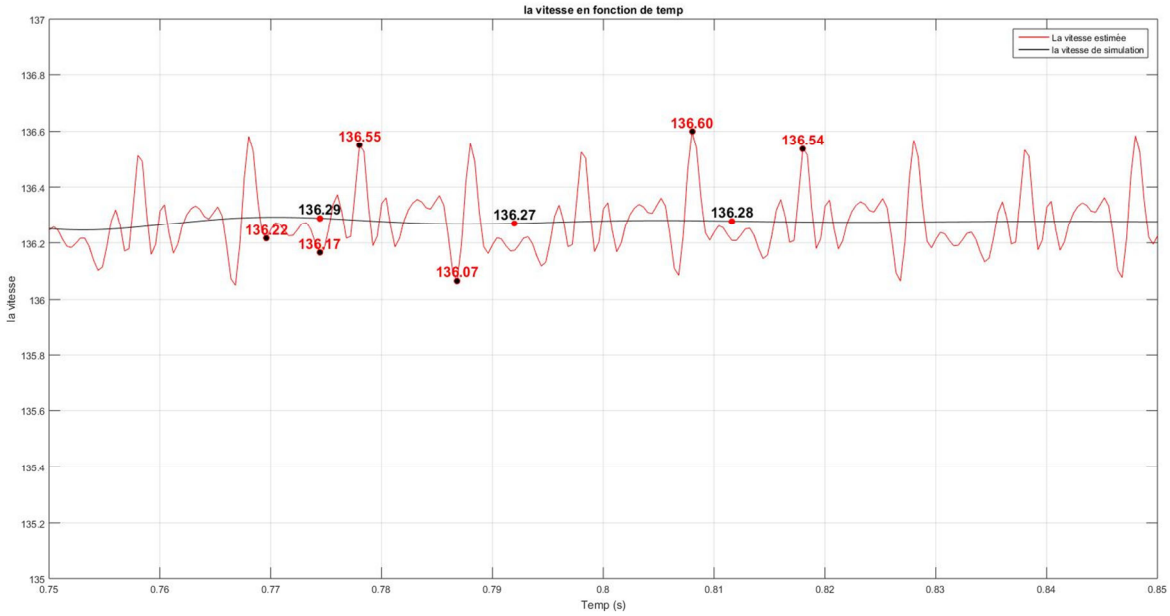

Figure 5.6 : Réponse en régime permanent (Cas 2 – simulation MATLAB).

- **En régime transitoire :**

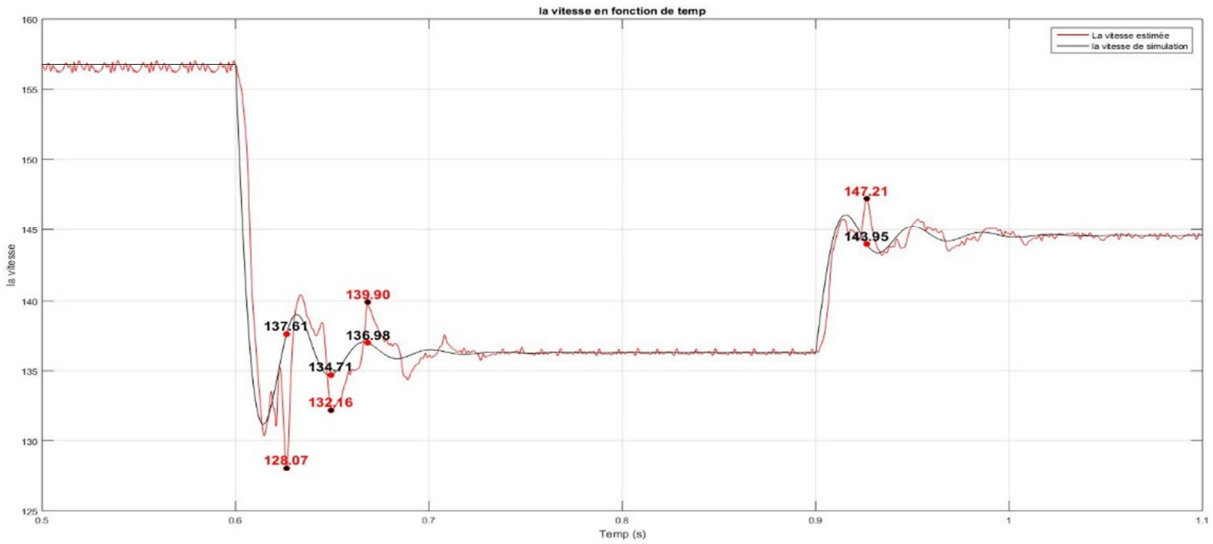

Figure 5.7 : Réponse en régime transitoire (Cas 2 – simulation MATLAB).

- **Bilan comparatif des résultats**

L'analyse des résultats en simulation montre que le modèle basé sur les courants triphasés (Cas 1) présente une estimation plus stable et mieux synchronisée avec la vitesse de référence simulée, en particulier lors des phases transitoires. Bien que le second modèle (Cas 2), utilisant les composantes **dq** des courants et tensions, affiche une erreur quadratique moyenne plus faible, il présente néanmoins un décalage temporel significatif, notamment lors des variations de couple.

Ainsi, en se basant uniquement sur les résultats MATLAB, Cas 2 semble plus précis numériquement, mais Cas 1 offre une meilleure cohérence dynamique, ce qui est crucial pour une implémentation pratique en temps réel.

5.2.3 Estimation du premier modèle par rapport à la vitesse mesurée

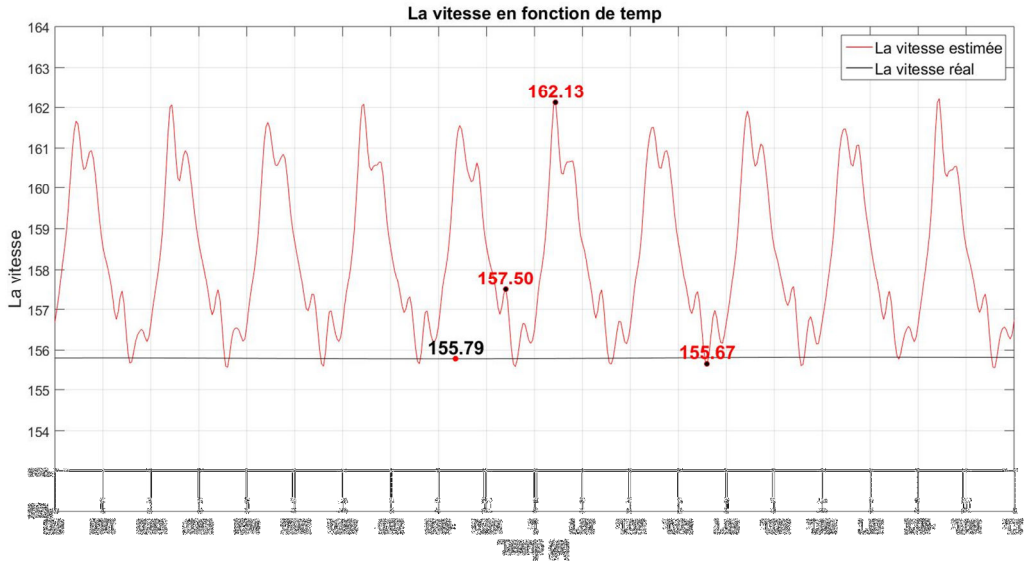

Figure 5.8 : Estimation de la vitesse Cas 1comparée à la vitesse réelle.

5.2.4 Estimation du deuxième modèle par rapport à la vitesse mesurée

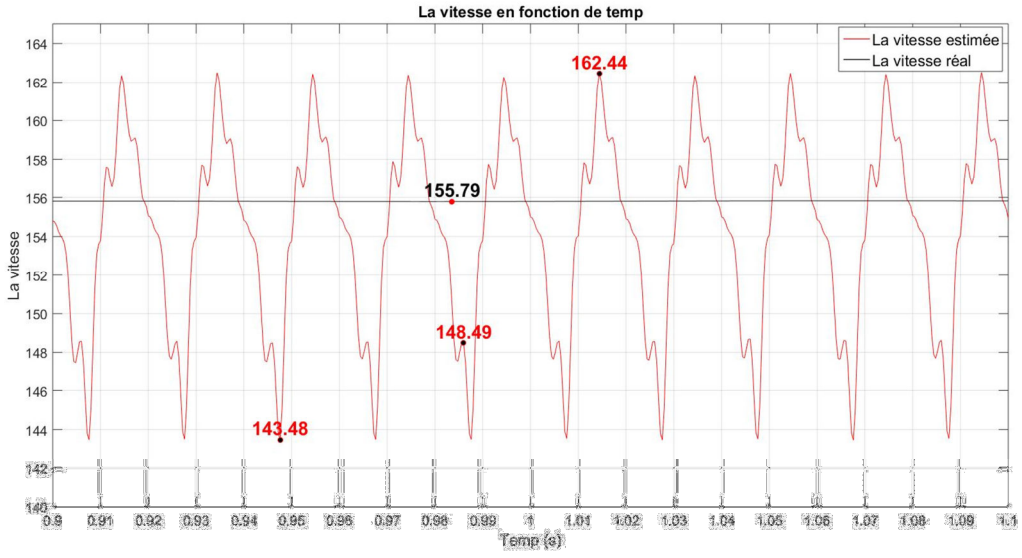

Figure 5.9 : Estimation de la vitesse Cas 2 comparée à la vitesse réelle.

- **Comparaison des modèles avec la réalité**

Bien que le modèle 2 ait présenté de meilleures performances numériques en simulation (erreur plus faible, régression plus élevée), son estimation s'éloigne nettement de la vitesse réelle obtenue en conditions semi-expérimentales. Cette divergence s'explique principalement par le fait que les tensions simulées utilisées comme entrées dans ce modèle ne correspondent pas fidèlement aux tensions réellement présentes dans le système. Cette différence impacte la précision de l'estimation et introduit un décalage temporel significatif.

En revanche, le modèle 1, fondé sur les courants triphasés i_a i_b i_c, fournit une estimation beaucoup plus cohérente avec la réalité. Les courants simulés sont en effet plus proches des courants mesurés dans le système réel, ce qui rend le modèle plus robuste aux variations physiques. Ce résultat confirme que le modèle 1 est le plus adapté à une implémentation embarquée fiable, en particulier dans une approche sans capteur.

5.2.5 Test du modèle embarqué avec des données réelles

Dans cette section, nous testons le modèle neuronal embarqué sur l'ESP8266 à l'aide de **données réelles** mesurées en laboratoire. Contrairement aux tests précédents utilisant des vecteurs simulés, les entrées utilisées ici proviennent de **mesures expérimentales** de courants triphasés i_a i_b i_c enregistrées dans MATLAB à partir d'un banc essai expérimental.

a Objectif et méthodologie

L'objectif est de valider le fonctionnement du modèle neuronal embarqué en traitant des données expérimentales capturées en laboratoire. Les vecteurs d'entrée sont extraits depuis MATLAB, puis transmis à l'ESP8266 via une liaison série. Le traitement est effectué localement par le microcontrôleur, et l'estimation du modèle est ensuite renvoyée à MATLAB pour affichage et analyse.

Ce test, réalisé en mode **hors ligne,** ne constitue pas un fonctionnement en temps réel, mais permet d'évaluer la précision et la cohérence des estimations du modèle embarqué en le comparant aux résultats attendus ou simulés.

b Envoi des données depuis MATLAB

Le code MATLAB lit les vecteurs i_a , i_b , i_c depuis l'espace de travail, les convertit en single (format flottant 32 bits), puis les regroupe sous forme d'une matrice 3×N, où N est le nombre d'échantillons.

Chaque vecteur colonne (représentant un instant de mesure) est envoyé **en binaire brut** à l'ESP8266 via un port série à 115200 bauds. Voici extrait clé du code :

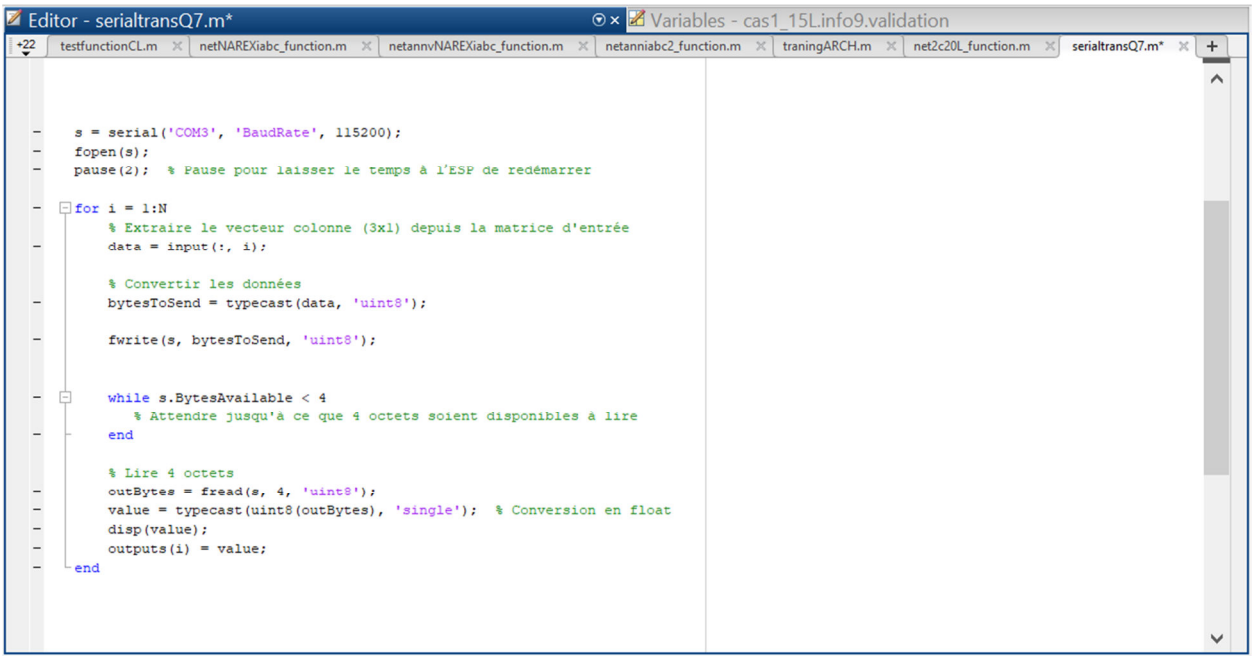

Figure 5.10 : code Matlab pour la communication série.

Ce protocole permet une transmission rapide et efficace, sans conversion texte, en minimisant la latence.

c Traitement et réponse côté ESP8266

Le microcontrôleur ESP8266 attend en boucle la réception de **12 octets** correspondant à un vecteur d'entrée :

```
94
95    static byte buf[12];  // 3 flottants × 4 octets
96    if (Serial.available() >= 12) {
97      Serial.readBytes(buf, 12);  // Lire les 12 octets
98      float input[3];
99      memcpy(input, buf, 12);     //  byte-to-float conversion
100
101     float result = runNN(input);
102
103     // Renvoyer le résultat sous forme flottant (4 octets)
104     Serial.write((byte*)&result, 4);
105   }
106
```

Figure 5.11 : code Matlab pour la communication série.

Le calcul de la sortie est immédiat. Une fois le vecteur reçu, la prédiction du réseau est renvoyée sous forme d'un flottant binaire (4 octets), assurant une réponse rapide adaptée à une architecture temps réel.

5.2.6 Résultat de la comparaison entre le modèle embarqué et l'architecture MATLAB

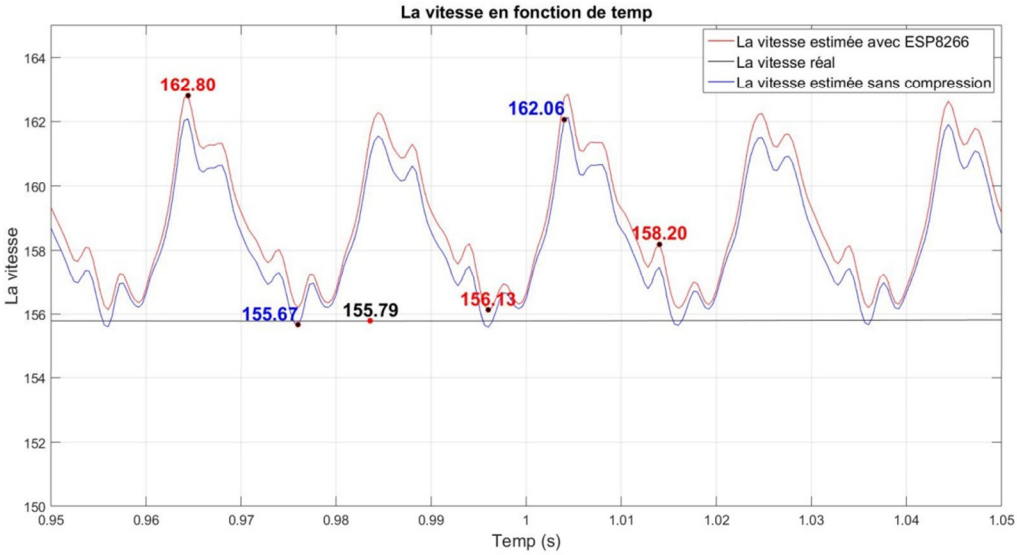

Figure 5.12 : Comparaison entre estimation embarquée et estimation MATLAB cas1.

5.2.7 Résultat embarqué comparé à la vitesse simulée dans ControlDesk

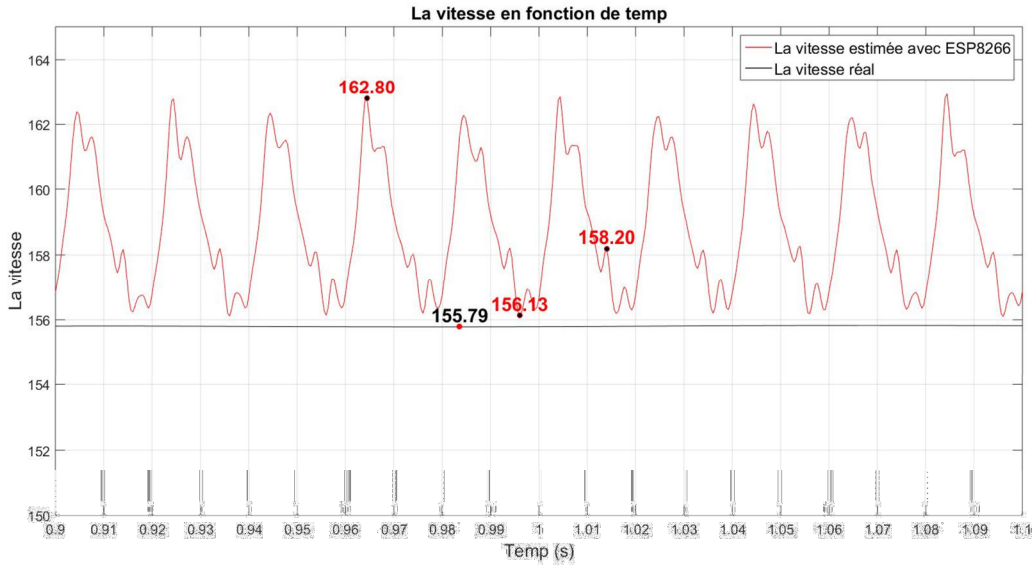

Figure 5.13 : Comparaison entre estimation embarquée et vitesse réel.

- **Comparaison entre estimation embarquée et vitesse réelle**

La **Figure 5.6** illustre la comparaison entre la vitesse estimée par le modèle embarqué (implémenté sur le microcontrôleur ESP8266) et la vitesse de référence simulée, représentant le comportement réel de la machine.

On observe que la courbe estimée suit globalement la forme dynamique de la vitesse réelle, avec une bonne correspondance dans les phases stationnaires. En régime transitoire, de légers écarts apparaissent, notamment lors des changements rapides de charge. Cela est dû à la limitation en précision numérique et au temps de traitement réduit sur le microcontrôleur, qui n'égale pas la précision d'une simulation MATLAB.

Toutefois, aucun décalage majeur n'est constaté, ce qui montre que le modèle embarqué conserve la structure du comportement réel malgré les ressources limitées. L'estimation est stable, réactive et exploitée en temps réel, validant ainsi le transfert réussi du réseau neuronal vers un environnement embarqué.

5.3 Conclusion

L'ensemble des résultats comparatifs montre que le modèle utilisant les courants triphasés constitue une solution efficace et réaliste pour l'estimation de la vitesse rotorique. Le modèle embarqué offre une estimation satisfaisante et suffisamment fidèle pour une application sans capteur. La cohérence globale avec la vitesse réelle confirme la faisabilité pratique de cette solution dans un système embarqué à faible coût.

Conclusion générale

L'objectif principal de ce travail était de développer une méthode d'estimation de la vitesse rotorique d'une machine asynchrone triphasée sans capteur, en utilisant un réseau de neurones artificiels, puis de valider cette approche dans un système embarqué temps réel.

Dans un premier temps, une modélisation mathématique de la machine asynchrone a été réalisée, suivie d'une simulation complète sous MATLAB/Simulink, permettant de générer les données nécessaires à l'entraînement. Deux configurations de réseau de neurones ont ensuite été conçues, en utilisant différentes entrées physiques (courants triphasés d'une part, composantes dq d'autre part), afin de comparer leurs performances respectives.

Après entraînement, évaluation et validation des modèles, le réseau le plus cohérent avec la réalité dynamique celui basé sur les courants triphasés a été retenu pour une implémentation embarquée sur microcontrôleur ESP8266. Cette phase a permis de vérifier le fonctionnement du modèle en conditions proches du réel, et de confirmer sa capacité à fournir une estimation rapide et stable de la vitesse.

Les résultats obtenus montrent que cette approche est fiable, efficace et applicable pour des systèmes industriels nécessitant une commande sans capteur. Le modèle embarqué a démontré une bonne précision et une exécution fluide, validant ainsi l'intégration d'une solution d'intelligence artificielle dans un environnement matériel contraint.

Annexes

Annexe 1 : Les paramètres de la machine asynchrone triphasé.

```
Rr=13.6324;
Rs=13.3072;
Lr=0.678;
Ls=0.678;
Lm=0.638;
J=0.00177007;
fv=0.000643777;
p=2;
fs=0;
```

Annexe 2 : La fonction de réseau neuronal développée pour le ESP8266

```
float Exp(float x) {
  x = 1.0f + x / 256.0f;
  x *= x; x *= x; x *= x; x *= x;
  x *= x; x *= x; x *= x; x *= x;
  return x;
}
float runNN(float input[3]) {
  // ===== Normalisation =====
  const float x_offset[3] = {-5.70306205, -8.30267454, -8.28912282};
  const float gain[3]     = {0.13189215, 0.13569191, 0.13638397};
  const float ymin        = -1.0;
  float X_norm[3];
  for (int i = 0; i < 3; i++) {
    X_norm[i] = (input[i] - x_offset[i]) * gain[i] + ymin;
  }
```

```
// ===== 1er couche =====
const float b1[20] = {
  -4.9337767, -1.7322565, -3.2067782, -4.7167535, -0.4052007,
  -0.8857649, 0.3639111, -1.7891679, -0.0125886, 1.4456788,
  0.0278048, 0.9134097, 0.5884083, 1.5772399, 0.1320413,
  -1.6724678, 3.7058876, 1.4490888, -1.6808804, 5.4108006
};
const float IW1[20][3] = {
  {3.6553454, -2.2709094, 1.5366125}, {1.0763329, 1.0657903, -3.6278745},
  {2.5234599, 1.7741113, 2.2483134}, {3.0732668, 3.5280629, -2.3400300},
  {1.8780342, -3.8086612, 0.0142280}, {-3.9612678, 0.4315471, -2.5388569},
  {-3.1043652, -0.1715602, 1.7474831}, {0.1784960, -2.2596928, -3.9069270},
  {1.3632571, -4.9828976, 2.2059661}, {-5.3154440, 2.4451823, 2.9398291},
  {-1.6766763, -1.9344430, 1.5711599}, {3.2424679, -4.4889646, -2.2622263},
  {0.5017951, -3.5032607, 4.0072530}, {3.6111115, 1.2548095, -2.8640282},
  {0.2631875, -1.5711427, -0.2250829}, {-2.9670131, 1.6486525, 2.4771507},
  {3.4812371, 3.8712234, -0.8717179}, {-0.0911990, -1.2335741, -1.7579183},
  {-1.0567777, -2.1030214, 2.6072457}, {1.8141728, 1.3105465, 2.3802458}
};
float a1[20];
for (int i = 0; i < 20; i++) {
  float sum = b1[i];
  for (int j = 0; j < 3; j++) {
    sum += IW1[i][j] * X_norm[j];
  }
```

```
  // ===== 2eme couche =====
  const float b2 = -2.2830123;
  const float LW2[20] = {
    -2.5120308, -1.1767374, 2.4939104, -2.2919576, 0.5632917,
    -1.2730093, -0.9763373, 2.2403413, -0.1538493, 0.0975268,
    1.6769646, -0.1797080, 0.1639917, 0.3039423, -5.8111450,
    -0.4021917, -1.1671494, 4.4953499, 0.8793764, -0.5861087
  };

  float a2 = b2;
  for (int i = 0; i < 20; i++) {
    a2 += LW2[i] * a1[i];
  }

  // ===== Denormalisation =====
  const float y_gain = 0.01222136;
  const float y_offset = 0.0;
  const float y_min = -1.0;
  float y = ((a2 - y_min) / y_gain) + y_offset;

  return y;
}
```

Annexe 3 : le code de ESP8266 utilisé pour l'exécution de fonction et la communication série

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  static byte buf[12];  // 3 flottants × 4 octets
  if (Serial.available() >= 12) {
    Serial.readBytes(buf, 12);  // Lire les 12 octets
    float input[3];
    memcpy(input, buf, 12);     //  byte-to-float conversion
    float result = runNN(input);
    // Renvoyer le résultat sous forme flottant (4 octets)
    Serial.write((byte*)&result, 4);
  }
}
```

Annexe 4 : le code MATLAB utilise pour la communication série

```
input = single([ias'; ibs'; ics']);
N = size(input, 2);
outputs = zeros(1, N, 'single');


s = serial('COM3', 'BaudRate', 115200);
fopen(s);
pause(2);  % Pause pour laisser le temps à l'ESP de redémarrer


for i = 1:N
    % Extraire le vecteur colonne (3x1) depuis la matrice d'entrée
    data = input(:, i);
    % Convertir les données
    bytesToSend = typecast(data, 'uint8');
    fwrite(s, bytesToSend, 'uint8');
    while s.BytesAvailable < 4
       % Attendre jusqu'à ce que 4 octets soient disponibles à lire
    end
    % Lire 4 octets
    outBytes = fread(s, 4, 'uint8');
    value = typecast(uint8(outBytes), 'single');  % Conversion en float
    disp(value);
    outputs(i) = value;
end
```

Bibliographie

[1] Zouggar El Walid. **« Titre »**
MEMOIRE Présenté en vue de l'obtention du diplôme de MAGISTER\ BADJI MOKHTAR- Annaba, 2008

[2] Laichour Naima, « Modélisation et commande de la vitesse d'un moteur asynchrone triphasé piloté par un onduleur à MLI », Mémoire de Master, Université Saad Dahlab de Blida, 2022.

[3] B. Chermat, « Diagnostic des défauts de MAS par les méthodes paramétriques de traitement du signal », Mémoire de Master, Université Mohamed Khider Biskra, 2012.

[4] Département d'Automatique – UMMTO : « Chapitre 3 : Moteur asynchrone », support de cours en ligne, Université Mouloud Mammeri de Tizi-Ouzou, disponible à l'adresse : https://ummto-automation.github.io/homepage/modules/actuators/Chap_3-Moteur_asynchrone.pdf

[5] Cherier.F, Amade. G, «Modélisation en vue du diagnostic des défauts dans une machine asynchrone » mémoire d'Ingénieur d'Etat, Université M'hamed Bougara-Boumerdès, 2009.

[6] O. Ondel, « Diagnostic par reconnaissance des formes : application à un ensemble convertisseur - machine asynchrone » Thèse de Doctorat, Ecole Centrale de Lyon, 2006.

[7] B. Kerroum, « Etude et Diagnostic de mauvais branchement d'un moteur à induction» Mémoire de Master, Université Badji Mokhtar, Annaba, 2017.

[8] Hakima Cherif, « détection des défauts statorique et rotorique dans la machine asynchrone en utilisent l'analyse par FFT et ondelettes » mémoire de magister,

Université Mohamed Khider, Biskra, 2014.

[9] Franck Weinbissinger : « L'alimentation électrique pour les moteurs asynchrones », Socomec, 2010.

[10] N.Khalil «Contribution à la Commande de la Machine Asynchrone par DTC et logique floue» Thèse de magister, université de Constantine, 2006

[11] Benaissa Malika : « Minimisation des pulsations du couple dans une commande directe du couple "DTC" d'une machine asynchrone », Thèse de Doctorat, Université de Batna.

[12] Bennoui Hassina : « Apport de la logique floue et des réseaux de neurones pour la commande avec minimisation des pertes de la machine asynchrone », Thèse de Doctorat, Université de Batna.

[13] Wildi T. : « *Électrotechnique* », 4e édition, De Boeck, 2006.

[14] S. Belkacem «Contribution à la Commande directe du couple de la machine à induction», Thèse de doctorat en sciences en génie électrique, université de Batna, 2011.

[15] J.Bonal, G.Seguier, «Entrainement Electrique à Vitesse Variable, Rappels d'Electrotechnique de Puissance et d'Automatisme les Variateurs Electroniques de Vitesse», Volume 2, Edition Technique et Documentation, Paris, 1998.

[16] Saouli Youcef : « Étude et modélisation d'un moteur asynchrone pour commande vectorielle », Mémoire de Master, Université de Béjaïa, 2019.

[17] P. C. Krause, O. Wasynczuk, S. D. Sudhoff, *Analysis of Electric Machinery and Drive Systems*, 3rd ed., Wiley-IEEE Press, 2013.

[18] Z. Djaghout : « Notes de cours de commande », Université Badji Mokhtar, Annaba, 2007.

[19] Lotfi Baghli : « Contribution à la commande de la machine asynchrone : utilisation

de la logique floue, des réseaux de neurones et des algorithmes génétiques », Thèse de doctorat, Université Henri Poincaré (Nancy I), 1999.

[20] Boubekri B. et Khechkhouche A. : « Commande d'une machine monophasée par réseaux de neurones », Thèse d'ingénieur, ENP, Alger, 2002.

[21] Messaoudi A. : « Modélisation et commande d'un moteur synchrone », Thèse de magister, Université de Batna, 2001.

[22] Stone J. V.: « A Very Short History of Artificial Neural Networks », *Medium*, 2017.

[23] Bennia O. et Mohamadi L. : « Identification des systèmes non linéaires par réseaux de neurones », Université de M'sila, 2002.

[24] Charu C. Aggarwal: « *Neural Networks and Deep Learning – A Textbook* », Springer, 2018.

[25]: Zighem H, « Commande DTC par réseaux de neurones d'un moteur à induction alimenté par un onduleur de tension », université Kasdi Merbah, Ouargla 2009.

[26] J.-P. Nadau : « Les réseaux de neurones artificiels et l'algorithme de rétropropagation de l'erreur », Support de cours, Université Paul Sabatier – Toulouse, 2015.

[27] Guenif N. et Djotni A. : « Technique MPPT d'un système PV par le réseau de neurone », Université Badji Mokhtar, Annaba, 2023.

[28] C. M. Bishop: « *Neural Networks for Pattern Recognition* », Oxford University Press, 1995.

[29] Hervé A. : « Les réseaux de neurones », Université de Grenoble, 1994.

[30] B. Krose et P. van der Smagt : « *An Introduction to Neural Networks* », Academic Press, 1996.

[31] M. T. Hagan, H. B. Demuth et M. H. Beale: « *Neural Network Design* », 2e édition, Martin Hagan, 2014.

[32] R. Marino et al., “Sensorless control of induction motors,” *IEEE Trans. Ind. Electron.*, vol. 42, no. 1, 1995.

[33] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed., Pearson, 2009.

[34] P. Thubert, M. Vial, “A low-cost IoT system using ESP8266 microcontroller for environmental monitoring,” IEEE Sensors Applications Symposium (SAS), 2019.