الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira -
Tasdawit Akli Muḥend Ulḥağ - Tubirett -

وزارة التعليم العالي والبحث العلمي
جامعة أكلي محند أولحاج
- البويرة -

Faculté des Sciences et des Sciences Appliquées

كلية العلوم والعلوم التطبيقية

Référence : ...../MM/2021

المرجع:...../م م/ 2021

Mémoire de Master

Présenté au

Département : Génie Électrique
Domaine : Sciences et Technologies
Filière : Electronique
Spécialité : Electronique des systèmes embarqués

Réalisé par :

BOUGRID CHAHINEZ

Thème

Implémentation des filtres adaptatif appliqués pour l'identification des systèmes linéaires

Soutenu le : **03/07/2023**

Devant le Jury composé de :

Mr :	HAROUN SMAIL	M.C.A	Univ. Bouira	Président
	BENZIANE MOURAD	M.A.A	Univ. Bouira	Rapporteur
	FEKIK AREZKI	M.C.A	Univ. Bouira	Examinateur

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

وزارة التعليم العالي و البحث العلمي
جامعة أكلي محند أولحاج - البويرة

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira
Tasdawit Akli Muhend Ulhağ - Tubirett

نموذج التصريح الشرفي الخاص بالالتزام بقواعد النزاهة العلمية لإنجاز بحث.

انا الممضي اسفله،

السيد(ة) Bougrid chaima الصفة: طالب، استاذ، باحث Etudiante

الحامل(ة) لبطاقة التعريف الوطنية: 107193483 والصادرة بتاريخ 2018/04/03

المسجل(ة) بكلية: **العلوم و العلوم التطبيقية** قسم: **G.E** **الهندسة الكهربائية** ESE

والمكلف(ة) بإنجاز اعمال بحث(مذكرة، التخرج، مذكرة ماستر، مذكرة ماجستير، اطروحة دكتوراه).

عنوانها: Implémentation des filtres adaptatif appliqués pour d'identification des systèmes linéaires

تحت إشراف الأستاذ(ة): Benziane Mouaade

أصرح بشرفي اني ألتزم بمراعاة المعايير العلمية والمنهجية الاخلاقيات المهنية والنزاهة الاكاديمية المطلوبة في انجاز البحث المذكور أعلاه.

التاريخ: 09/07/2023

توقيع المعني(ة)

رأي هيئة مراقبة السرقة العلمية:

النسبة: % 20

Turnitin

الامضاء:

Remerciement

Je tiens tout d'abord à exprimer ma profonde gratitude envers dieu pour m'avoir donné la force, la détermination et la guidance nécessaires tout au long de mon parcours académique et professionnel.

Je souhaite également exprimer ma sincère reconnaissance inverse ma famille, dont le soutien inconditionnel, les encouragements constants et les sacrifices consentis ont été d'une importance capitale pour mon épanouissement et mes réussites. Leur amour et leur soutien indéfectibles ont été une source d'inspiration sans fin.

Je tiens à adresser mes remerciements les plus chaleureux à mon promoteur, monsieur **BENZIANE Mourad**, pour sa guidance experte, ses conseils avisés et sa disponibilité constante. Sa contribution essentielle à mon projet de recherche a été inestimable, et le suis reconnaissante d'avoir eu l'opportunité de bénéficier de son encadrement attentif.

Enfin, je tiens à remercier tous ceux qui des prés ou de loi, ont contribué à la réalisation de ce travail de recherche. Leurs conseils, leurs encouragements et leur soutien ont joué un rôle fondamental dans l'aboutissement de ma thèse de Master.

Mes remerciements vont également à tous ceux qui ont contribué de quelque manière que ce soit à mon cheminement académique et à mon développement personnel. Votre appui a été d'une importance capitale, et je suis honorée d'avoir pu compter sur votre soutien tout au long de cette belle aventure.

Résumé

Le filtrage adaptatif est une technique utilisée pour l'ajustement automatique des paramètres d'un filtre lorsque ces derniers sont difficiles à déterminer où variant dans le temps et parmi les applications de cette approche on trouve l'identification des systèmes linéaires.

Le travail présenté dans ce mémoire porte sur la simulation et l'implémentation de deux systèmes distincts : l'indentification de la fonction de transfert d'un moteur à courant continu et un système d'annulation d'écho acoustique sur une carte NodeMCU, en utilisant trois algorithmes de filtrage adaptatif : LMS, NLMS et RLS. L'objectif de cette étude est d'évaluer les performances de ces différents algorithmes en termes de précision de calcul et de vitesse de traitement.

Mots clés : moteur à courant continu, Annulation écho acoustique, NodeMCU8266, LMS, NLMS, RLS.

Abstract

Adaptive filtering is a technique used for automatically adjusting filter parameters when they are difficult to determine or change over time. Applications of this approach include the linear systems identification.

This work focuses on the simulation and implementation of two different systems: the identification of the transfer function of a DC motor and an acoustic echo cancellation system on a NodeMCU board, using three adaptive filtering algorithms: LMS, NLMS, and RLS. The aim of this study is to evaluate the performances of these different algorithms in terms of computational accuracy and processing speed.

Keywords: DC motor, Acoustic echo cancellation, NodeMCU8266, LMS, NLMS, RLS

ملخص

التصفية التكيفية هي تقنية مستخدمة للضبط التلقائي لمعلمات المرشح عندما يصعب تحديد الأخير أو تغييره بمرور الوقت ومن بين تطبيقات هذا النهج نجد تحديد الأنظمة الخطية.

يركز العمل المقدم في هذه الرسالة على محاكاة وتنفيذ نظامين متميزين: تحديد وظيفة النقل لمحرك DC ونظام إلغاء صدى صوتي على لو□ة NodeMCU ، باستخدام ثلاث خوارزميات ترشيح تكيفية LMS :و NLMS و . RLS الهدف من هذه الدراسة هو تقييم أداء هذه الخوارزميات المختلفة من□يث دقة الحساب وسرعة المعالجة.

الكلمات الرئيسية: محرك DC ، إلغاء الصدى الصوتي ، NodeMCU8266 ، LMS، NLMS، RLS.

Chapitre III : simulation et évaluation

Liste des figures

Liste des tableaux

Liste des abréviations

LMS	Least Mean Square
NLMS	Normalized Least Mean Square
RLS	Recursive Last Square
RIF	Recursive Inverse Filter
RTOS	Real Time Operating Systeme
CPU	Central Processing unit
ASIC	Application-Specific Integrated Circuit
RAM	Random Aces Memory
ROM	Read-Only Memory
IOT	Internet of Things
GPIO	General Purpose Input/Output
IDE	Integrated Development Environment
MSE	Mean Square Error
SNR	Signal to Noise Ration

Introduction générale

Les filtres adaptatifs sont des outils puissants utilisés pour l'dentification des systèmes. L'objectif d'un filtre adaptatif est d'ajuster en permanence les coefficients d'un filtre linéaire afin de s'adapter aux changements du modèle du système à identifier. Il existe plusieurs types d'algorithmes adaptatifs qui se défèrent de leurs performances et de leur complexité de calcul et qu'on peut les classer en deux grandes classes, LMS (***L**east **M**ean **S**quare*) connus pour des performances réduites et une simplicité dans les calculs et RLS (***R**ecursive **L**east **S**quare*) caractérisés par des performances bien meilleurs mais une complexité de calcul élevée.

L'implémentation matérielle des algorithmes de filtrage adaptatif pour l'identification des systèmes linéaires présente de nombreux avantages et utilités. Tout d'abord, elle permet une exécution en temps réel, ce qui est essentiel pour de nombreuses applications en temps réel telles que le contrôle de processus industriels, la communication sans fil ou le traitement du signal en temps réel, elle permet également une exécution rapide et efficace des algorithmes, offrant ainsi des performances en termes de vitesse et de précision. De plus, l'implémentation matérielle offre une plus grande efficacité énergétique car les circuits spécialisés conçus pour les filtres adaptatifs peuvent être optimisés pour minimiser la consommation d'énergie, ce qui est particulièrement important dans les applications embarquées ou alimentées par batterie.

L'objectif principal de ce mémoire est de mettre en œuvre trois algorithmes de filtrage adaptatif : LMS, NLMS et RLS, appliquées pour deux systèmes linéaires différents, il s'agit de la fonction de transfert du moteur à courant continu ainsi que la réponse impulsionnelle du chemin d'écho dans un système d'annulation d'écho acoustique sur la carte NodeMCU8266, une plateforme embarquée couramment utilisée. Cette implémentation est précédée d'une étude en simulation effectuée sur le logiciel MATLAB afin de pouvoir évaluer et comparer les performances en termes de vitesse et de précision de calcul.

Le mémoire est structure autour de trois chapitres :

Le premier chapitre a été consacré à la présentation du filtrage adaptatif d'une manière générale, Parmi une variété vaste d'algorithmes existant, nous allons plutôt traiter en particulier trois algorithmes : LMS, NLMS et RLS.

Le deuxième chapitre présentera les systèmes embarqués d'une manière générale, en particulier les cartes Arduino et leur environnement de développement. Nous allons focaliser sur et les caractéristiques de la carte NodeMCU8266 qui représente un choix approprié pour notre étude.

Le troisième chapitre portera sur l'implémentation, l'évaluation des performances et la présentation des résultats obtenus. Nous allons d'abord parler des deux systèmes choisis pour cette étude : le moteur à courant continu et le système d'annulation d'écho acoustique. Dans la deuxième partie, les différents résultats de simulation sont montrés et discuté. Une dernière partie de ce chapitre est consacrée à l'implémentation sur la carte NodeMCU8266 et à la comparaison de résultats.

Nous terminerons notre travail par une conclusion générale.

Chapitre □

Filtrage adaptatif

I.1. Introduction

Dans ce premier chapitre, nous allons donner les concepts de base autour d'un filtre adaptatif. Dans un premier lieu, la définition et l'historique de ces techniques ainsi que son principe mathématique sont présentés, ensuite, les différentes applications sont montrées et enfin, les algorithmes de bases avec leurs équations de calcul sont minutieusement détaillés. Nous allons en particulier, parler de trois algorithmes (LMS, NLMS, RLS), qui seront implémentés dans les cartes Arduino.

I.2. Définition Filtrage adaptatif

Le filtrage adaptatif est une technique utilisée pour l'ajustement automatique des paramètres d'un filtre lorsque ces derniers sont difficiles à déterminer où variant dans le temps. La synthèse des filtres adaptatifs impose le plus souvent les phases suivantes :

- Spécification des performances désirées (minimisation de l'énergie du signal d'erreur).
- Définition de la structure du filtre utilisé en vue de réaliser les performances souhaitées.
- Ajustement automatique des paramètres qui sera réalisé par un algorithme d'adaptation, cet algorithme sera évalué en fonction de ses performances (la convergence, la poursuite des variations…etc.).

Un filtre adaptatif est généralement constitué de deux parties distinctes :

- Un filtre de structure utilisé pour améliorer une fonction.
- Un algorithme adaptatif pour ajuster les coefficients de ce filtre de façon à minimiser l'énergie à la sortie du filtre à partir de la différence entre la valeur réelle et la valeur estimé.

Le filtrage adaptatif regroupe un ensemble de techniques : le filtrage au sens de Wiener, le filtrage au sens des moindres carrés et le filtrage de Kalman, dédiées à la résolution d'un des problèmes les plus importants des sciences de l'ingénieur, à savoir, la reproduction d'un signal en milieu bruité [1].

I.3. Historiques

- Le début d'une théorie sur l'estimation dans laquelle plusieurs tentatives sont faites pour minimiser une fonction d'erreur remonte à Galileo Galilée en 1632.
- L'origine de la théorie sur l'estimation linéaire est créditée à Gauss qui en 1795 inventa la méthode des moindres carrés.

- Les premières études utilisant l'erreur quadratique moyenne dans les systèmes stochastiques sont dues à Kolmogorov, Krein, et Wiener vers la fin des années 1930.
- Wiener formula (en temps continu) le problème de filtrage pour estimer un système corrompu par du bruit.
- En 1947, Levinson formula le filtre de Wiener en temps discret.
- Les premiers travaux sur les filtres adaptatifs ont commencé vers la fin des années 1950.
- Swerling et Kalman, en 1958 et 1960 respectivement, furent les premiers à s'attaquer aux systèmes non stationnaires.
- L'un des premiers algorithmes sur le filtrage adaptatif est l'algorithme du gradient stochastique ou LMS (least-mean-square) conçu par Widrow et Hoff en 1959.
- Le "Filtrage adaptatif" a connu une grande activité dans la communauté du traitement du signal et des images depuis les années 1970 [2].

I.4. Principe du filtrage adaptatif

Le filtrage adaptatif et ses méthodes d'application se sont considérablement répandus depuis les années 1960. Ce développement du filtrage adaptatif est né de l'essor du traitement numérique, de la croissance soutenue de la puissance du processeur permet une exécution en temps réel, Les algorithmes deviennent de plus en plus complexes à un rythme croissant. Les méthodes adaptatives en traitement du signal, ont pour objectif :

L'adaptation des outils de traitement aux propriétés statistiques des signaux et des systèmes, ainsi que l'adaptation à leurs fluctuations dans le temps. Il s'agit donc d'un mélange bien équilibré, entre la stationnarité et le non stationnarité.

La stationnarité permet de maintenir de façon permanente, dans le temps les propriétés statistiques, grâce auxquelles sont éliminées ou tout au moins réduites les fluctuations purement aléatoires. Le non stationnarité, est la variation lente ou rapide, au cours du temps des propriétés statistiques, sans lesquelles, il n'y aurait nul besoin d'adaptation.

En l'absence de signal et que le système oscille, le filtre optimal ne peut être calculé qu'une seule fois. Les filtres peuvent être classés comme linéaires ou non linéaires [3].

Les filtres adaptatifs sont des filtres numériques dont les coefficients sont déterminés et mis à jour par un algorithme adaptatif, comme illustré à la Figure □.1.

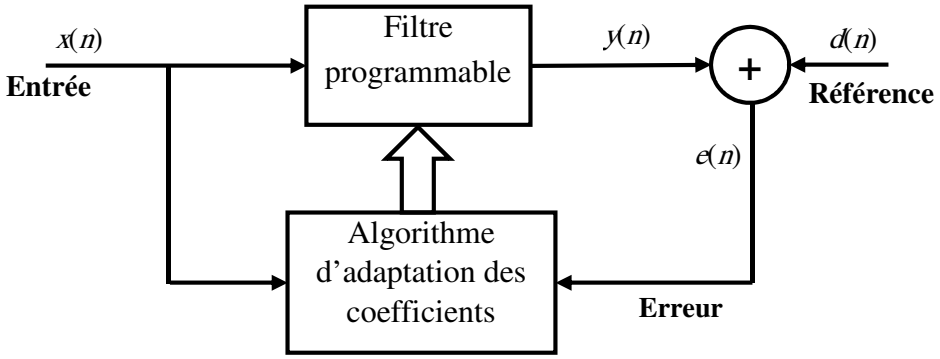


Figure □.1 : Schéma de principe du filtrage adaptatif

Les filtres adaptatifs peuvent être classés en fonction des choix qui sont faits sur les points suivants :

- ✓ Le critère d'optimisation,
- ✓ L'algorithme de mise à jour des coefficients,
- ✓ Le type de signal traité, mono ou multidimensionnel [4].

I.5. Applications

Le rôle d'un filtre adaptatif est d'ajuster ses coefficients **w** pour un objectif bien défini (minimisation de l'EQM : erreur quadratique moyenne) [5]. Le filtrage adaptatif est un outil puissant dans le traitement du signal, les communications numériques et le contrôle automatisé. Les applications varient, mais ont les propriétés suivantes :

Nous avons l'entrée $x(n)$ et la réponse désiré (référence) $d(n)$ et l'erreur $e(n)$ qui est la différence entre $d(n)$ et la sortie du filtre $y(n)$ utilisé pour calculer la valeur. Contrôle (accorde) les coefficients du filtre. Ce qui distingue essentiellement les résultats de l'application est la façon dont la réponse désiré $d(n)$ est définie. On peut distinguer quatre grandes classes d'applications :

I.5.1. Identification des systèmes

Une préoccupation particulière des ingénieurs en automatisation est de savoir comment déduire la fonction de transfert et le comportement d'un système inconnu à partir de la connaissance de son excitation $x(n)$ et de sa puissance $y(n)$. Un filtre qui représente un modèle est estimé en observant la différence entre la sortie du système et la valeur estimée à la sortie du filtre. Le schéma de la figure □.2 montre le principe de l'identification, où $d(n)$ est la sortie du système identifié [6]. Notons que notre travail est une application typique de ce principe.

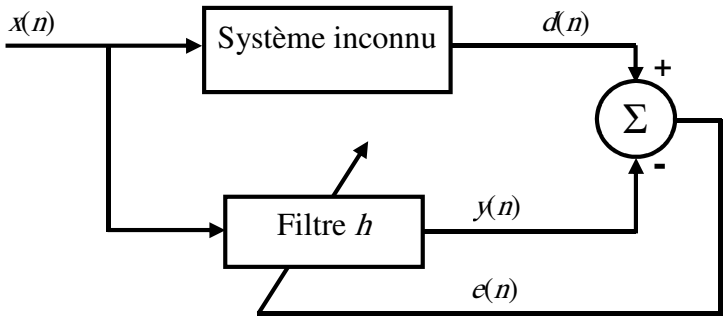


Figure □.2 : Identification d'un système

I.5.2. Prédiction

Puisque nous connaissons la valeur mesurée du signal à l'instant *n*, nous essayons de prédire la valeur à cet instant en utilisant la valeur à l'instant précédent. Ce prédicteur correspond souvent à l'estimation sans interférence de $x(n)$. Cela se fait selon le schéma de principe de la figure □.3. $x(n)$ est le signal à l'instant n et $y(n)$ est le signal prédit à partir du signal au temps précédent.

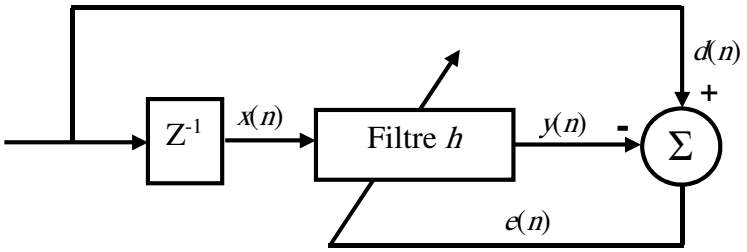


Figure □.3 : Principe de prédiction

I.5.3. Modélisation inverse, égalisation, déconvolution

Le signal x(n) est déformé par le système, comme c'est le cas pour les systèmes de transmission où la distorsion est induite par le canal de transmission. Cette déformation peut toujours être modélisée en utilisant un filtre avec une fonction de transfert H(z) appliqué au signal x(n). Ensuite, le signal déformé peut être traité en utilisant le filtre inverse avec une fonction de transfert G(z) = 1/H(z) [7].

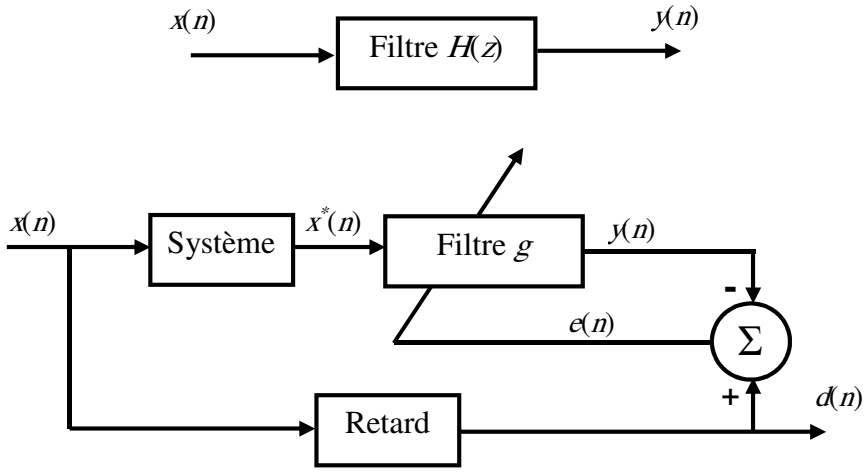


Figure □.4 : Principe de modélisation inverse

Le signal de référence $d(n)$ utilisé ici est la version retardée de $x(n)$, $x^*(n)$ est en fait le résultat de la convolution et l'opération inverse est donc une déconvolution. Pour les télécommunications, nous parlons d'égalisation qui traite de la distorsion de la forme d'onde de la modulation utilisée.

I.5.4. Annulation d'interférences

Le processeur d'annulation des interférences est illustré dans le schéma de la figure □.5. Son but est de restituer un signal utile sans ne lui causer aucune distorsion. En fait, il s'agit d'estimer le bruit qui entache notre signal utile à aide d'un filtre, puis de le soustraire du signal reçu pour acquérir à la fin le signal utile tout seul [8].

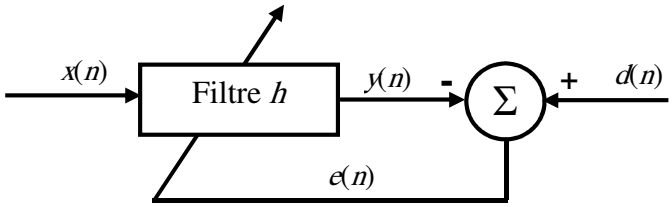


Figure □.5 : Principe d'annulation d'interférences

I.6. Algorithmes de filtrage adaptatif

Le filtrage adaptatif a pour objectif d'approcher les filtres optimaux. Pour cela, les coefficients de la réponse impulsionnelle du filtre sont adaptés en fonction de l'erreur par une boucle de retour comme illustré déjà dans la figure la figure □.1.

Cette adaptation nécessite une séquence d'apprentissage et une stratégie de mise à jour des coefficients du filtre visant à minimiser l'erreur. Un algorithme d'optimisation est utilisé pour cela. Les détails de ces algorithmes dépassant le cadre du traitement du signal, mais on

donnera ici les grandes lignes de trois approches largement utilisées en filtrage adaptatif tel que le LMS, le NLMS et le RLS.

Par conséquence la réponse impulsionnelle d'un filtre adaptatif varie dans le temps, cela dépend du signal reçu, de la séquence d'apprentissage et de l'algorithme d'optimisation utilisé [9].

I.6.1. Critères de performance de l'algorithme

Le choix de l'algorithme se fera en fonction des critères suivants :

- La rapidité de convergence qui sera le nombre d'itérations nécessaires pour converger "assez près" de la solution optimale ;
- La mesure de cette "proximité" entre cette solution optimale et la solution obtenue ;
- La capacité de poursuite (*tracking*) des variations (non-stationnarités) du système ;
- La robustesse au bruit ;
- La complexité ;
- La structure (modularité, parallélisme, ...) ;

Les propriétés numériques (stabilité et précision) dans le cas d'une précision limitée sur les données et les coefficients du filtre [10].

I.6.2. Algorithme du moindre carré moyennes LMS

En 1959, les ingénieurs Bernard Widrow et Marcian Hoff ont développé un algorithme de filtrage adaptatif appelé Moindres Carrés Moyennes (***L**east **M**ean **S**quare,* LMS). Il est devenu l'un des algorithmes les plus largement utilisés dans de nombreuses applications telles que l'égalisation des canaux, l'annulation d'écho et la réduction du bruit. Le principe du LMS est d'ajuster les coefficients d'un filtre pour minimiser l'erreur quadratique moyenne entre le signal d'intérêt et la sortie du filtre. Pour se faire, nous utilisons un vecteur gradient du filtre. Cela en fait, une sorte de filtre adaptatif. L'un des avantages de l'algorithme LMS est sa simplicité de calcul, ce qui en fait une solution attrayante pour de nombreux utilisateurs. Les équations suivantes décrivent l'algorithme LMS pour mettre à jour les poids du filtre adaptatif à chaque itération [5].

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \mathbf{x}(n) e(n) \qquad \textbf{I. 1}$$

$\mathbf{x}(n)$: Vecteur des valeurs d'entrée de même taille que le filtre adaptatif ;

$\mathbf{w}(n)$: Poids du vecteur à l'instant n ;

μ : pas d'adaptation ;

Le paramètre μ est important pour contrôler la rapidité de changement des poids du filtre adaptatif. Son choix peut avoir un impact significatif sur la performance de l'algorithme LMS. En effet, une valeur trop petite de μ peut entraîner une convergence lente vers la solution optimale, tandis qu'une valeur trop grande peut causer une divergence du filtre adaptatif et le rendre instable. Il est donc important de trouver une valeur appropriée de μ pour assurer une convergence rapide et stable de l'algorithme LMS.

I.6.2.1. Dérivation et l'algorithme LMS [5]

L'algorithme LMS est dérivé de la méthode de la plus grande pente et s'inspire de la solution optimale de Wiener. Il repose sur des formules de mise à jour des coefficients du filtre qui utilisent les vecteurs de poids **w** et permettant également de mettre à jour le gradient de la fonction de coût en fonction du coefficient de pondération du filtre.

L'un des avantages clés de l'algorithme LMS est, comme nous l'avons déjà signalé, sa simplicité. En effet, il ne nécessite que $(2L + 1)$ multiplications et $2L$ additions par itération, où L est le nombre de coefficients du filtre. Cette simplicité de calcul le rend largement utilisé dans de nombreuses applications de filtrage adaptatif.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \nabla \varepsilon(n) \qquad \textbf{I. 2}$$

Avec

$$\nabla \varepsilon(n) = \nabla \left\{ |e(n)|^2 \right\} = E\left\{ \nabla |e(n)|^2 \right\} = E\left\{ e(n) \nabla e^*(n) \right\} \qquad \textbf{I. 3}$$

On a :

$$\nabla e^*(n) = -x^*(n) \qquad \textbf{I. 4}$$

Ainsi on obtient la formule suivante :

$$\nabla(n) = -\nabla E\left\{ e(n) x^*(n) \right\} \qquad \textbf{I. 5}$$

En pratique, la valeur de l'espérance $E\left\{ e(n) x^*(n) \right\}$ est souvent inconnue. Donc nous avons besoin d'introduire l'approximation ou estimée comme la moyenne de l'échantillon.

$$\hat{E}\left\{ e(n) x^*(n) \right\} = \frac{1}{L} \sum_{i=0}^{L-1} e(n-l) x^*(n-l) \qquad \textbf{I. 6}$$

Avec cette estimation, on obtient le vecteur poids de mise à jour comme suit :

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu}{L}\sum_{i=0}^{L-1} e(n-l)x^*(n-l) \qquad \textbf{I. 7}$$

Si nous utilisons une moyenne d'échantillon ponctuel ($L=1$) alors :

$$E\{e(n)x^*(n)\} = \mu(n)x^*(n) \qquad \textbf{I. 8}$$

Et enfin, l'équation de mise à jour de poids devient la forme simple :

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)x^*(n) \qquad \textbf{I. 9}$$

Avec :

$$0 < \mu < \frac{1}{tr(R)}$$

Tableau □.1: Resume l'algorithme LMS

Paramètres :	L= ordre du filtre.
	μ= pas d'adaptation
Initialisation:	$\mathbf{w}(0) = \mathbf{0}_{L\times 1}$
Calcul :	Pour $n=0,1,2,\ldots,N-1$
	$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n)$
	$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{x}(n)e(n)$

I.6.3. Algorithme de moindres carrés normalisés (NLMS) :

Le NLMS (*Normalized LMS*) est une variante de l'algorithme LMS, utilisée en traitement du signal pour estimer les filtres adaptatifs. Comme l'algorithme LMS, le NLMS ajuste de manière itérative les poids des filtres en fonction de l'erreur de prédiction entre les sorties réelles et estimées du filtre. La principale différence entre LMS et NLMS est la normalisation des coefficients de pondération mis à jour. En fait, NLMS utilise une normalisation basée sur la puissance d'entrée du filtre, ce qui améliore la convergence de l'algorithme dans des environnements bruyants ou lorsque le niveau de puissance d'entrée varie. En utilisant le paramètre de taille normalisée μ dans LMS, nous obtenons un autre algorithme appelé LMS normalisé (NLMS). La formule utilisée pour calculer le vecteur de poids de mise à jour est :

$$\mu(n) = \frac{\beta}{c + \|x(n)\|^2} \quad \textbf{I. 10}$$

Avec :

$\mu(n)$: Le pas d'adaptation à l'échantillon n.

β : Le pas d'adaptation normalisé ($0 < \beta < 2$)

c : Est une constante pour éviter la division par zéro lorsque la norme du vecteur $x(n)$ est nulle.

I.6.3.1. Dérivation de l'algorithme NLMS

Le NLMS est une variante de l'algorithme LMS qui traite la puissance variable dans le temps du signal d'entrée. Ce changement affecte la vitesse de convergence de l'algorithme LMS, ralentissant la convergence pour les signaux faibles et augmentant l'erreur pour les signaux forts. Pour éviter cette situation, le NLMS ajuste dynamiquement la taille du pas de mise à jour du coefficient en fonction de l'énergie instantanée du signal d'entrée. Cette normalisation améliore la convergence de l'algorithme dans des environnements bruyants ou lorsque les niveaux de puissance d'entrée varient. Afin d'assurer la stabilité de l'algorithme, certaines conditions doivent être remplies [5].

$$0 < \mu < \frac{2}{\lambda_{\max}} \qquad \text{Et} \qquad 0 < \mu < \frac{2}{trac(R_x)} \quad \textbf{I. 11}$$

λ_{max} désigne le maximum des valeurs propres de la matrice d'autocorrélation du vecteur d'entrée $x(n)$. Cependant, dans la pratique, la matrice d'autocorrélation Rx est souvent inconnue, il est donc nécessaire d'estimer le maximum λmax ainsi que Rx pour utiliser les bornes. Pour résoudre ce problème, une nouvelle estimation de la trace de Rx est introduite

$$trac(R_x) = (p+1)E\left\{|x(n)|^2\right\} \quad \textbf{I. 12}$$

Pour $p = 0,1, \ldots$

$E\left\{|x(n)|^2\right\}$: est la puissance du signal d'entrée. Il peut être estimé par l'estimateur :

$$\hat{E}\left\{|x(n)|^2\right\} = \frac{1}{p+1}\sum_{k=0}^{p}|x(n-k)|^2 \quad \textbf{I.13}$$

Par conséquent, les limites du paramètre deviendront :

$$0 < \mu < \frac{2}{(p+1)E\left\{|x(n)|^2\right\}} \quad \textbf{I. 14}$$

Par substitution l'équation 16 dans l'équation 17 on obtient le paramètre μ comme suit :

$$0 < \mu < \frac{2}{x(n)x^*(n)} \quad \textbf{I. 15}$$

Pour les processus complexes variables dans le temps, on calcule le paramètre détaille de l'échelon dans le temps (échantillon N) :

$$\mu = \frac{2}{x^H(n)x(n)} = \frac{\beta}{\|x(n)\|^2} \quad \textbf{I. 16}$$

β : le pas d'adaptation normalise (0<β<2)

En remplaçant μ par $\mu(n)$ dans l'équation 9 pour mettre à jour le vecteur de poids dans algorithme LMS, nous atteignons un nouvel algorithme qui est le NLMS. La mise à jour du vecteur de poids est maintenant :

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{\|x(n)\|^2} e(n)x^*(n) \quad \textbf{I.17}$$

Dans l'algorithme LMS, comme le vecteur de poids w(n) varie en fonction du signal d'entrée x(n), alors lorsque x(n) est très petit le calcul de poids de l'équation de mise à jour du vecteur sera très grand ce qui constitue un problème. Pour cette raison, on met en œuvre la constante « c » qui est un paramètre de régularisation.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{c + \|x(n)\|^2} e(n)x^*(n) \quad \textbf{I. 18}$$

I.6.4. Algorithme de moindre carrées récursifs (RLS)

L'algorithme RLS (***R**ecursive **L**east **S**quare*) implémente récursivement une solution de moindres carrés exacte. Comme indiqué précédemment, la solution de Wiener pour un filtre adaptatif de longueur finie est donnée par :

$$w_{opt} = R^{-1}P \quad \textbf{I. 19}$$

Où R est la matrice d'autocorrélation des entrées et P est la corrélation entre les entrées et le signal de référence. À chaque intervalle de temps, RLS estime récurremment R^{-1}et P sur la base de toutes les données antérieures et calcule le vecteur de poids comme :

$$w_n \underline{\underline{\text{def}}} R^{-1} P_n \qquad \textbf{I. 20}$$

Ce qui est donc la meilleure approximation à jour de la solution Wiener. La formule de mise à jour des poids peut être écrite comme suit :

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu_n R_{n-1}^{-1} a_n \mathbf{x}_n \qquad \textbf{I. 21}$$

Où :

$$a_n = d(n) - \mathbf{w}_{n-1}^T \mathbf{x}_n \qquad \textbf{I. 22}$$

$$\mu_n = \frac{\lambda^{-1}}{1 + \lambda^{-1} \mathbf{x}_n \mathbf{x}_n^T R_{n-1}^{-1}} \qquad \textbf{I.23}$$

λ : est une constante légèrement inférieure à 1, appelée facteur d'oubli.

Cette formulation met l'accent sur l'opération de décorrélation que RLS effectue sur les données d'entrée : le gradient stochastique $\alpha_n X_n$ est prémultiplié par une estimation de la matrice d'autocorrélation inverse, ce qui a pour effet de décorréler l'entrée au filtre adaptatif. Cela réduit la sensibilité de l'algorithme à la propagation de ses valeurs propres d'entrée. La prémultiplication par R^{-1} peut malheureusement nuire à la stabilité du filtre si la matrice R est mal conditionnée ; Une situation qui se produit chaque fois que le filtre contient plus de poids que nécessaire. RLS est donc une version mise à jour du célèbre filtre Wiener. La vitesse de convergence de la solution optimale est rapide, les performances de traitement du signal non stationnaire sont élevées et l'erreur quadratique moyenne minimale est la plus petite pendant la convergence [5].

I.6.4.1. Dérivation de l'algorithme RLS :

Au moment n, toutes les valeurs de l'erreur d'estimation depuis le début de l'algorithme RLS sont requises. De toute évidence, au fur et à mesure que le temps progresse, la quantité de données requises pour traiter cet algorithme augmente. Le fait que la mémoire et les capacités de calcul sont limitées rend l'algorithme RLS une impossibilité pratique dans sa forme la plus pure. Cependant, la dérivation suppose toujours que toutes les valeurs de données sont traitées. En pratique, seul, un nombre fini de valeurs précédentes est considéré, ce nombre correspond à l'ordre du filtre RLS RIF (***R**ecursive **I**nverse **F**ilter),* La dérivation suivante du RLS est résumée

de Farhang-Boroujeny [11]. $y_n(k)$ est défini comme la sortie du filtre RIF, à n, en utilisant le vecteur de poids et le vecteur d'entrée d'un temps précédent k. La valeur d'erreur d'estimation en $e(k)$ est la différence entre la valeur de sortie souhaitée à l'instant k et les valeurs correspondantes de $y_n(k)$. Pour k = 1,2, 3,..., n.

$$y_n(k) = \mathbf{w}^T(n)\mathbf{x}(n) \quad \textbf{I. 24}$$

$$e_n(k) = d(k) - y_n(k) \quad \textbf{I. 25}$$

$$d(n) = \left[d(1), d(2), ..., d(n)\right]^T \quad \textbf{I. 26}$$

$$y_n(n) = \left[y_n(1), y_n(2), ..., y_n(n)\right]^T \quad \textbf{I. 27}$$

$$e(n) = \left[e(1), e(2), ..., e(n)\right]^T \quad \textbf{I. 28}$$

$$e(n) = d(n) - y(n) \quad \textbf{I. 29}$$

Tableau .2 : Résume l'algorithme RLS

Paramètres :	*L*= ordre du filtre.
	λ= facteur d'oubli.
	δ= constante positive.
Initialisation:	$\mathbf{w}(0) = \mathbf{0}_{L\times 1}$
	$\mathbf{Q}(0) = \delta^{-1}\mathbf{I}$
Calcul :	Pour *n=0,1,2,…,N-1*
	$\mathbf{k}(n) = \dfrac{\lambda^{-1}\mathbf{Q}(n-1)\mathbf{x}(n)}{1+\lambda^{-1}\mathbf{x}^T(n)\mathbf{Q}(n-1)\mathbf{x}(n)}$
	$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)$
	$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)e(n)$
	$\mathbf{Q}(n) = \lambda^{-1}\mathbf{Q}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{x}^T(n)\mathbf{Q}(n-1)$

I.6.4.2. Mise en œuvre de l'algorithme RLS :

Comme mentionné précédemment, la mémoire de l'algorithme RLS est limitée à un nombre fini de valeurs correspondant à l'ordre du vecteur de pondération des prises de filtre.

Premièrement, il faudrait noter deux facteurs dans la mise en œuvre RLS : le premier est que, bien que l'inverse de la matrice soit essentiel pour la dérivation de l'algorithme RLS, il n'y a pas de calcul d'inversion de matrice nécessaires pour atteindre, réduisant ainsi considérablement la complexité de calcul algorithme. Deuxièmement, contrairement aux algorithmes basés sur LMS, les coefficients du filtre sont mis à jour pendant l'itération qu'ils doivent utiliser, en utilisant la valeur de dernière itération [11].

I.7. Conclusion

Ce premier chapitre a été consacré à la présentation du filtrage adaptatif d'une manière générale, Parmi une variété vaste d'algorithmes existant, nous avons plutôt étudier en particulier trois algorithmes : LMS, NLMS et RLS qui seront mis en œuvre et évalué dans le troisième chapitre. Le chapitre suivant donnera des généralités sur les systèmes embarqués.

Chapitre II

Généralités sur les systèmes embarqués et choix de la carte

II.1. Introduction

Les systèmes embarqués sont de plus en plus présents dans notre quotidien, dans des domaines tels que l'industrie, la médecine et l'automobile. Pour allier les performances de la programmation à celles de l'électronique, un système tel que Arduino a été conçu en 2004 par un groupe d'enseignants et d'étudiants. Les cartes Arduino permettent de réaliser des prototypes et des maquettes de cartes électroniques pour l'informatique embarquée de manière simplifiée et à moindre coût grâce à leur aspect « ***O**pen **S**ource* » et « ***O**pen **H**ardware* ».

Dans ce chapitre nous allons présenter en premier lieu des généralités sur les systèmes embarqués ensuite on se focalise sur les cartes Arduino servant de support pour notre application de filtrage adaptatif en donnant leurs caractéristiques techniques et leurs avantages et inconvénients.

II.2. Définition d'un système embarqué

Un système embarqué est une combinaison de matériel informatique et de logiciel, et peut-être d'autres pièces mécaniques ou autres, conçues pour exécuter une fonction spécifique. Ces systèmes sont basés sur l'utilisation des calculateurs numérique, entre autres, des microcontrôleurs ou des microprocesseurs conçus pour effectuer des tâches spécifiques. Un système embarqué se compose de composants matériels et de composants logiciels d'application. Il dispose, le plus souvent, d'un système d'exploitation en temps réel (Real Time Operating System RTOS) qui supervise le logiciel d'application et fournit un mécanisme permettant au processeur d'exécuter un processus conformément à la planification en suivant un plan pour contrôler les latences. RTOS définit le fonctionnement du système. Il définit les règles lors de l'exécution du programme d'application. Un système embarqué à petite échelle peut ne pas avoir de RTOS [12].

II.2.1. Caractéristiques d'un système embarqué [13] :

- **Mono-fonction** : En général, un système embarqué est programmé pour accomplir une fonction spécialisée de manière itérative.
- **Très contraint :** Les systèmes embarqués sont soumis à des contraintes particulièrement strictes en termes de métriques de conception, qui mesurent les performances de l'implémentation en fonction de son coût, sa taille, sa puissance et ses performances. Ces systèmes doivent être conçus pour tenir sur une seule puce, traiter les données en temps réel à grande vitesse et consommer le moins d'énergie possible pour prolonger la durée de vie de la batterie.

- **Réactif et temps réel :** Les systèmes embarqués doivent être réactifs et capables de traiter les informations en temps réel sans délai. Ils doivent réagir constamment aux changements de l'environnement du système et effectuer des calculs rapides pour assurer un fonctionnement efficace.
- **Basé sur des microprocesseurs :** Le système embarqué doit être construit autour d'un microprocesseur ou d'un microcontrôleur pour fonctionner.
- **Mémoire :** il doit disposer d'une mémoire car son logiciel est généralement stocké dans la ROM. Contrairement aux ordinateurs, il n'a généralement pas besoin de mémoire secondaire.
- **Connecté :** il doit être doté de périphériques connectés pour pouvoir interagir avec les périphériques d'entrée et de sortie.
- **Systèmes HW-SW :** Le logiciel est utilisé pour plus de fonctionnalités et de flexibilité. Le matériel est utilisé pour les performances et la sécurité.

II.2.2. Utilisations d'un système embarqué

Les systèmes embarqués sont omniprésents dans notre vie quotidienne, que ce soit dans des applications grand public, industrielles, médicales ou militaires. Ils sont présents dans une multitude d'appareils tels que les lave-vaisselles, les ouvre-portes de garage, les thermostats programmables, les manettes de jeux vidéo, les lecteurs de cartes, etc. En effet, environ 98% des microprocesseurs sont utilisés dans des systèmes embarqués, tandis que moins de 2% sont utilisés dans les ordinateurs. Les systèmes embarqués continuent de transformer chaque aspect de notre vie et sont utilisés dans de nombreux domaines d'application [13].

- **Automobile :** tel que les systèmes de mise à feu, Contrôle du moteur, Système de freinage
- **Electronique grand public** : tel que les téléphones cellulaires, Décodeurs, Assistants de données personnelles, Appareils de cuisine, Automobile, aéronautique… ;
- **Industriel :** Contrôle de la robotique et des systèmes de contrôle (fabrication), Captures intelligentes … ;
- **Médical :** Pompes à perfusion**,** Machines de dialyse**,** Prothèses, Moniteurs cardiaques**,** Imagerie (rayon X, ultra-sons, IRM) … ;
- **Bureautique :** Télécopieur, Photocopieur, Moniteurs, Scanners … ;

II.2.3. Structure de base d'un système embarqué

Les systèmes embarqués impliquent l'utilisation de différents composants tels que les capteurs qui mesurent une quantité physique et la convertissent en signal électrique, le

convertisseur A/N qui convertit le signal analogique en signal numérique, les processeurs et ASIC qui traitent les données et les stockent dans la mémoire, le convertisseur N/A qui convertit les données numériques en données analogiques, et les actionneurs qui comparent la sortie réelle avec la sortie attendue et stockent la sortie approuvée. Ces composants sont utilisés dans divers domaines d'application tels que l'industrie, la médecine, l'armée et les appareils électroniques grand public [14]. La figure □.1 illustre la structure de base d'un système embarqué.

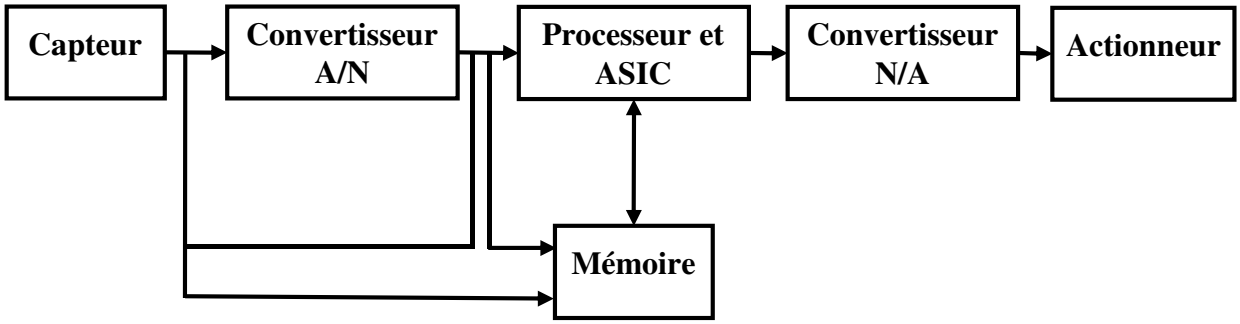


Figure □.1 : la structure de base d'un système embarque

II.2.4. Les processeurs

Le processeur est la composante principale d'un système embarqué, responsable de la prise en charge des entrées et de la production des sorties après le traitement des données. Pour un concepteur de systèmes embarqués, il est important de comprendre les microprocesseurs et microcontrôleurs. Le processeur est composé de deux unités principales [15] :

- **L'unité de contrôle de flux de programme (CU) :** comprend une unité d'extraction (UExt) qui extrait les instructions de la mémoire, avec des circuits implémentant les instructions de transfert de données et de conversion de données
- **L'unité d'exécution (UE)** : comprend l'unité arithmétique et logique (ALU) ainsi que les circuits qui exécutent des instructions pour une tâche de contrôle de programme telle qu'une interruption ou un saut vers un autre ensemble d'instructions.

II.2.4.1. Microprocesseur

Un microprocesseur est un composant électronique qui regroupe sur une seule puce un processeur central (CPU), de la mémoire vive (RAM), de la mémoire morte (ROM) ainsi que d'autres composants essentiels pour le fonctionnement d'un ordinateur. Il est conçu pour traiter les instructions d'un programme informatique et effectuer des opérations arithmétiques et logiques sur les données en temps réel. La figure □.2 représente le schéma de principe d'un microprocesseur [16].

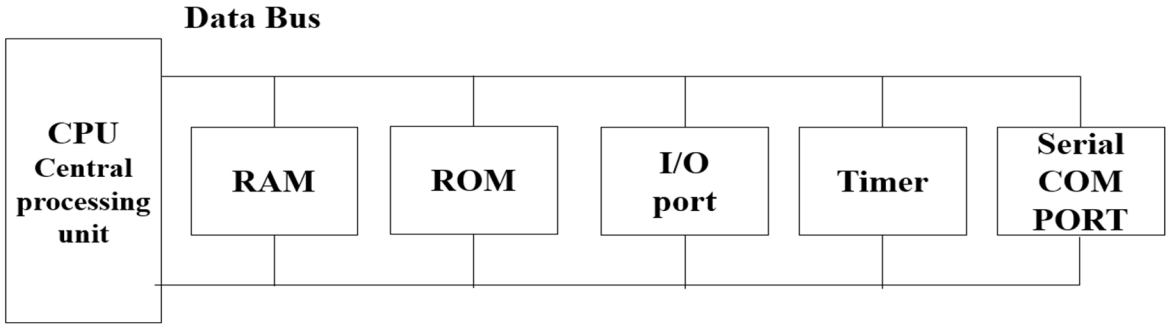


Figure □.2 : Un schéma de principe simple d'un microprocesseur

II.2.4.2. Microcontrôleur

Un microcontrôleur (ou *Microcontroller*) est un circuit intégré qui regroupe sur une seule puce un microprocesseur, une mémoire et des périphériques d'entrée/sortie (E/S) nécessaires pour contrôler des systèmes embarqués. Contrairement aux microprocesseurs qui sont conçus pour une large gamme d'applications, les microcontrôleurs sont conçus pour des tâches spécifiques et sont souvent utilisés dans des produits électroniques tels que les appareils ménagers, les jouets, les systèmes de contrôle industriels, les voitures, les avions, etc. Les microcontrôleurs sont généralement peu coûteux et faciles à utiliser, et sont programmables en langages de programmation de haut niveau tels que le C ou le C++ [16].

Tableau □.1 : puce de microcontrôleur

CPU	RAM	ROM
I/O Port	Timer	Serial COM Port

II.2.5. Comparaison entre microcontrôleur et microprocesseur [17]

Tableau □.2 : Résume la différence entre le microprocesseur et le microcontrôleur

Microprocesseur	**Microcontrôleur**
Les microprocesseurs sont de nature multitâche. Peuvent effectuer plusieurs tâches àla fois. Par exemple, sur ordinateur, nous pouvons jouer de la musique tout en écrivant du texte dans l'éditeur de texte.	Orienté vers des tâches uniques. Par exemple, une machine à laver est conçue pour laver des vêtements uniquement.
La RAM, la ROM, les ports d'E/S et les minuteries peuvent être ajoutés en externe et peuvent varier en nombre.	La RAM, la ROM, les ports d'E/S et les minuteries ne peuvent pas être ajoutés en externe. Ces composants doivent être intégrés ensemble sur une puce et sont fixés en nombre.

Les concepteurs peuvent décider du nombre de mémoire ou de ports d'E/S nécessaires.	Un nombre fixe de mémoire ou d'E/S rend un microcontrôleur idéal pour une tâche limitée mais spécifique.
La prise en charge externe de la mémoire externe et des ports d'E S rend un système à microprocesseur plus lourd et plus coûteux.	Les microcontrôleurs sont légers et moins chers qu'un microprocesseur.
Les périphériques externes nécessitent plus d'espace et leur consommation d'énergie est plus élevée.	Un système basé sur un microcontrôleur consomme moins d'énergie et prend moins de place.

II.3. Description de la partie matérielles

II.3.1. Choix de la carte programmable

Durant notre recherche sur les cartes programmables, nous avons constaté qu'au cours des dernières années, de nombreuses cartes de développement modulaire sont apparues on site les plus connus ; carte Arduino, Raspberry-pi, ou ESP8266 qui est des micro-ordinateurs dotés de systèmes d'exploitation Lunix, Beaglebone qui possède les fonctionnalités d'un ordinateur basique. Tous ces dispositifs présentent des avantages et des inconvénients et se diffèrent par leurs fonctionnalités leurs complexités et leurs prix. Et pour notre projet nous avons choisi la carte Arduino le modèle Uno ainsi que la ESP8266.

II.3.1.1. Arduino

L'Arduino est un microcontrôleur programmable qui permet, comme son nom l'indique, de contrôler des éléments mécaniques : systèmes, lumières, moteurs, etc. Cette carte électronique permet donc à son utilisateur de programmer facilement des applications et de créer des mécanismes automatisés, sans avoir de connaissances particulières en programmation. C'est un outil conçu et destiné aux inventeurs, artistes ou amateurs qui souhaitent créer leur propre système automatique.

II.3.1.1.1. Arduino Uno

Arduino Uno est l'un des modèles les plus rependus des carte Arduino. Il permet un large éventail de possibilités. C'est la première version stable de carte Arduino. Elle possède toutes les fonctionnalités d'un microcontrôleur classique en plus de sa simplicité d'utilisation. Livré avec une puce ATMega328P cadencée à 16 Mhz. Elle possède 32 ko de mémoire flash destinée à recevoir le programme, 2 ko de SRAM et 1 ko d'EEPROM. Elle offre 14 broches d'entrée/sortie numérique dont 6 pouvant générer des PWM. Chaque broche est capable de

délivrer un courant de 40 mA pour une tension de 5V. La carte Arduino peut aussi s'alimenter et communiquer avec un ordinateur grâce à son port USB. On peut aussi l'alimenter avec une alimentation comprise en 7V et 12V grâce à son connecteur Power Jack. Pour obtenir plus de fonctionnalités que la carte elle-même ne possède pas elle peut être relié à des extensions, Modules et Shields [18][19].

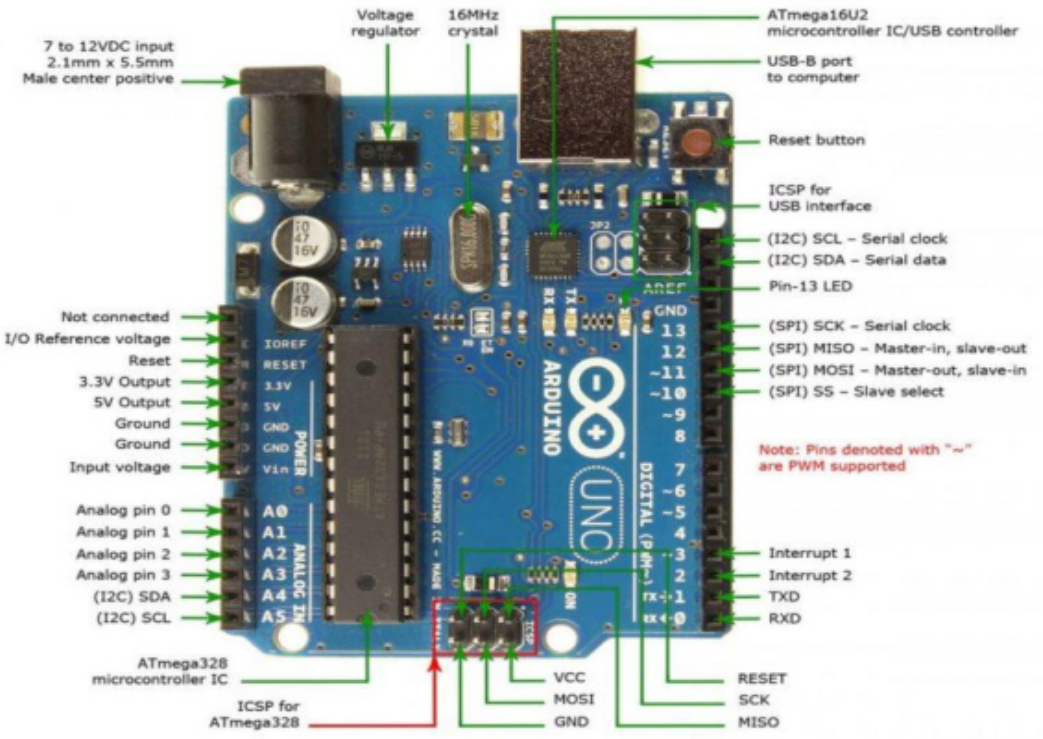


Figure □.3 : Arduino uno avec ces principales caractéristiques techniques

Pour l'application qu'on souhaite effectuer, la carte Arduino Uno n'est pas, tout à fait, en mesure de nous offrir les résultats souhaités pour tous les algorithmes vu la capacité de sa mémoire réduite. De ce fait, nous avons opté pour une autre carte qui nous permet d'avoir un peu plus de performances, il s'agit donc de la carte ESP8266 [18].

II.3.1.2. NodeMCU ESP8266

La carte ESP8266 est un microcontrôleur doté de fonctionnalités Wi-Fi intégrées, développé par ***Espressif Systems***. Elle est souvent utilisée dans des projets IoT (Internet des objets, ***Internet of Things***) et des applications de connectivité sans fil, comme illustré sur la figure II.4.

La carte ESP8266 est disponible sous différentes formes, notamment sous la forme de modules prêts à l'emploi, tels que l'ESP-01, l'ESP-12E et l'ESP-32. Ces modules intègrent le microcontrôleur ESP8266, une puce Wi-Fi et des broches GPIO (***General Purpose***

***I**nput/**O**utput*) pour la connectivité et l'interaction avec d'autres composants. Elle est programmable dans divers langages et environnements de développement, tels que l'IDE Arduino, le langage Lua avec NodeMCU, MicroPython et bien d'autres. Cela le rend accessible aux développeurs de différents niveaux d'expérience [20].

Grâce à sa connectivité Wi-Fi, l'ESP8266 peut se connecter à des réseaux sans fil, se synchroniser avec des serveurs distants et échanger des données avec d'autres périphériques connectés. Cela permet de créer une grande variété de projets IoT, tels que des capteurs sans fil, des systèmes de surveillance à distance, des objets connectés pour la maison intelligente, et bien plus encore. La communauté des utilisateurs de l'ESP8266 est très active, ce qui facilite l'accès à des tutoriels, des exemples de code et des forums d'entraide. De plus, la popularité de l'ESP8266 a conduit à la création de nombreuses bibliothèques et extensions, offrant une flexibilité et des fonctionnalités étendues pour les projets basés sur cette carte [21][22].

II.3.1.3. NodeMCU 8266

La carte NodeMCU ESP8266 est l'une des variantes populaires de la famille ESP8266. Elle combine le microcontrôleur ESP8266 avec des composants supplémentaires pour faciliter le développement de projets IoT. Voici les caractéristiques techniques typiques de la carte NodeMCU ESP8266 [21][22] :

- Microcontrôleur : ESP8266EX (32 bits RISC CPU).
- Fréquence d'horloge : 80 MHz (peut être augmentée jusqu'à 160 MHz).
- Mémoire flash intégrée : 4 Mo (certains modèles peuvent avoir une capacité différente).
- Mémoire RAM : généralement de 80 Ko à 160 Ko (partagée entre l'exécution du programme et les données).
- Wi-Fi : compatible avec les normes 802.11 b/g/n (2,4 GHz).
- Interfaces de communication : UART, SPI, I2C, GPIO.
- Tension d'alimentation : 5 V via le port USB ou 3,3 V via les broches d'alimentation.
- Ports USB : micro-USB pour l'alimentation et la programmation.
- Broches d'E/S (GPIO) : généralement 10 à 11 broches d'E/S pour la connectivité avec des périphériques externes.
- Programmation : compatible avec l'IDE Arduino, Lua avec NodeMCU firmware, Micro Python et d'autres environnements de développement.
- Dimensions : varient en fonction des modèles spécifiques, mais généralement compactes et adaptées aux projets embarqués.

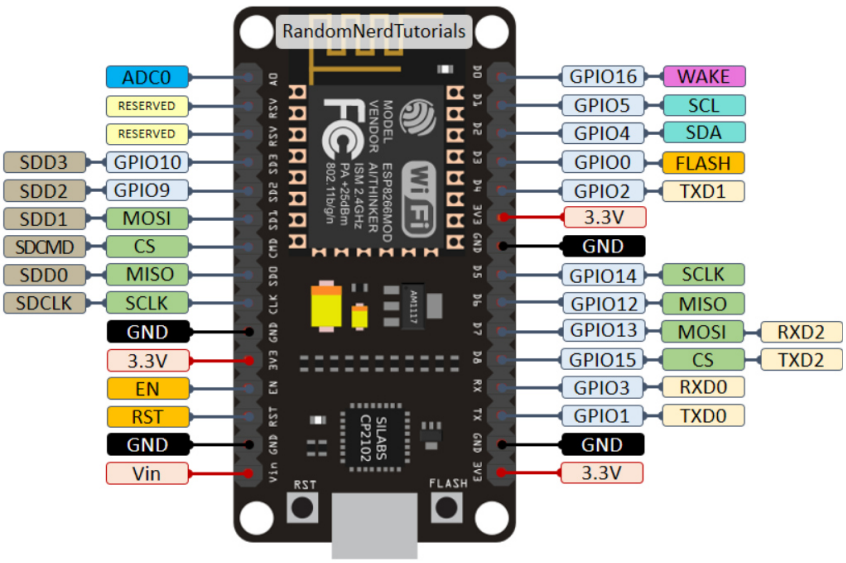


Figure .4 : la carte NodeMCU 8266

La carte NodeMCU ESP8266 est appréciée pour sa facilité d'utilisation et sa polyvalence, offrant une plateforme de développement rapide pour les projets IoT. Elle dispose également d'une communauté active de développeurs qui partagent des exemples de code, des tutoriels et des bibliothèques pour faciliter le processus de développement.

II.3.1.4. Mémoires et manipulation de données

La carte NodeMCU ESP8266 dispose de deux types de mémoire principaux : la mémoire flash intégrée et la mémoire RAM [20] :

- **Mémoire flash** : La plupart des modèles de la carte NodeMCU ESP8266 sont équipés de 4 Mo de mémoire flash intégrée. Cette mémoire est utilisée pour stocker le firmware du microcontrôleur, les programmes, les bibliothèques et les données non volatiles. Il est important de noter que l'espace disponible pour le stockage des programmes et des données utilisateur peut varier en fonction du firmware et des bibliothèques utilisées. Une partie de la mémoire flash est généralement réservée pour le système d'exploitation et les fonctions internes de l'ESP8266.
- **Mémoire RAM** : La carte NodeMCU ESP8266 dispose généralement de 80 Ko à 160 Ko de mémoire RAM (***R**andom **A**ccess **M**emory*). Cette mémoire est utilisée pour l'exécution du programme en cours d'exécution et pour stocker les données temporaires lors de l'exécution du code. Une partie de la mémoire RAM est également utilisée par le système d'exploitation et les pilotes internes de l'ESP8266.

II.3.2. Comparaison entre Arduino et NodeMCU

Pour justifier notre choix de carte nous allons faire une comparaison entre les deux cartes citées ci-dessus. La comparaison entre la carte Arduino Uno et la carte NodeMCU (basée sur l'ESP8266) peut être faite sur plusieurs aspects clés [20] :

- Microcontrôleur : L'Arduino Uno est basé sur l'ATmega328P, un microcontrôleur 8 bits, tandis que la carte NodeMCU est basée sur l'ESP8266, un microcontrôleur 32 bits avec des fonctionnalités Wi-Fi intégrées.
- Puissance de calcul : En raison de sa nature 32 bits, l'ESP8266 offre une puissance de calcul supérieure par rapport à l'ATmega328P de l'Arduino Uno.
- Connectivité : L'Arduino Uno n'a pas de connectivité sans fil intégrée, tandis que la carte NodeMCU dispose d'un module Wi-Fi intégré, permettant une connectivité sans fil directe.
- GPIO et interfaces : L'Arduino Uno dispose de 14 broches GPIO numériques et 6 broches PWM, ainsi que de 6 broches d'entrée analogique. La carte NodeMCU offre généralement plus de broches GPIO (généralement 10 à 11), mais moins de broches PWM (généralement 1 à 2), ainsi que des broches d'entrée analogique.
- Mémoire : L'Arduino Uno dispose de 32 Ko de mémoire flash et de 2 Ko de mémoire RAM. La carte NodeMCU offre généralement 4 Mo de mémoire flash et entre 80 Ko et 160 Ko de mémoire RAM.
- Programmation : Les deux cartes peuvent être programmées en utilisant l'IDE Arduino, mais la carte NodeMCU offre également la possibilité d'utiliser Lua avec le firmware NodeMCU, ainsi que MicroPython et d'autres environnements de développement.
- Compatibilité des shields : L'Arduino Uno est compatible avec un large éventail de shields (cartes d'extension), ce qui facilite l'ajout de fonctionnalités supplémentaires. La carte NodeMCU a une forme différente et n'est pas compatible avec les shields Arduino traditionnels, mais elle dispose de ses propres modules d'extension spécifiques à l'ESP8266.

En résumé, la carte Arduino Uno est un choix classique pour les projets basés sur l'Arduino et offre une grande compatibilité avec les shields existants. Elle est plus adaptée aux projets sans nécessité de connectivité Wi-Fi. En revanche, la carte NodeMCU (ESP8266) offre une connectivité Wi-Fi intégrée et une puissance de calcul supérieure, ce qui la rend idéale pour les projets IoT et les applications nécessitant une communication sans fil. Elle offre également une plus grande capacité de mémoire, permettant de stocker plus de programmes et de données.

II.3.3. Choix de l'environnement de développement

La carte NodeMCU peut être programmée en utilisant l'IDE Arduino, ce qui facilite son utilisation pour les développeurs familiers avec l'environnement de développement Arduino. Néanmoins in faut ajouter le support pour la carte NodeMCU.

Un langage de programmation est un langage permettant à un être humain d'écrire un ensemble d'instructions (code source) qui seront directement converties en langage machine grâce à un compilateur (c'est la compilation). Les créateurs d'Arduino ont développé un logiciel pour que la programmation des cartes Arduino soit visuelle, simple et complète à la fois. C'est ce que l'on appelle une IDE, qui signifie ***I**ntegrated **D**evelopment **E**nvironnent* [23][24].

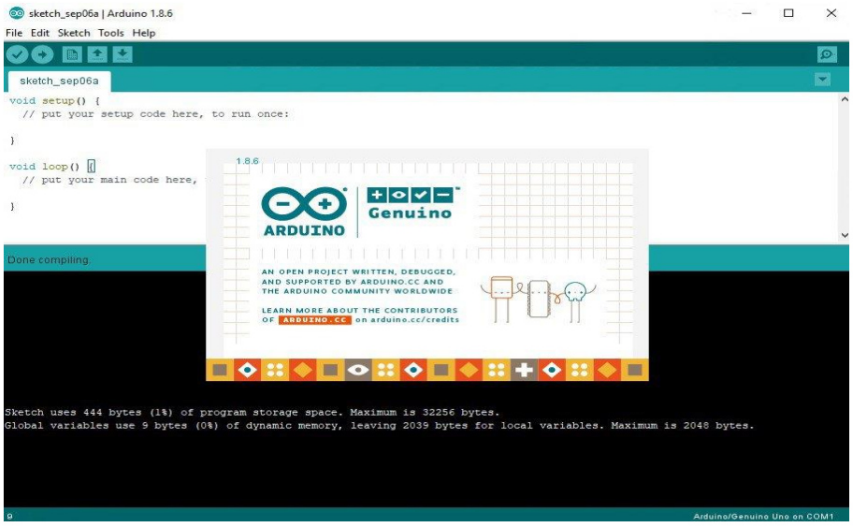


Figure □.5 : la fenêtre qui apparait lors de l'activation du programme

L'IDE affiche une fenêtre graphique qui contient un éditeur de texte et tous les outils nécessaires à l'activité de programmation. On peut donc écrire le programme, l'enregistrer, le compiler, le vérifier, le transférer sur une carte Arduino, néanmoins il est nécessaire d'installer les bibliothèques des composants utilisées dans notre projet pour qu'ils soient reconnue par IDE lors du transfert de notre programme [23][24].

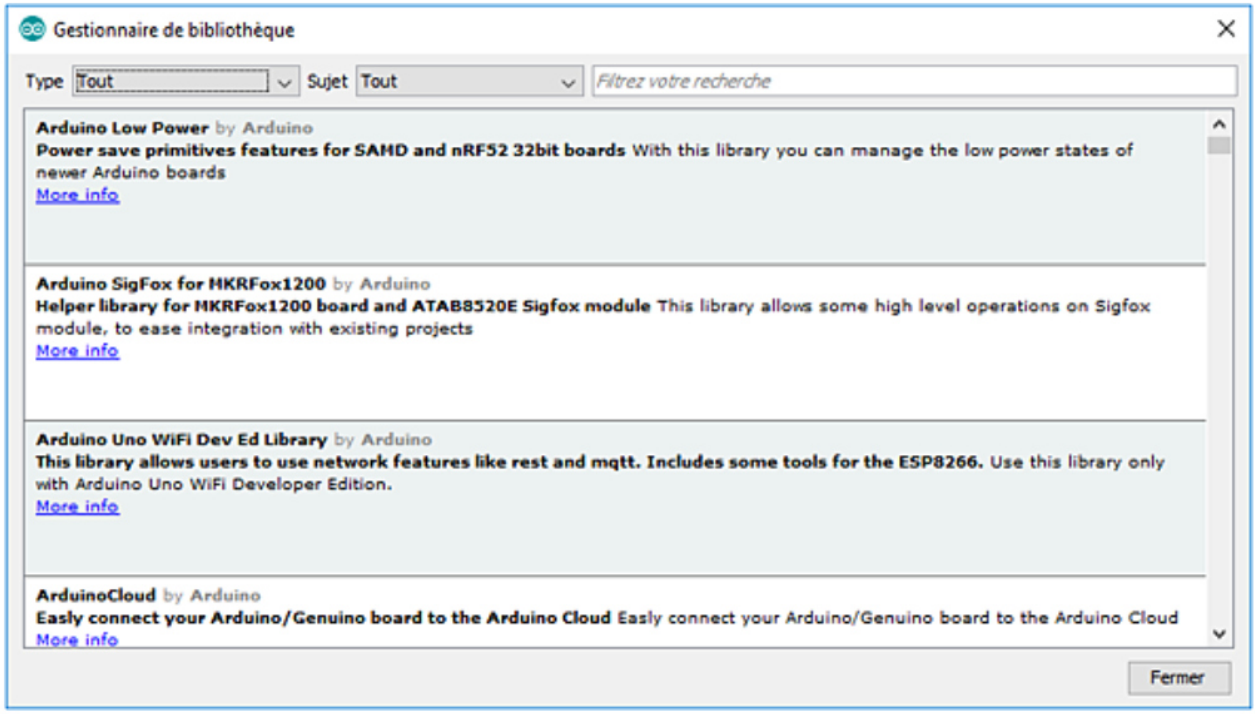


Figure □.6 : Gestionnaire de bibliothèque Arduino IDE

L'aspect est à peu près identique sur chaque plate-forme (Windows, Mac et Linux). Le langage Arduino est inspiré de plusieurs langages. On retrouve notamment des similarités avec le C, le C++, le Java et le Processing [23][24]. La structure de l'IDE se compose de trois fonctions principales :

- **La partie déclaration des variables (optionnelle)**
- **La partie initialisation et configuration des entrées/sorties (void setup)** : La fonction setup () est appelée lorsqu'un croquis commence. On l'utilise pour initialiser les variables, les pins, commencé à utiliser les bibliothèques, etc. La fonction de configuration ne s'exécute qu'une fois, après chaque mise sous tension ou réinitialisation de la carte Arduino.
- **La partie principale qui s'exécute en boucle (void loop)** : Après avoir créé une fonction setup (), qui initialise et définit les valeurs initiales, la fonction loop () fait exactement ce que son nom indique. Il boucle consécutivement, ce qui permet au programme de se changer et de répondre.

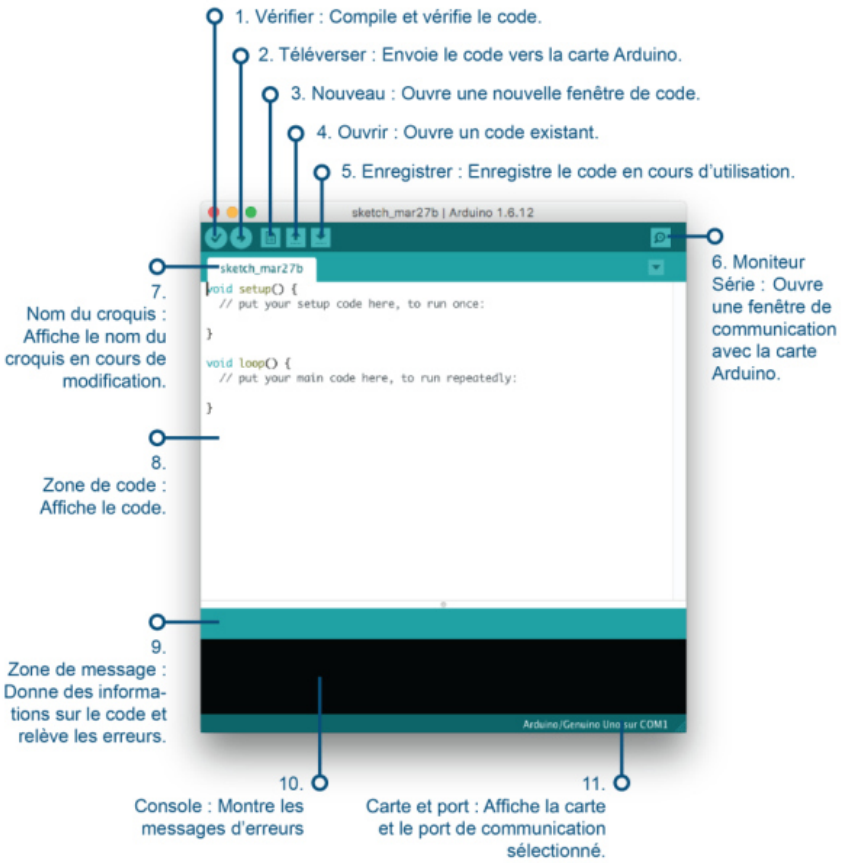


Figure □ .7 : Structure générale du programme IDE

II.4. Conclusion

Dans ce chapitre, nous avons présenté une étude générale sur les cartes programmables utilisées dans notre application ainsi que leur environnement de développement. Dans le dernier chapitre, nous allons mettre en œuvre les filtres adaptatifs présentés dans le premier chapitre dans les deux cartes choisies et comparer les résultats obtenus avec les résultats de simulation effectuée sous l'environnement Matlab Simulink.

Chapitre III
Simulation et évaluation

III.1. Introduction

Ce dernier chapitre se concentre sur l'implémentation, l'évaluation des performances et la présentation des résultats obtenus. Tout d'abord, les deux systèmes sélectionnés pour cette étude, à savoir le moteur à courant continu et le système d'annulation d'écho acoustique, sont présentés. Ensuite, les résultats de simulation sont exposés et analysés dans la deuxième partie de ce chapitre. Enfin, l'implémentation sur la carte NodeMCU8266 et la comparaison des résultats sont discutées dans la dernière section.

III.2. Présentation des systèmes à identifier

III.2.1. Moteur à courant continu

Actuellement, les machines à courant continu voient leur domaine d'application se réduire en raison des progrès significatifs des machines à courant alternatif associées à leurs électroniques de commande. L'avantage principal du moteur à courant continu, qui a contribué à son succès, réside dans la facilité de contrôle de sa vitesse.

Le moteur à courant continu à excitation séparée est largement utilisé dans les systèmes de positionnement. Dans ce cas, les circuits de l'inducteur et de l'induit sont électriquement indépendants, ce qui est également appelé machine à excitation séparée, à flux constant ou à excitation constante.

Les moteurs à courant continu sont excités par deux champs magnétiques indépendants. Ainsi, on distingue deux types de moteurs à courant continu : ceux commandés par la tension d'induit à flux constant, et ceux commandés par le flux à courant d'induit constant. Un exemple d'application des moteurs à courant continu dans l'instrumentation est l'utilisation d'un aimant permanent comme source d'un champ magnétique constant (excitation), tandis que le signal de commande est appliqué à l'induit (commande par la tension d'induit). Dans notre cas d'étude, nous allons examiner le moteur à courant continu commandé par l'induit [25], tel que représenté dans la Figure □.1.

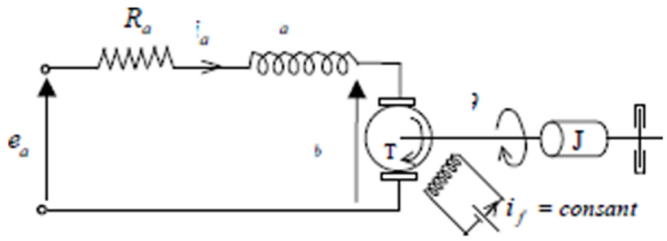


Figure □.1 : Représentation du moteur à courant continu

R_a : Résistance d'induit, ohms ;

L_a: Inductance d'induit, henrys ;

i_a: Courant d'induit, ampères ;

i_f: Courant du champ d'excitation, ampères ;

e_a: Tension de contrôle appliquée aux bornes de l'induit, volts ;

e_b: f.e.m induite, volts ;

θ : Déplacement angulaire de l'axe du moteur, radians ;

T : Couple délivré par le moteur, newton-mètre ;

J : Moment d'inertie équivalent entre le moteur et la charge relative à l'axe du moteur, Ib.ft.

f : Coefficient de frottement équivalent entre le moteur et la charge relative à l'axe du moteur, N.m/rad/sec.

Le couple T délivré par le moteur est proportionnel au produit du courant d'induit et au flux d'excitation Ψ.

$$\Psi = k_f i_f \qquad \textbf{III. 1}$$

Où k_f est une constante. Le couple T peut donc être écrit sous la forme :

$$T = k_f i_f . k_1 . i_a \qquad \textbf{III. 2}$$

Où k_1 est une constante.

Dans ce type de moteurs, le courant de champ i_f est maintenu constant, et pour un courant de champ constant, le flux devient constant, et par conséquent le couple devient proportionnel au Courant d'induit :

$$T = k . i_a \qquad \textbf{III. 3}$$

Où K est dite constante du couple moteur. Quand le rotor tourne, une tension proportionnelle au produit du flux et la vitesse angulaire est induite au niveau de l'induit. Pour un flux constant, la tension induite e_b est donc proportionnelle à la vitesse angulaire$\frac{d\theta}{dt}$.

$$e_b = k_b \frac{d\theta}{dt} \qquad \textbf{III. 4}$$

Ou k_b est la constante de la f.e.m induite.

La vitesse de ce type de moteurs est contrôlée par la tension appliquée aux bornes de L'induit e_a.

L'équation différentielle du circuit d'induit est donnée par :

$$l_a \frac{di_a}{dt} + R_a i_a + e_b = e_a \qquad \textbf{III. 5}$$

Le courant i_a produit un couple qui sera applique a une inertie et un frottement, donc :

$$J \frac{d^2\theta}{dt^2} + f \frac{d\theta}{dt} = T = k.i_a \qquad \textbf{III.6}$$

Supposons que les conditions initiales sont nulles, et prenons la transformée de Laplace des Équations 4, 5 et 6 :

$$\boldsymbol{k_b s\theta(s) = E_b(S)} \qquad \textbf{III.7}$$

$$\boldsymbol{(L_a S + R_a) I_a(S) + E_b(S) = E_a(S)} \qquad \textbf{III.8}$$

$$\boldsymbol{(Js^2 + fs)\theta(s) = T(s) = kI_a(s)} \qquad \textbf{III.9}$$

Prenons E_a(s) comme entrée et $\theta(s)$ comme sortie, on peut à partir des équations 7, 8 et 9 construire le diagramme de blocks correspondant à ce moteur comme la montre sur la figure1. La f.e.m induite e_b est prise comme un signal de réaction qui est proportionnel à la vitesse du moteur. La fonction de transfert du système est alors donnée par :

$$\frac{\theta(s)}{E_a(s)} = \frac{k}{s[L_a J s^2 + (L_a f + R_a J) S + k k_b]} \qquad \textbf{III. 10}$$

L'inductance L_a dans le circuit d'induit est généralement faible et peut être négligée. Si on la néglige, l'équation 10 devient :

$$\frac{\boldsymbol{\theta(s)}}{\boldsymbol{E_a(s)}} = \frac{\boldsymbol{k_m}}{\boldsymbol{s(T_m s+1)}} \qquad \textbf{III. 11}$$

Où : $k_m = \frac{k}{R_a f + k k_b}$ gain de moteur

$T_m \frac{R_a J}{R_a f + k k_b}$ Constante de temps du moteur

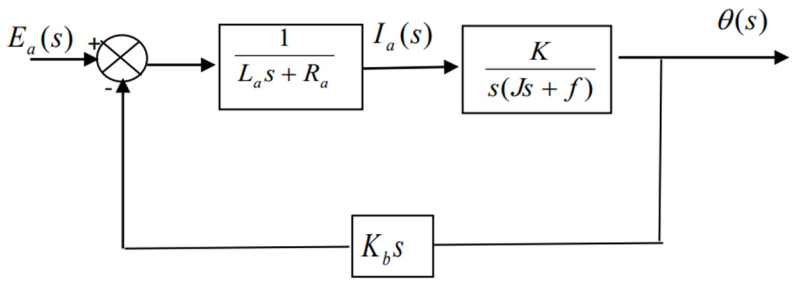


Figure □.2 : Schéma bloc du moteur à courant continu

Donc le système dont on veut identifier les paramètres est régit par le modèle suivant :

$$\boldsymbol{G(s) = \frac{Y(s)}{U(s)} = \frac{\alpha(s)}{U(s)} = \frac{k_m}{s(T_m s+1)}} \qquad \textbf{III. 12}$$

Où :

$U(s)$: Entrée du système

Y(s) = θ(s) : sortie du système

$G(s)$: Fonction de transfert continue du moteur a courant continu

On a :

$$G(s) = \frac{K_m}{s\left(T_m s+1\right)} \Rightarrow g(t) = L^{-1}\{G(s)\} \qquad \textbf{III. 13}$$

Ou L^{-1} est l'opérateur de la transformée inverse de Laplace, et g(t) la réponse impulsionnelle du moteur. Alors, on obtient :

$$\boldsymbol{g(t) = k_m - k_m e^{-\frac{t}{tm}}} \qquad \textbf{III. 14}$$

La discrétisation de la relation 13 nous donne

$$\boldsymbol{g(KT) = K_m - K_m e^{-\frac{t}{tm}}} \qquad \textbf{III. 15}$$

Prenant la transformée en Z de la relation 14

$$\boldsymbol{G(z) = Z\{g(KT)\} = \sum_{k=0}^{\infty} g(kt).Z^{-k}} \qquad \textbf{III. 16}$$

De la relation 16, on obtient le modèle discrétisé suivant

$$\boldsymbol{y(t+2) = \left(1 + e^{-\frac{t}{tm}}\right) y(t+1) - e^{-\frac{t}{tm}} y(t) + k_m \left(1 - e^{\frac{t}{tm}}\right) u(t+1)} \qquad \textbf{III. 17}$$

Posons :

$$a_1 = -\left(1 + e^{-\frac{T}{Tm}}\right)$$

$$b_1 = e^{-\frac{T}{T_m}}$$

$$c_1 = K_m\left(1 + e^{-\frac{T}{T_m}}\right)$$

Alors l'équation 17 devient

$$y(t+1) = -a_1 y(t) - b_1 y(t-1) + c_1 u(t) \qquad \textbf{III. 18}$$

$$= \phi(t)$$

Où :

$$\theta^T = [a_1 b_1 c_1]$$
$$\phi(t) = [-y(t) \quad -y(t-1) \quad u(t)]$$

Le modèle de prédiction propose prendra alors la forme suivante :

$$\hat{y}(t+1) = \hat{a}_1(t)y(t) - \hat{b}_1(t)y(t-1) + \hat{c}_1(t)u(t) \qquad \textbf{III. 19}$$
$$= \theta^T(t)\Phi(t)$$

Où : $\theta^T(t) = [\hat{a}_1(t)\ \hat{b}_1(t)\hat{c}_1(t)]$

$$\phi(t) = [-y(t) - y(t-1)\ u(t)]$$

Supposons pour la simulation que les paramètres du moteur à identifier sont $a_1 = 0.5$, $b_1 = 0.45$, $c_1 = 0.8$

III.2.2. L'écho acoustique

III.2.2.1. Définition de l'écho

L'écho acoustique est un phénomène de versions réduites et déformées du son Réfléchi et renvoyé à la source [26]. Ce problème se traduira par une mauvaise qualité du signal vocal, par conséquent, L'interlocuteur ne peut pas entendre clairement la conversation, même si Des informations importantes sont perdues. Cet écho acoustique est en fait amélioré par Ondes sonores réfléchies par les murs de la pièce et d'autres éléments présents dans la pièce.[27]
L'objectif principal de l'ingénieur est d'annuler cet écho acoustique et de fournir Problèmes d'écho acoustique ambiant anéchoïque pour les haut-parleurs pendant les conversations interférant avec la conversation des gens et dégradant la qualité Systématique [28].

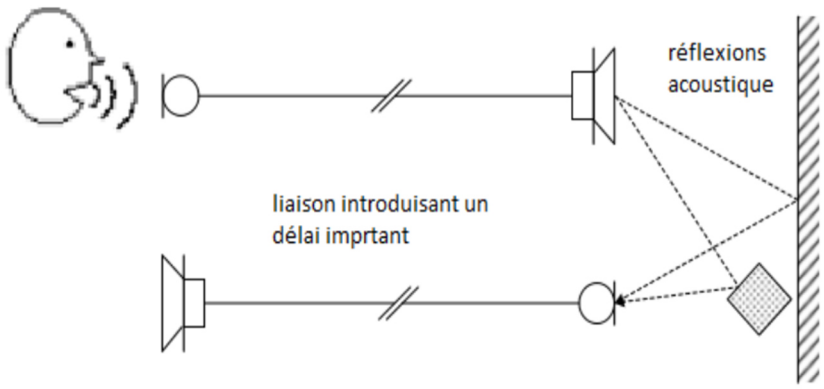


Figure □.3: Génération d'un écho acoustique

III.2.2.2. Principe de l'annulation d'écho acoustique

L'écho se produit dans le téléphone lorsque le microphone capte le son sortant à travers l'enceinte et ses multiples reflets. La figure III.4 montre une configuration générale du système d'annulation d'écho dans cette configuration. Discours Le son du haut-parleur distant est amplifié et reproduit par le haut-parleur. Une partie du signal de sortie du haut-parleur sera captée par le microphone, c'est-à-dire Direct (chemin direct) ou après réflexion (multi-trajet). L'utilisateur veut L'extrémité distante entend sa propre voix, retardée par l'aller-retour du signal dans le réseau système, sauf pour son interlocuteur.

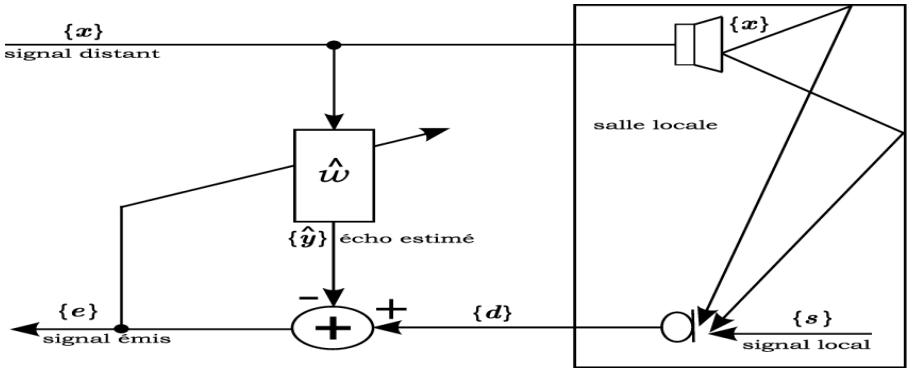


Figure □.4 : Principe de l'annulation d'écho acoustique

Cet écho peut être annulé à l'aide d'un filtre adaptatif comme la montre la Figure 4. Les filtres adaptatifs simulent le chemin d'écho entre les haut-parleurs et les microphones. La fonction de transfert estimée est utilisée pour filtrer le signal x en faisant une copie de x le signal d'écho, une copie de cet écho est soustraite du signal d du microphone Annuler l'écho. Ainsi, le signal e est transmis à l'interlocuteur sans écho. Eliminer Faire écho dans un système téléphonique n'est pas toujours une tâche facile [29].

III.3. Description des critères d'évaluation

Pour analyser les performances des algorithmes, nous définissons les critères d'évaluation suivants :

III.3.1. Mean Square Error (MSE)

Cette quantité représente la puissance du signal d'erreur :

$$\text{MSE (dB)}=10\log_{10}\{E\,[|e\,(n)\,|^2]\}$$

Où $e\,(n)$ est le signal d'erreur, et l'opérateur $E\,[.]$ indique l'espérance mathématique.

Nous appelons la valeur d'erreur quadratique moyenne convergée l'erreur quadratique moyenne minimale MMSE (Minimum MSE), et ce critère sera utilisé pour étudier la convergence de l'algorithme dans le cas d'un signal stationnaire.

III.3.2. Le critère du Désajustement (Misalignement)

Le critère misalignement (système non apparié) est un critère robuste dans l'évaluation des performances. Ce critère est calculé à partir de la distance euclidienne entre les coefficients de réponse impulsionnelle utilisés (chemin d'écho réel) et les coefficients estimés du filtre adaptatif. Le critère est défini par l'expression :

$$Misalignment(db) = 10log_{10}[\frac{||w(n)\_h||^2}{||h||^2}]$$

Où ‖w (n) – h‖ c'est la distance Euclidienne entre le vecteur des coefficients du filtre adaptatif w (n) et le vecteur de la réponse impulsionnelle utilisée h.

III.3.3. Signal to Noise Ratio (SNR)

Pour simuler le bruit ambiant, nous allons utiliser un bruit additif de type bruit blanc gaussien avec différents niveaux de rapport signal sur bruit (SNR). En fait, le SNR est défini comme :

$$SNR(db) = 10log_{10}\{\frac{E[|y(n)|^2]}{E[|b(n)|^2]}\}$$

La convergence de l'algorithme adaptatif vers la réponse impulsionnelle du système à identifier fait que l'erreur $e(n)$ est en fait égale au bruit $b(n)$.

III.4. Simulation sous Matlab

III.4.1. Identification des paramètres d'un moteur à courant continu

Les résultats de l'identification sont montrés par les courbes suivantes avec une entrée :

u= sin (0.1*i) + sin (0.2*i) + sin (0.3*i) + sin (0.4*i) + 3*sin(i)

et des paramètres de configuration sont les suivants : nombre d'échantillons = 16000, longueur du filtre = 3, LMS avec un pas d'adaptation μ de 0.01, NLMS avec un pas d'adaptation normalisé β de 0.5,et un constante C = 0.00001, et RLS avec un facteur d'oubli λ de 0.95.

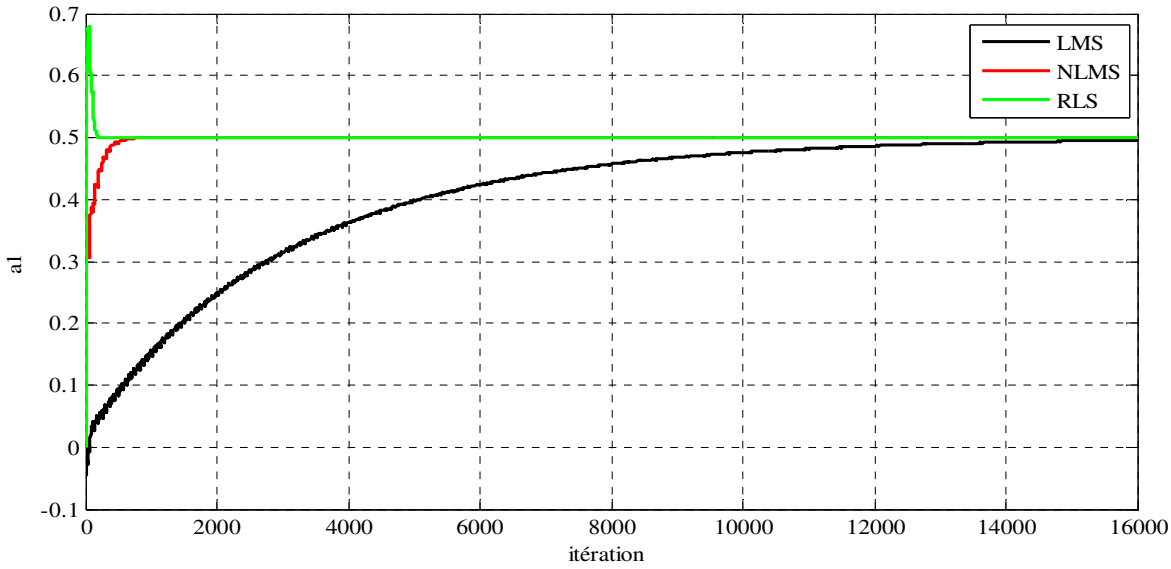


Figure □.5 : évolution du paramètre a1 pour les algorithmes LMS, NLMS et RLS

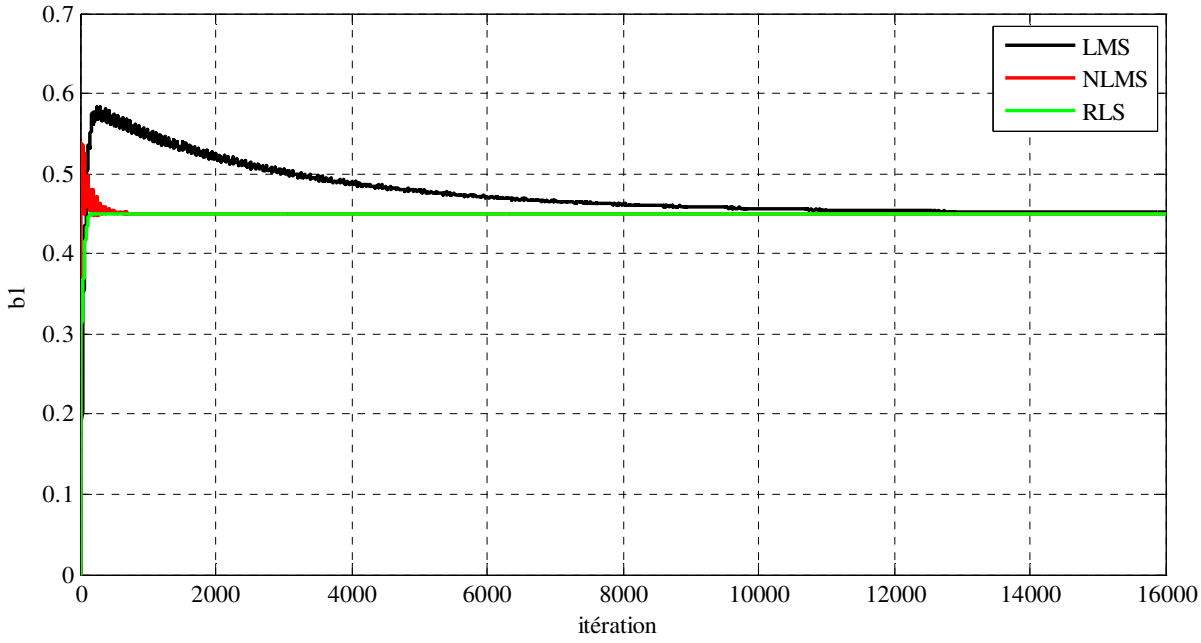


Figure □.6 : évolution du paramètre b1 pour les algorithmes LMS, NLMS et RLS

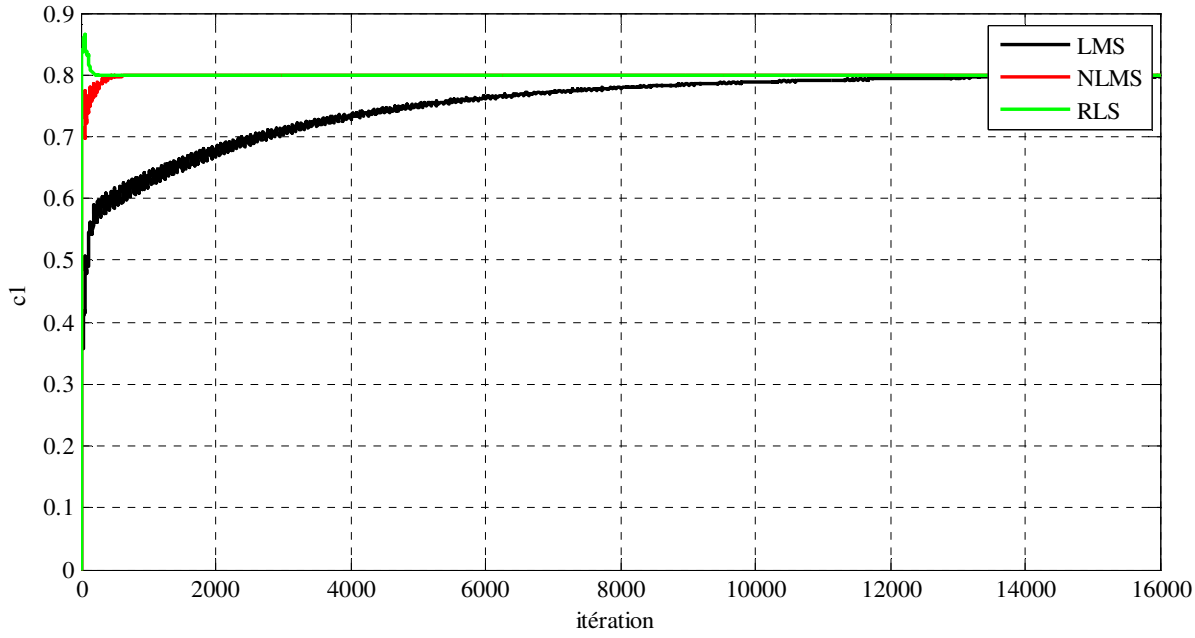


Figure □.7 : évolution du paramètre c1 pour les algorithmes LMS, NLMS et RLS

III.4.1.1. Interprétation des résultats

Paramètre a1

On remarque que la valeur estimée de a1 pour l'algorithme LMS converge de 0 vers la solution réelle après 14 000 itérations, tandis que pour l'algorithme NLMS, cette convergence s'effectue après seulement 1 000 itérations. Quant à l'algorithme RLS, il converge également de 0 vers la valeur réelle, mais il dépasse temporairement avant de se stabiliser à environ 500 itérations.

Paramètre b1

On remarque que la valeur estimée de b1 pour l'algorithme LMS converge vers la solution réelle après 10 500 itérations, mais dépasse cette valeur avant de se stabiliser. Pour l'algorithme NLMS, la valeur estimée dépasse également la valeur réelle avant de se stabiliser après environ 400 itérations. En revanche, pour l'algorithme RLS, la convergence de la valeur estimée de b1 vers la solution réelle s'effectue après seulement 150 itérations.

Paramètre c1

On remarque que la valeur estimée de c1 pour l'algorithme LMS converge après 12 000 itérations, tandis que pour l'algorithme NLMS, cette convergence s'effectue après seulement 400 itérations. En revanche, pour l'algorithme RLS, la valeur estimée de c1 dépasse la valeur réelle avant de se stabiliser après environ 200 itérations.

Comparaison

- L'algorithme LMS est simple mais peut nécessiter plus d'itérations pour identifier les paramètres d'un moteur à courant continu.
- L'algorithme NLMS offre une adaptation automatique de pas d'adaptation, ce qui permet d'identification des paramètres rapide.
- L'algorithme RLS est complexe en termes de calculs mais identifier les paramètres avec moins d'itérations.

III.4.2. Echo acoustique

Dans cette partie, nous avons utilisé un signal audio de longueur N=40000 comme entrée, avec une longueur de filtre de 128 et un bruit de 40000. De plus, nous avons ajouté un rapport signal/bruit (SNR) de 45 dB.

La réponse impulsionnelle : La réponse impulsionnelle d'un système décrit comment il réagit aux différentes fréquences et influences temporelles, et elle peut être utilisée pour caractériser les propriétés acoustiques et temporelles du système.

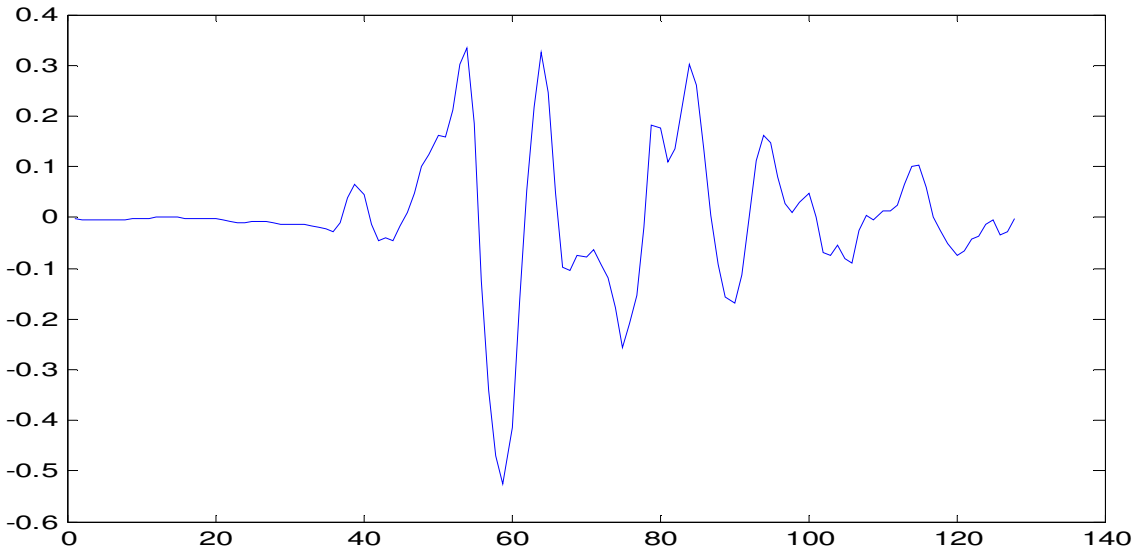


Figure □.8 : la réponse impulsionnelle

Signal parole: Un signal de parole est un signal acoustique produit par les cordes vocales d'un être humain lorsqu'il parle**.**

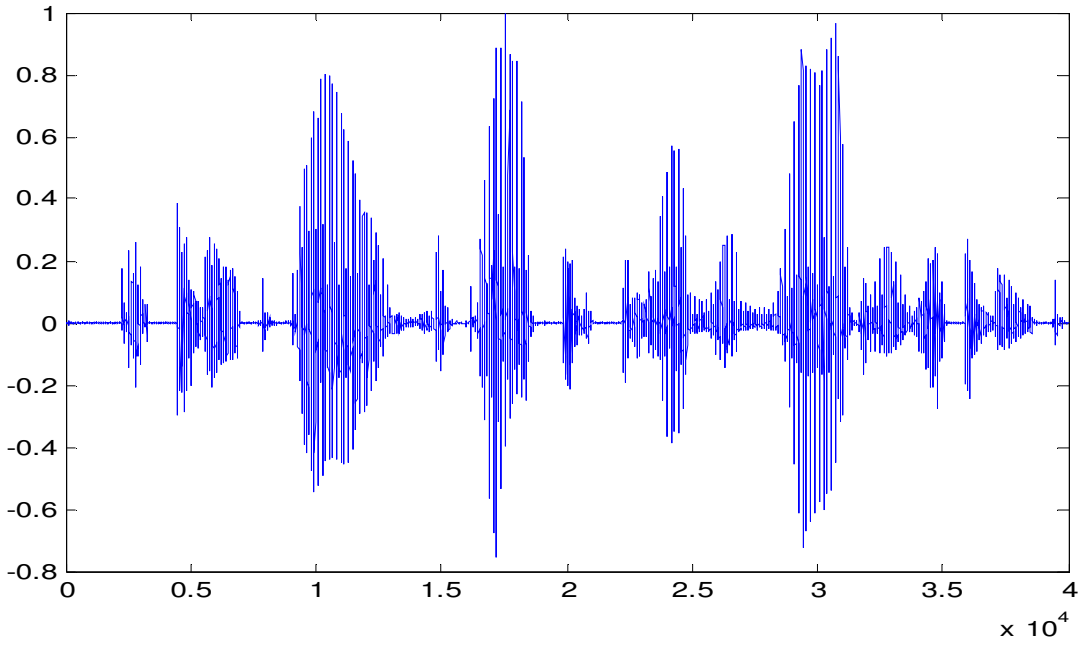


Figure □.9 : signal parole

Bruit : Le bruit est généralement considéré comme un signal indésirable ou perturbateur qui interfère avec un signal utile. En acoustique, le bruit se réfère à tout son non désiré ou non intentionnel présent dans un environnement

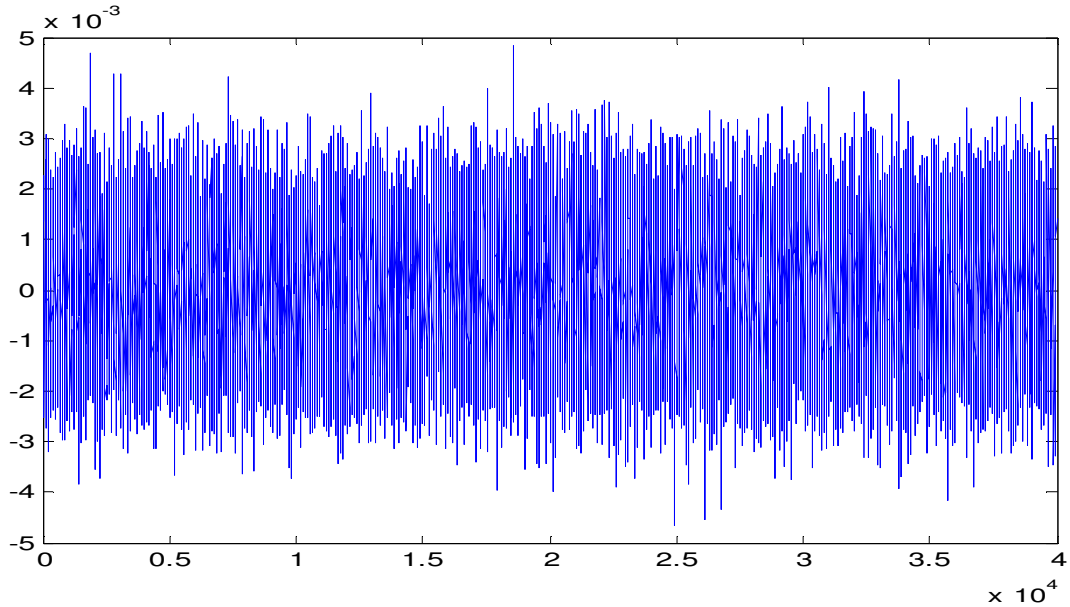


Figure □.10 : bruit

III.4.2.1. Algorithme LMS

Avec un pas d'adaptation de 0.1.

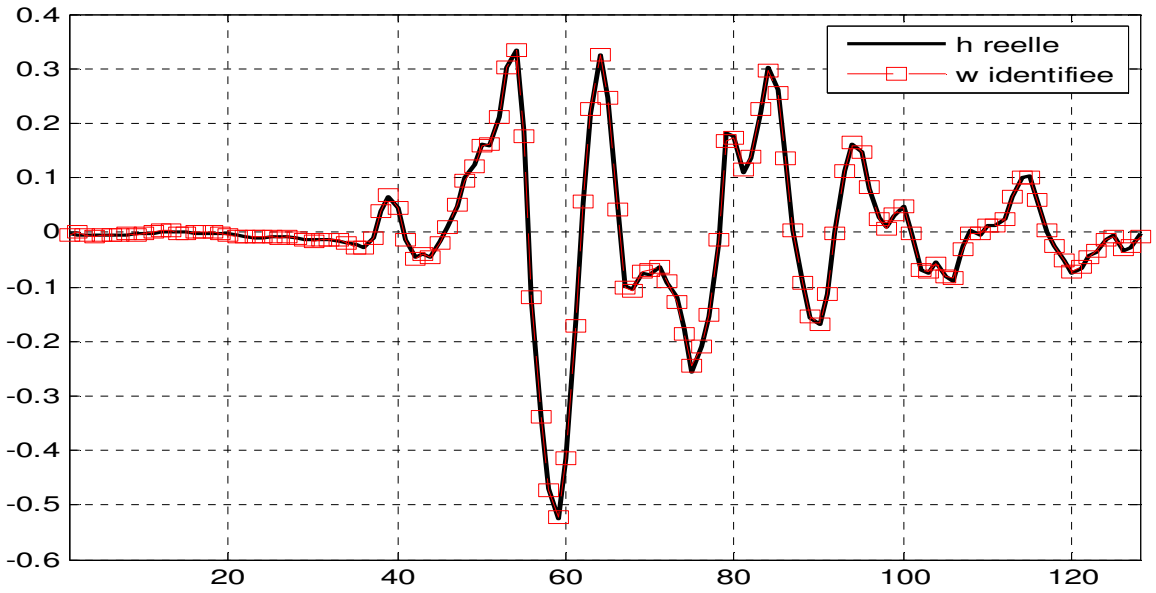


Figure .11 : la réponse impulsionnelle réelle h(N) et estime w(N) de l'algorithme LMS

La figure .11 elle montre que les allures les réponses impulsionnelles réelle et estimée sont superposées. Cela indique que l'algorithme LMS a réussi à estimer de manière précise la réponse impulsionnelle du système. Cela signifie que la sortie estimée du filtre adaptatif suit de près la sortie réelle du système, ce qui est un bon indicateur de l'efficacité de l'algorithme

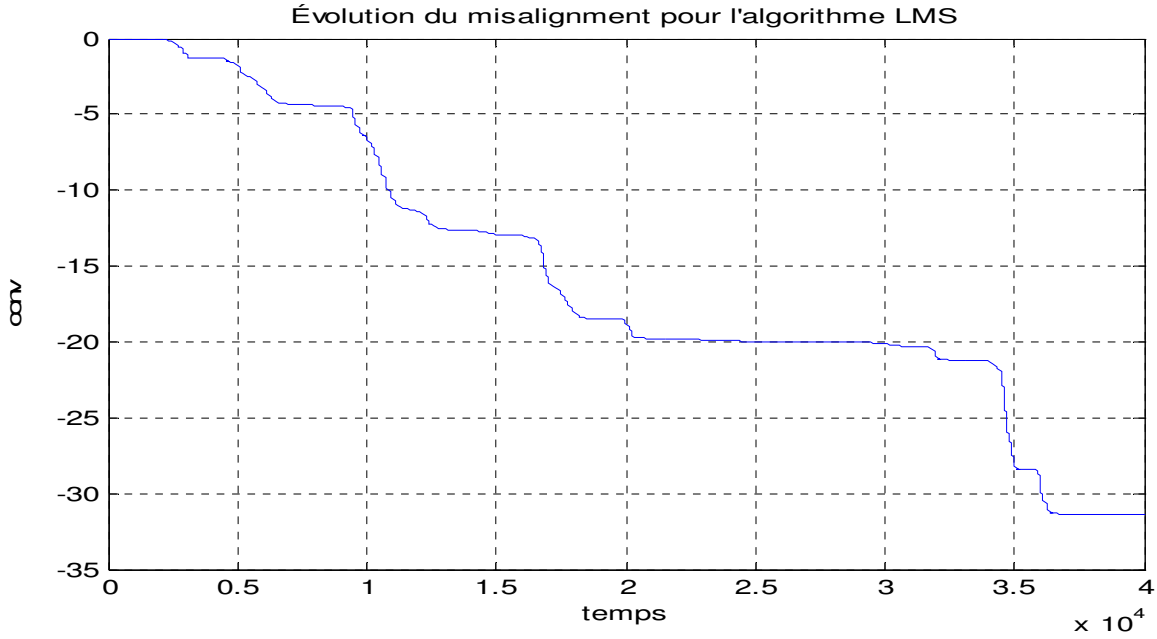


Figure .12 : évolution du misalignment pour l'algorithme LMS

Le résultat d'évaluation de la convergence est montre dans la figure. On peut bien remarquer la bonne convergence du LMS, le niveau atteint à la fin de la simulation est de -32db.

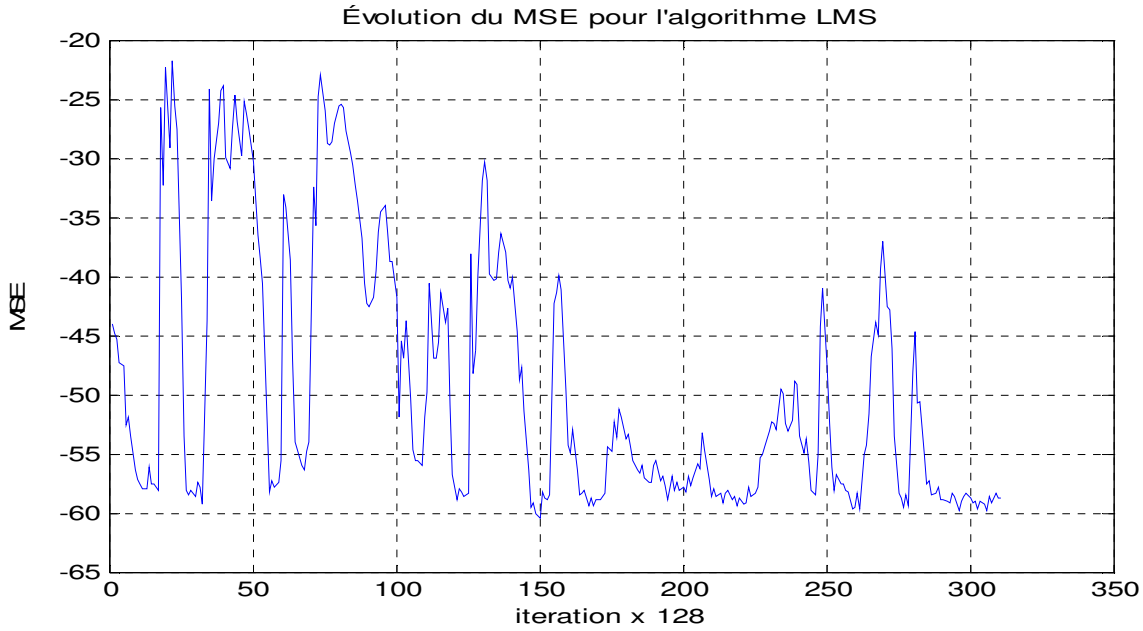


Figure □.13 : évolution du MSE pour l'algorithme LMS

III.4.2.2. Algorithme NLMS

Avec un pas d'adaptation normalise de 1.2 et un constante c=0.00001

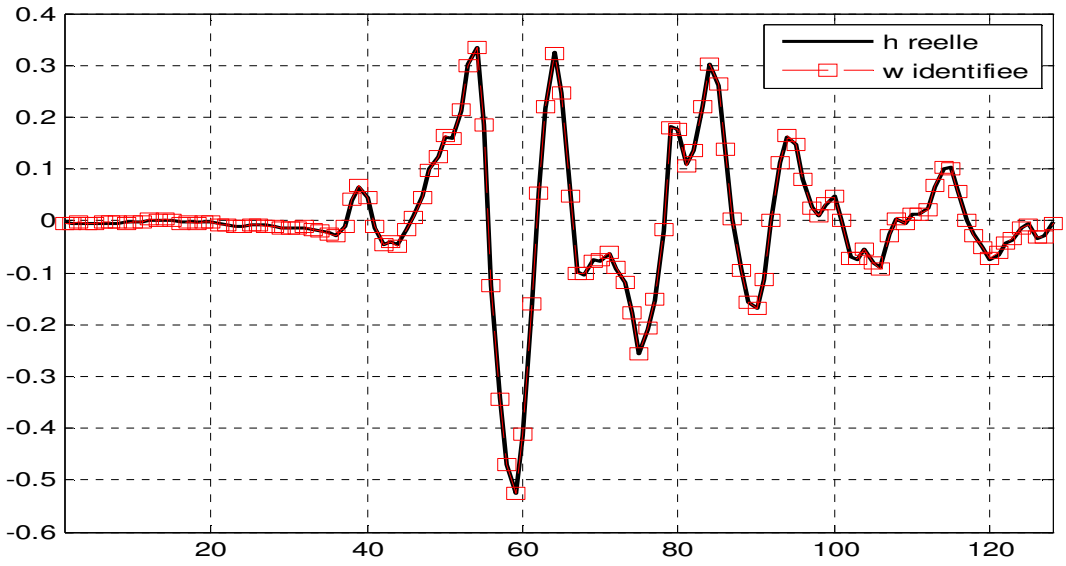


Figure □.14 : la réponse impulsionnelle réelle h(N) et estime w(N) de l'algorithme NLMS

La figure □.14 elle montre que les allures les réponses impulsionnelles réelle et estimée sont superposées. Cela indique une bonne capacité de l'algorithme à estimer de manière précise la réponse impulsionnelle réelle du système, ce qui témoigne de sa performance et de son efficacité dans l'identification des paramètres.

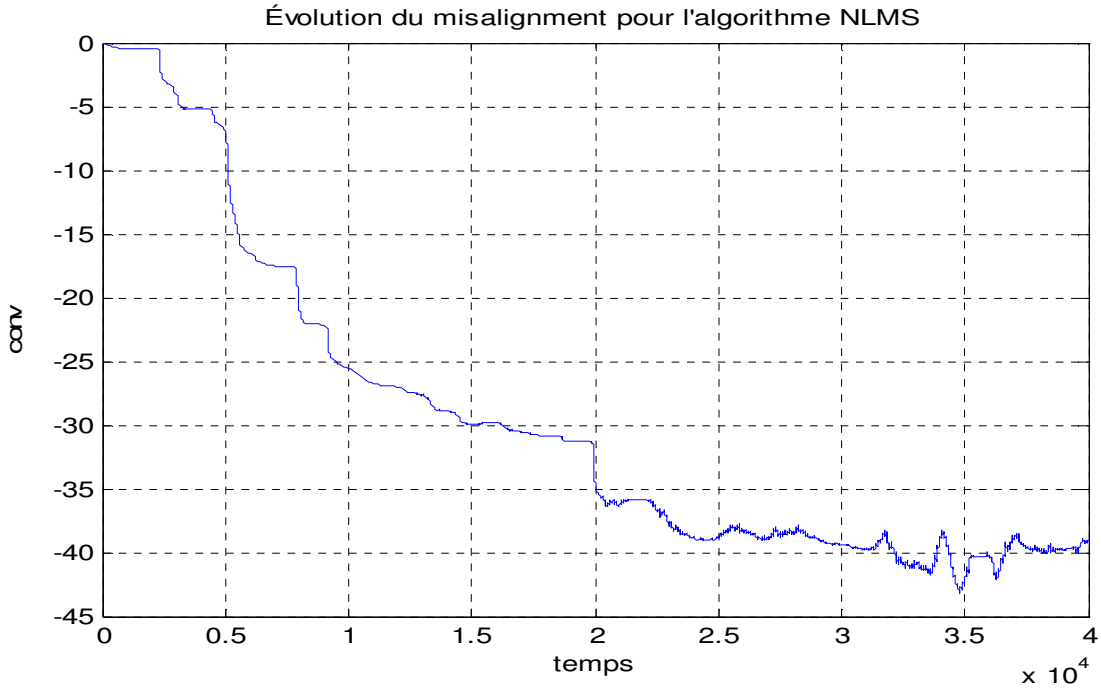


Figure □.15 : évolution du misalignment pour l'algorithme NLMS

Le résultat d'évaluation de la convergence est montre dans la figure. On peut bien remarquer la bonne convergence du NLMS, le niveau atteint à la fin de la simulation est de -40db.

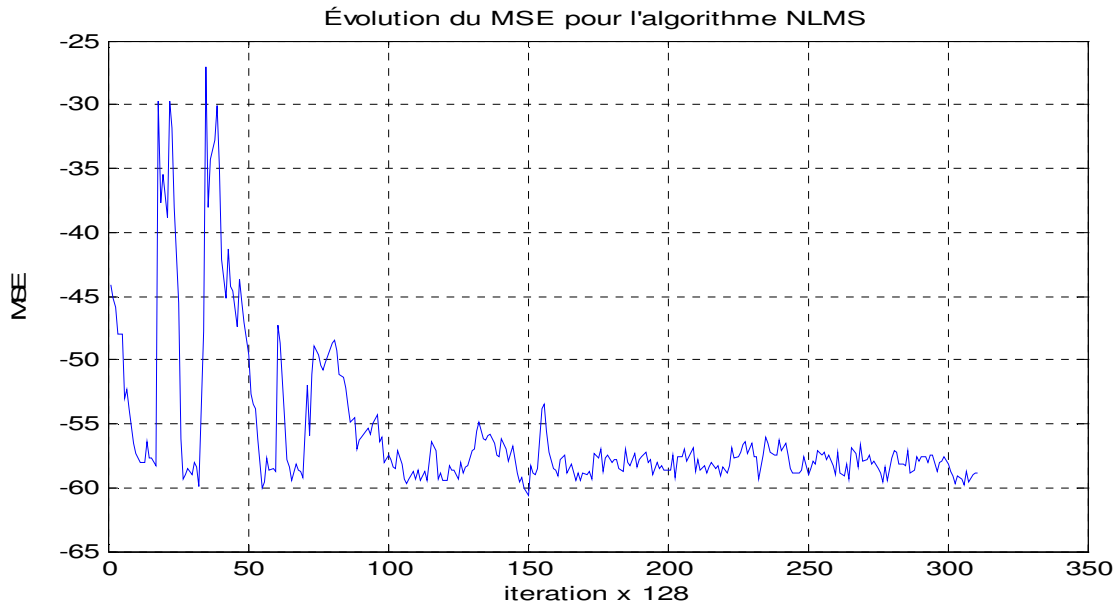


Figure □.16 : évolution du MSE pour l'algorithme NLMS

III.4.2.3. Algorithme RLS

Avec une facture d'oubli λ de 1 et un paramètre de régularisation δ de 0.05

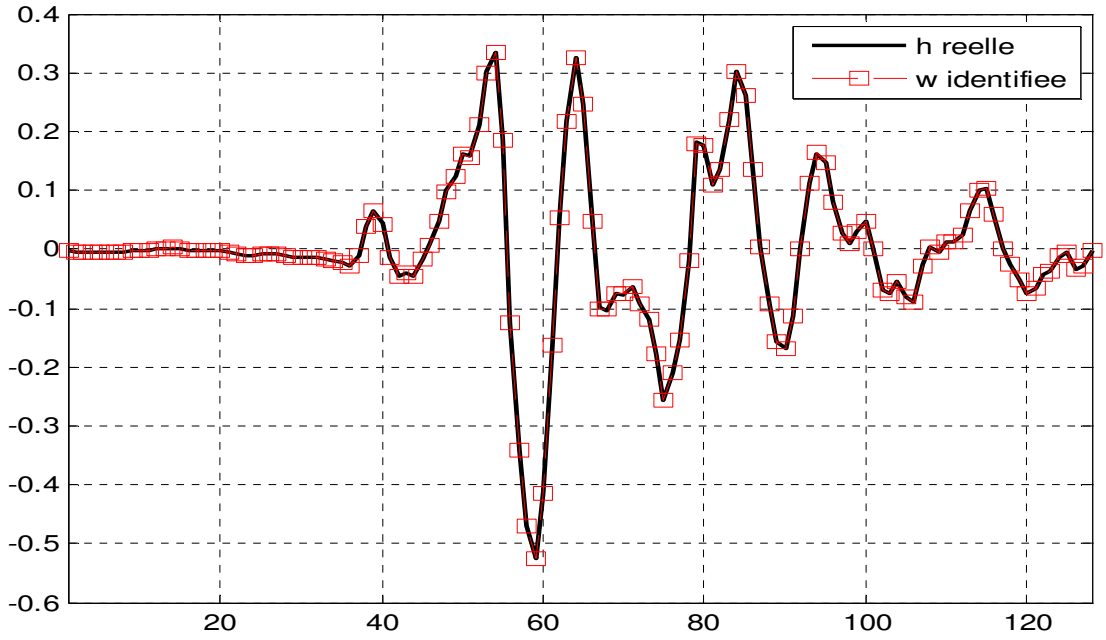


Figure □.17 : la réponse impulsionnelle réelle h(N) et estime w(N) de l'algorithme RLS

La figure □.17 elle montre que les allures les réponses impulsionnelles réelle et estimée sont superposées. Cela témoigne de la performance et de l'efficacité de l'algorithme dans l'identification des paramètres du système. Cela indique également sa capacité à capturer les

caractéristiques essentielles du système et à estimer de manière précise la réponse impulsionnelle réelle

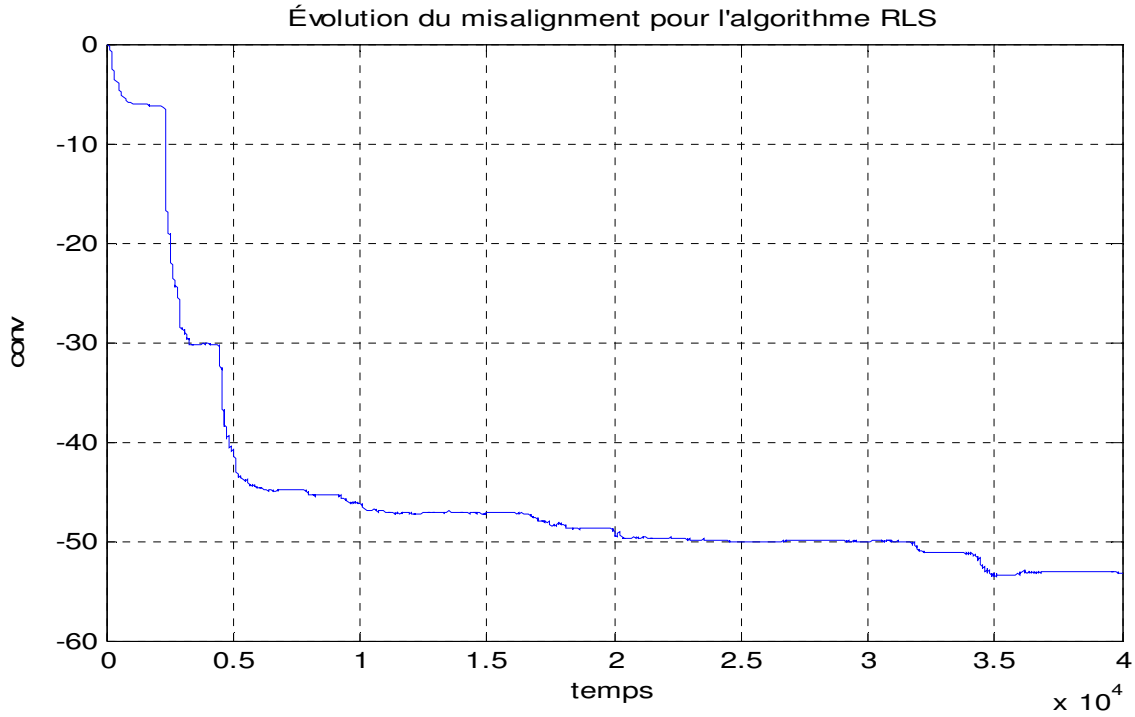


Figure □.18 : évolution du misalignment pour l'algorithme RLS

Le résultat d'évaluation de la convergence est montre dans la figure. On peut bien remarquer la bonne convergence du RLS, le niveau atteint à la fin de la simulation est de -52db.

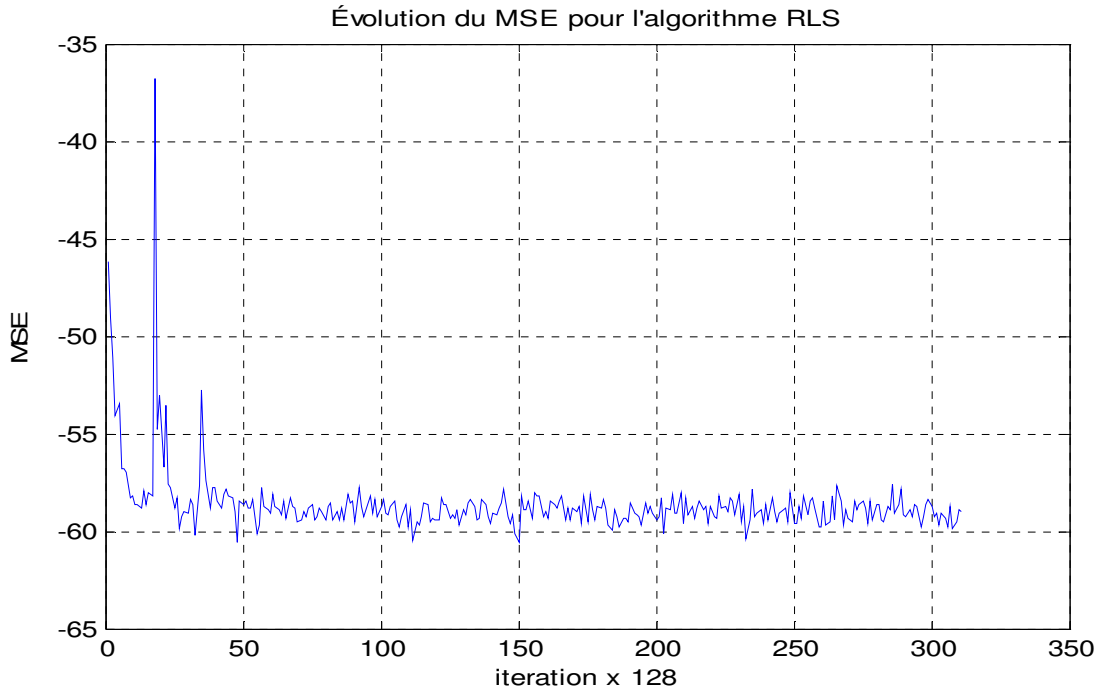


Figure □.19 : évolution du MSE pour l'algorithme RLS

III.4.2.4. Comparaison entre les différents algorithmes

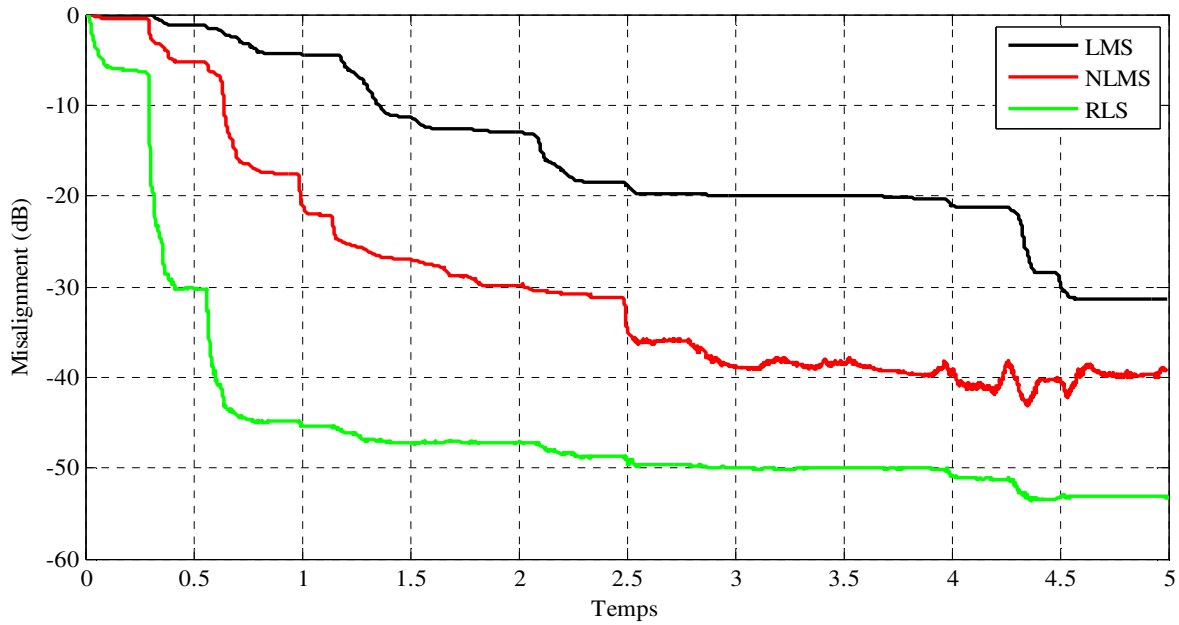


Figure □.20 : Misalignment (dB) en fonction de temps pour la comparaison des vitesses de convergences des algorithmes LMS, NLMS et RLS

L'algorithme LMS est simple mais converge généralement plus lentement, tandis que l'algorithme NLMS peut être plus rapide que LMS grâce à sa normalisation adaptative du pas

d'apprentissage. Cependant, l'algorithme RLS est généralement considéré comme le plus rapide en termes de convergence en raison de son approche récursive et de sa gestion efficace des signaux d'entrée corrélés.

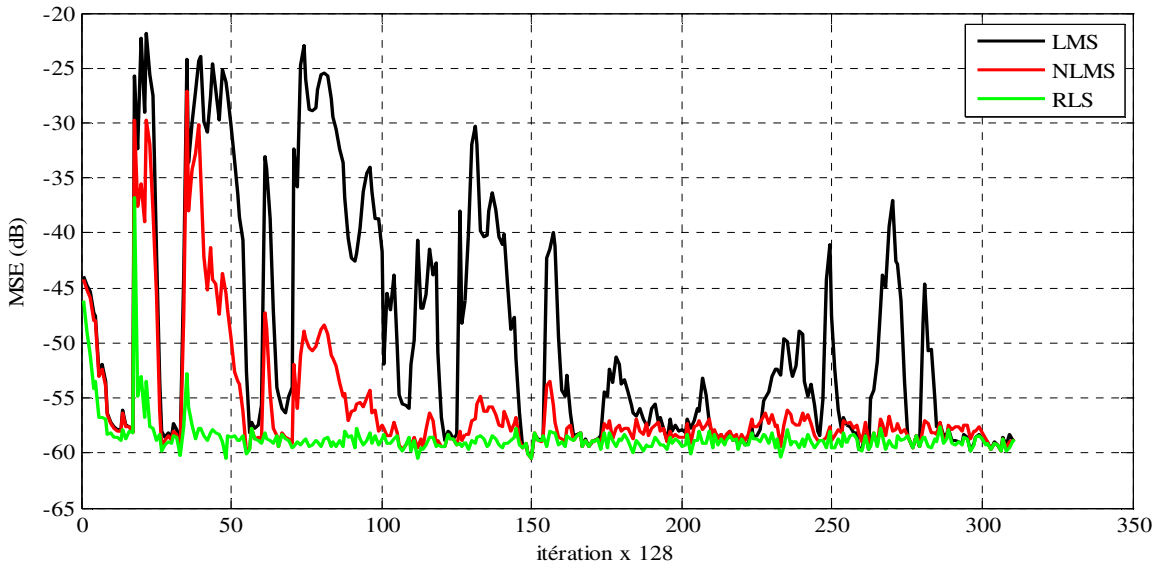


Figure □.21 : MSE en fonction d'un nombre d'itération pour la comparaison des algorithme LMS, NLMS et RLS

On remarque que MSE obtenue par l'algorithme LMS est généralement plus élevée que celle des autres algorithmes, tandis que l'algorithme NLMS présente une erreur MSE inférieure à LMS avec un nombre d'itérations qui se stabilise. De plus, l'algorithme RLS a tendance à atteindre une erreur MSE très faible et à se stabiliser rapidement.

Tableau □.1 : la comparaison entre LMS, NLMS et RLS

	LMS	**NLMS**	**RLS**
Convergence	Lente	Rapide	Plus rapide
Stabilité	Moins stable	Plus stable	Très stable
Simplicité	Très simple	Simple	Complexe
Robustesse	Moins robuste	Robuste	Plus robuste
Poursuit	Moins	Bonne	Très bonne
Occupation de mémoire	Faible	Moyenne	Élevé

III.5. Implémentation des algorithmes dans la carte NodeMCU

Dans le but de mettre en pratique les algorithmes précédents nous allons les implémenter sur la carte NodeMCU et analyser les résultats puis procéder à une comparaison entre la simulation et l'expérimentation et pour se faire, nous aurons besoins de la carte ainsi que de l'environnement de développement comme montre la Figure III.22.

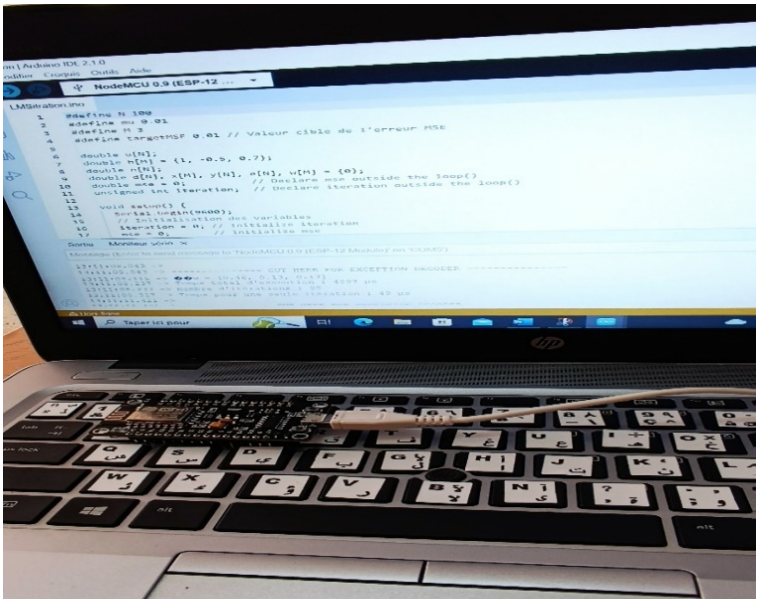


Figure III.22 : Partie expérimentale

III.5.1. Evaluation des performances

III.5.1.1. La vitesse de calcul

III.5.1.1.1. Moteur à courant continu

Nous avons implémenté les algorithmes LMS, NLMS et RLS sur une plateforme Arduino pour contrôler un moteur à courant continu. L'objectif de notre étude était de calculer le temps nécessaire pour une seule itération et le nombre d'itérations requis pour atteindre une valeur d'erreur MSE de 0.01.

Tableau □.2 : comparaison des temps d'exécution et du nombre d'itérations des algorithme LMS, NLMS et RLS

Algorithme	Temps d'une Itérations (µs)	Nombre d'itérations	Temps nécessaire pour l'identification (µs)
LMS	44	5434	229096
NLMS	63	1449	91287
RLS	110	1256	138160

L'algorithme LMS a montré le temps d'itération le plus court parmi les trois, avec une valeur de 44 µs. Cependant, il a nécessité un nombre d'itérations élevé de 5434 pour atteindre l'erreur MSE cible. Le temps total requis pour l'identification a été mesuré à 229096 µs.
L'algorithme NLMS, nous avons observé un temps d'itération légèrement supérieur, évalué à 63 µs, par rapport à l'algorithme LMS. Et le nombre d'itérations inférieur est 1449. Le temps total pour l'identification s'est élevé à 91287 µs.

L'algorithme RLS, nous avons constaté qu'il affichait le temps d'itération le plus long, avec une valeur de 110 µs. Et il a nécessité un nombre d'itérations relativement faible, soit 1256. Le temps total pour l'identification s'est établi à 138160 µs.

Lors de cette comparaison, il convient de noter que l'algorithme LMS présente un avantage en termes de temps d'itération moyen, bien qu'il exige un nombre d'itérations considérablement plus élevé. Parallèlement, l'algorithme NLMS offre un compromis intéressant entre le temps d'itération et le nombre d'itérations requis, avec un temps moyen légèrement supérieur, mais un nombre d'itérations plus faible que celui de l'algorithme LMS. Enfin, l'algorithme RLS se caractérise par le temps d'itération moyen le plus long, mais il nécessite un nombre d'itérations relativement réduit.

III.5.1.1.2. Echo acoustique

Pour que le système d'écho fonctionne en temps réel, l'exécution du programme doit s'effectuer dans un intervalle de temps inférieur à la période d'échantillonnage. Une fréquence d'échantillonnage maximale de 8 kHz pour les signaux de parole correspond à une période d'échantillonnage de 125 µs. Cet intervalle de temps est la durée maximale tolérable pour un fonctionnement en temps réel.

Pour vérifier la contrainte temps réel, nous devons calculer le temps nécessaire pour une seule itération des algorithmes LMS, NLMS et RLS. D'après des résultats de calcul, le temps pour une seule itération de LMS est de 79 µs, celui de NLMS est de 92 µs et celui de RLS est de 157 µs. Ce qui dépasse la période d'échantillonnage.

III.5.1.2. La précession de calcul

Dans le but d'évaluer la précision du calcul des résultats obtenus par MATLAB et DevC++ pour les algorithmes LMS, NLMS et RLS, appliqués aux systèmes de moteurs à courant continu et d'écho acoustique, nous avons calculé les erreurs relatives. Enfin, nous avons réalisé une comparaison précise entre MATLAB et DevC++ pour déterminer les résultats les plus précis.

III.5.1.2.1. Moteur à courant continu

Pour le moteur à courant continu, nous avons calculé l'écart absolu pour les paramètres a1, b1 et c1 pour les trois algorithmes LMS, NLMS et RLS. Ensuite, nous avons calculé l'erreur moyenne pour chaque algorithme, à la fois avec DevC++ et MATLAB. Enfin, nous avons réalisé une comparaison entre les deux pour évaluer leur performance respective.

Tableau □.3 : les résultats obtenus pas DevC++

Algorithmes	Valeur calculée	Valeurs réelles	Ecart absolue	Erreur moyenne
LMS	a1= 0.317407 b1=0.609979 c1=0.789826	a1= 0.5 b1= 0.45 c1= 0.8	a1=0.182593 b1=0.159979 c1=0.010174	0.117583
NLMS	a1=0.422996 b1=0.527004 c1=0.8	a1= 0.5 b1= 0.45 c1= 0.8	a1=0.077004 b1=0.072007 c1=0	0.049
RLS	a1=0.49613 b1=0.453866 c1=0.799998	a1= 0.5 b1= 0.45 c1= 0.8	a1=0.00387 b1=0.003866 c1= $2*10^{-6}$	0.00257

Tableau □.4 : les résultats obtenus par MATLAB

Algorithmes	Valeur calculée	Valeurs réelles	Ecart absolue	Erreur moyenne
LMS	a1= 0.1494 b1=0.5509 c1= 0.6324	a1= 0.5 b1= 0.45 c1= 0.8	a1=0.3506 b1=0.1009 c1=0.1676	0.2063
NLMS	a1= 0.4997 b1= 0.4501 c1= 0.7999	a1= 0.5 b1= 0.45 c1= 0.8	a1=0.0003 b1=0.0001 c1=0.0001	0.00016
RLS	a1= 0.5 b1= 0.45 c1=0.8	a1= 0.5 b1= 0.45 c1= 0.8	a1=0 b1=0 c1=0	0

Après avoir analysé les résultats, nous avons constaté que l'algorithme RLS offre une meilleure précision de calcul à la fois dans DevC++ et MATLAB, avec une erreur moyenne significativement plus faible par rapport aux algorithmes LMS et NLMS.

On remarque que l'erreur moyenne donnée par Dev C++ est généralement plus élevée par rapport à celle donnée par MATLAB.

III.5.2.1.2. Echo acoustique

$$\text{Erreur relative} = \left|\frac{les\ coefficients\ de\ matlab - les\ coefficients\ de\ devc++}{les\ coefficients\ de\ matlab}\right| * 100$$

Tableau □.5 : analyse comparative des erreurs relatives des algorithme LMS, NLMS et RLS en fonction du nombre d'itérations

Nombre d'itérations	LMS	NLMS	RLS
5000	0.37%	0.32%	0.75%
10000	0.15%	0.28%	0.46%
40000	0.45%	0.45%	0.46%

Il est remarquable que l'erreur relative pour les algorithmes soit très faible. Cela indique que DevC++ est une plateforme adaptée pour implémenter et calculer les filtrages adaptatifs.

III.5.2. Comparaison de précision entre MATLAB et DevC++

MATLAB utilise par défaut une précision double (64 bits), tandis que C++ peut utiliser par défaut une précision simple (32 bits).

MATLAB est réputé pour offrir une précision élevée dans les calculs numériques. Il utilise des bibliothèques mathématiques avancées et des algorithmes optimisés pour garantir des résultats précis. MATLAB gère automatiquement les types de données et les erreurs d'arrondi, ce qui contribue à minimiser les pertes de précision.

DevC++ est un environnement de développement pour le langage C++, qui offre une flexibilité et un contrôle supplémentaires sur les calculs. La précision des calculs dans DevC++ dépend de la manière dont le code est écrit et des bibliothèques utilisées. Il est essentiel de prendre en compte les types de données et les erreurs d'arrondi pour maintenir une précision élevée.

III.6. Conclusion

Dans ce chapitre, nous avons mis en œuvre des algorithmes de filtrage adaptatif appliqués à deux systèmes différents : un moteur à courant continu et un écho acoustique. L'objectif de cette mise en œuvre était d'évaluer les performances de ces algorithmes afin de les comparer.

Dans la simulation sous Matlab des deux systèmes, on a pu constater que le RLS est le meilleur pour identifier les paramètres d'un moteur à courant continu. Il identifie rapidement avec moins d'itérations, contrairement à LMS qui nécessite davantage d'itérations, et NLMS

qui présente une convergence moyenne. Pour ce qui est de l'écho acoustique, on constate également que RLS converge plus rapidement avec une erreur quadratique moyenne plus faible.

Dans la partie d'implémentation du filtrage sur une carte NodeMCU8266 afin d'évaluer les performances de vitesse de calcul des algorithmes pour les deux systèmes, ainsi que la précision de calcul des systèmes, nous effectuons une comparaison entre Matlab et Dev C++ pour déterminer lequel offre la meilleure précision de calcul. Nous constatons que Matlab offre une meilleure précision de calcul à cause du format de donnée utilisée qui est de 64 bits.

Conclusion générale

Ce travail a exploré l'implémentation des filtres adaptatifs pour l'identification des systèmes linéaires. L'étude s'est concentrée sur la simulation et l'implémentation de deux systèmes spécifiques : l'identification de la fonction de transfert d'un moteur à courant continu et un système d'annulation d'écho acoustique sur une carte NodeMCU. Trois algorithmes de filtrage adaptatif, à savoir LMS, NLMS et RLS, ont été utilisés pour évaluer leurs performances en termes de précision de calcul et de vitesse de traitement.

À travers la mise en œuvre et l'analyse de ces algorithmes, plusieurs conclusions clés ont émergé. Tout d'abord, l'algorithme LMS a démontré un bon équilibre entre simplicité et performances, le rendant adapté aux applications en temps réel avec des ressources de calcul limitées. L'algorithme NLMS a offert des performances améliorées en ajustant de manière adaptative la taille du pas, garantissant une meilleure convergence dans les environnements non stationnaires. En revanche, l'algorithme RLS a présenté une précision supérieure au prix d'une complexité de calcul accrue, le rendant plus adapté aux applications hors ligne ou à haute performance.

L'implémentation des filtres adaptatifs en matériel, notamment sur la carte NodeMCU, a démontré la faisabilité du traitement du signal adaptatif en temps réel dans les systèmes embarqués. Cette étude contribue à la compréhension et à l'application pratique des filtres adaptatifs pour l'identification des systèmes linéaires.

En perspective, ce travail reste ouvert aux améliorations à savoir :

- La mise en œuvre d’autres algorithmes de filtrage adaptatifs.
- L’utilisation des versions rapides de l’algorithme RLS afin de surmonter son problème de complexité de calcul.
- L’utilisation d’autres plateformes et calculateurs spécialité pour avois de meilleures performances notamment pour des systèmes plus complexes

Référence biographique

Références bibliographiques

[01] F. BERCHER & P. JARDIN, “Introduction au filtrage adaptatif“, ESIEE Paris, 2003.

[02] Mathieu. POULIQUEN, “Introduction au filtrage adaptatif et à l’égalisation“, ENSI Caen année 2008-2009.

[03] Jacob BENESTY, “Traitement des signaux numériques-II Filtrage adaptatif et analyse spectrale “, 2005.

[04] Jamal EL MHAMDI1, Fakhita REGRAGUI2 & Mimoun HARNAFI3, " Traitement adaptatif appliqué au signal sismique", Bulletin de l’Institut Scientifique, Rabat, section Sciences de la Terre, 2008

[05] BERRABAH, randa et MOHAMED MAAMAR, fadia. Filtre adaptatif IPNLMS à complexité réduite basé sur la mise à jour sélective de blocs de coefficients .88p. Mémoire de Master : Réseaux & Télécommunication : Blida, Université SAAD DAHLAB.2019/2020

[06] G. Binet, " Filtrage adaptatif introduction ", Université de Caen

[07] G.Bendjabar et M.Habib Boukharcha, "Soustraction de bruit : Approche linéaire et non linéaire ", Mémoire d’ingénieur d’état en génie électrique EMP 2002.

[08] Mathieu Pouliquen, « Introduction au filtrage adaptative et à l’égalisation », Cours de traitement numérique de signal.

[09] S.Haykin – Adaptative Filter theory. 4 the ed., Prentice Hall.

[10] Wiem Jebri Jemaii, Kamel Abderrahim, Faizy Msahli, ‘Comparaison de deux méthodes adaptatives LMS & RLS du modèle de Volterra’, The seventh international conference on Sciences and Techniques of Automatic control STA, December 17-19 2006, Tunisia

[11] FERRANE, Khaled et SADOUNI, IBTISSEM. Filtrage adaptatif pour l’annulation d’écho acoustique dans la communication mains libres .46p. Mémoire de Master : Génie électrique : Alger, Faculté des sciences et des sciences application :2017.

[12] BENDIB, Sonia-Sabrina, Méthodologie de Conception de Systèmes Embarqués Temps Réel, thèse de doctorat, université de Batna 2, 2019.

[13] AICHOUCHI, Manar El houda et CHANANE, Nour El houda. Conception et réalisation d'un système embarqué pour mesurer d'une ligne téléphonique. Mémoire de projet de fin d'études. Université Saad Dahleb Blida1, 2018.

[14] http://igm.univ-mlv.fr/~dr/XPOSE2002/SE/architecture.html

[15] https://www.commentcamarche.net/informatique/composants/26177-comment-choisir-un-processeur/

[16] https://www.techno-science.net/glossaire-definition/Microcontroleur-page-2.html

[17] https://www.javatpoint.com/microprocessor-vs-microcontroller

[18] LECHALUPE J. 2014. Cours d'initiation à Arduino, Université Paul Sabatier Cours d'initiation à Arduino. Cours. Université Paul Sabatier.

[19] ARDUINO FRANCE. 2019. Arduino Uno : Avantages, inconvénients, utilisation et Fonctionnement.

[20] Fethi, M. Abdellah. Conception d'un système immotique "smart Building" pour la société CNAS. Mémoire de Fin d'Études. Université Saad Dahleb de Blida, 2019.

[21] Handson Technology, ESP8266 NodeMCU WiFi Development Board, User manuel "Getting Started User Guide", consulter le : 19.5.2023 sur :https://handsontec.com/

[22] Université de Toulon ,FicheESP8266, consulter le 20.05.2023, sur :http://bravo.univ-tln.fr/er/GE2/Fiche_ESP8266.pdf

[23] LOCODUINO B. 2021. Bien utiliser l'IDE d'Arduino. Article en ligne disponible sur : https://www.locoduino.org/spip.php?article210.

[24] IDE Arduino. 2019. Installation et utilisation. Disponible sur :

https://www.arduino-france.com/tutoriels/ide-arduino-installation-et-utilisation/

[25] Dahmani, Nabil, et Berkouider, Houcine. "Identification paramétrique de la machine à courant continu." Mémoire de fin d'études en conversion et gestion de l'énergie électrique, Université de Bouira, 2016.

[26] K. Murano, S. Unagami, and F. Amano, "Echo cancellation and applications," *IEEE Communications Magazine,* vol. 28, no. 1, pp. 49-55, 1990.

[27] C. Y. Tchassi, "Acoustic echo cancellation for single-and dual-microphone devices : application to mobile devices," PhD thesis, Télécom ParisTech, 2013.

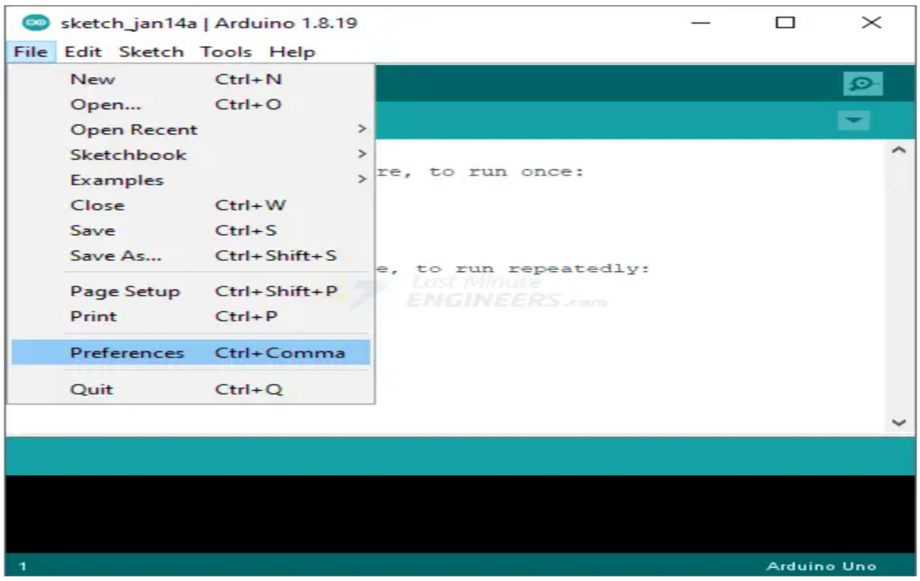
[28] C. Beaugeant, R. L. B. Jeannès, P. Scalart, and G. Faucon, "Synthèse sur la réduction conjointe de bruit et d'écho pour les systèmes mains-libres," in *Annales des télécommunications*, 2000, pp. 538-552.

[29] Hela DAASSI-GNABA, 'Annulation d'écho acoustique centralisée dans les réseaux radio-mobiles',121 pages, Thèse présentée en vue de l'obtention du grade de Docteur, Mathématiques et Informatique : UNIVERSITE RENE DESCARTES, PARIS, 2006

Annexe

Guide d’installation de la carte NodeMCU dans l’IDE Arduino.

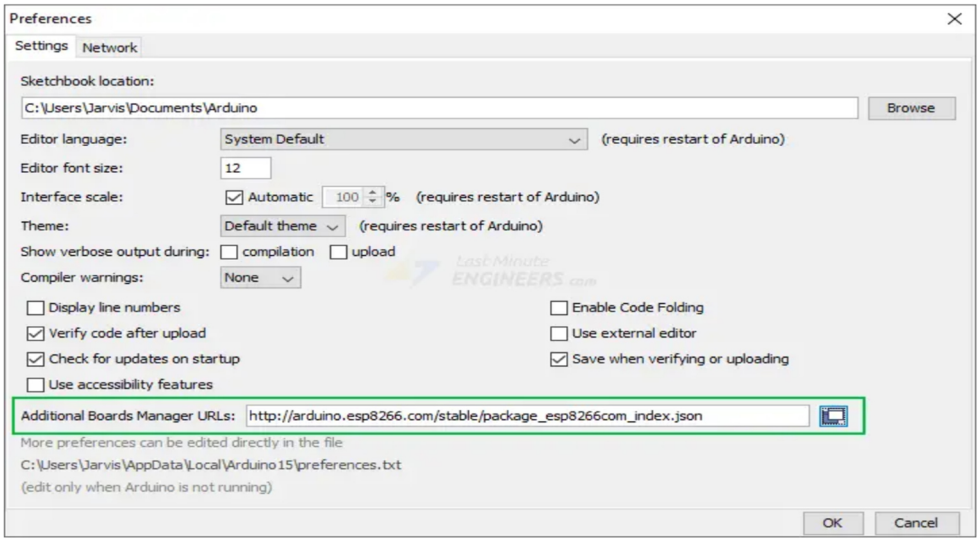
Premièrement, vous devez ajouter la série de cartes ESP8266 à l'IDE Arduino. Pour ce faire, rendez-vous dans le menu Fichier, puis sélectionnez Préférences :



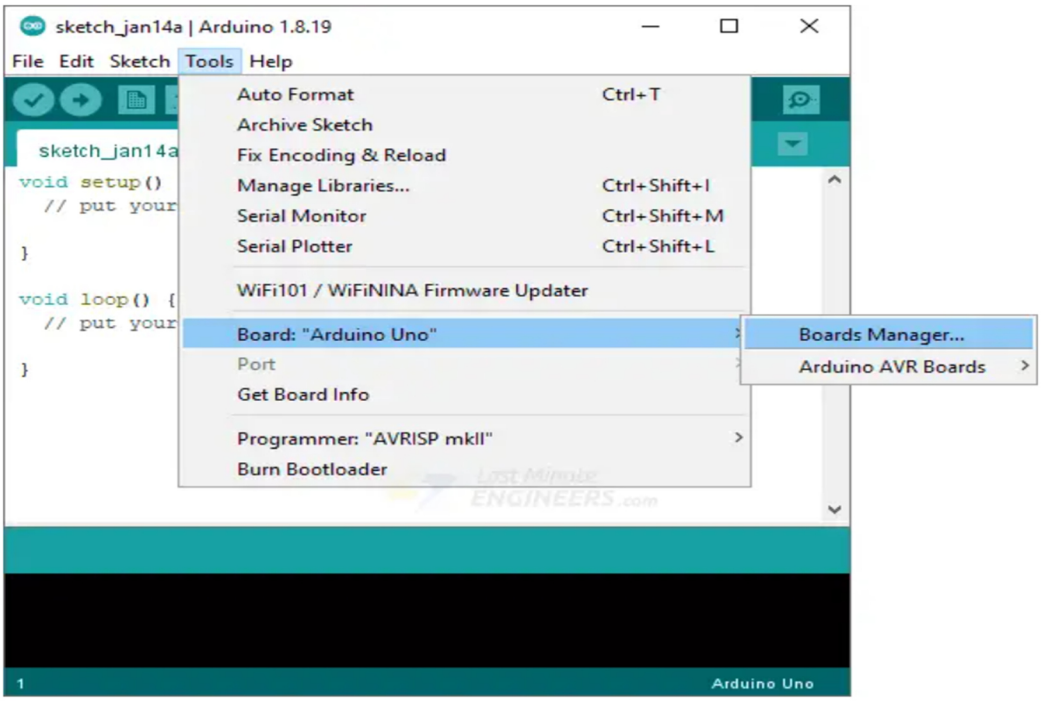
Dans la fenêtre des préférences, recherchez le champ "URL de gestionnaire de cartes supplémentaires" et collez le lien suivant :

http://arduino.esp8266.com/stable/package_esp8266com_index.json

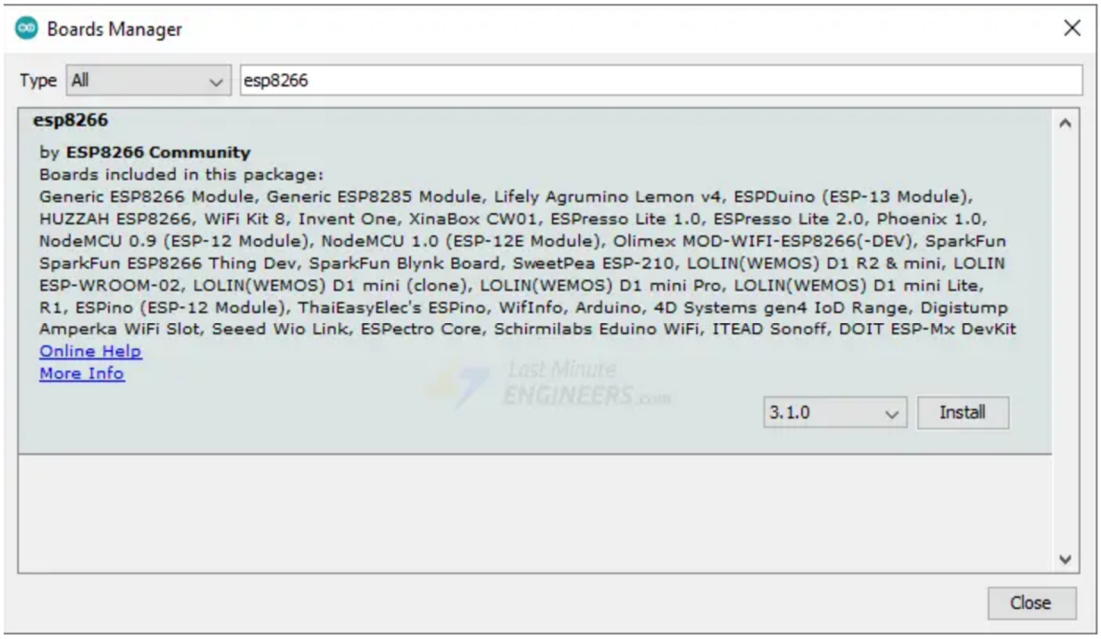
Ensuite, cliquea sur le bouton "OK"



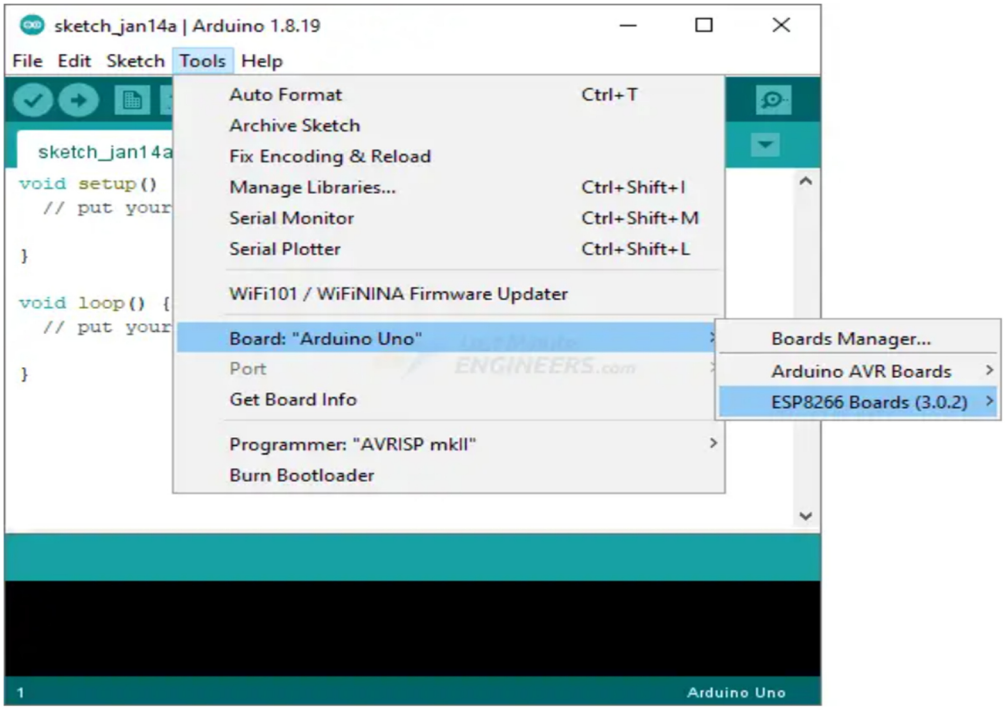
Maintenant, accédez à Outils > Carte > Gestionnaire de cartes...



Filtrez votre recherche en entrant "esp8266". Recherchez ESP8266 par ESP8266 Community. Cliquez sur cette entrée, puis choisissez Installer.



Après avoir installé le core ESP8266 Arduino, redémarrez votre Arduino IDE, puis accédez à Outils > Carte pour vous assurer que vous disposez des cartes ESP8266 disponibles.



Maintenant, sélectionnez votre carte dans le menu Outils > Carte (dans notre cas, c'est la NodeMCU 1.0 (Module ESP-12E)). Si vous n'êtes pas sûr de la carte que vous avez, sélectionnez Generic ESP8266 Module.