République **A**lgérienne **D**émocratique et **P**opulaire
Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

UNIVERSITE **M**OULOUD **M**AMMERI DE **T**IZI-**O**UZOU

FACULTE DU **G**ENIE **E**LECTRIQUE ET D' **I**NFORMATIQUE
DEPARTEMENT D'AUTOMATIQUE

Mémoire de Fin d'Etudes de MASTER ACADEMIQUE

Domaine : **Sciences et Technologies**
Filière : **Automatique**
Spécialité : **Automatique et Informatique Industrielle**

Présenté par
MOULLA Mohamed Cherif
NAIT LAZIZ Rezki

Thème

Implémentation d'une commande floue sur une carte Arduino pour la commande d'un moteur à courant continu

Mémoire soutenu publiquement le 25/06/2024 devant le jury composé de :

M KASRI Ahmed
MAA, UMMTO, **Président**

M KHATI Hocine
MAB, UMMTO, **Encadrant**

M HAMMOUCHE Kamel
Professeur, UMMTO, **Examinateur**

Mme ARAR Nacera
MAA, UMMTO, **Examinatrice**

Remerciements

On tient à exprimer notre gratitude à Dieu Tout-Puissant pour nous avoir donné la force et la sagesse nécessaires pour réaliser ce mémoire.

Nos sincères remerciements vont à notre promoteur, Dr. KHATI Hocine, de l'Université Mouloud Mammeri de Tizi-Ouzou, Département d'Automatique, pour ses conseils précieux et son soutien tout au long de ce travail. Son expertise, sa disponibilité et son encouragement constant ont été d'une grande aide.

Nos remerciements s'adressent également aux membres du jury pour l'honneur d'avoir assisté à notre soutenance et jugé ce travail.

On remercie également nos familles et nos amis pour leur soutien constant et leur compréhension pendant nos études.

On exprime notre reconnaissance envers tous les enseignants du Département d'Automatique de l'Université Mouloud Mammeri de Tizi-Ouzou pour leurs enseignements et leur disponibilité. Leur engagement et leur passion pour l'automatique ont enrichi notre parcours académique et nous ont inspiré à donner le meilleur de nous-mêmes.

Dédicaces :

Je dédie ce travail :

À mes chers parents, mon frère et ma sœur qui m'ont toujours soutenu et encouragé avec amour et bienveillance,

À la mémoire de mon grand-père paternel, ainsi qu'à ma chère grand-mère paternel, que Dieu lui accorde une longue vie.

À La mémoire de mes grands-parents maternels, je garde précieusement leurs souvenirs et l'amour qui continuent de résonner en moi, illuminant chaque étape de ma vie.

À mes oncles, tantes, cousins et cousines, pour leur soutien et leur affection constants,

À toute la famille MOULLA et IDRES, pour leur soutien,

À mes précieux amis et camarades,

À tous ceux qui m'ont aidé, encouragé et accompagné tout au long de mes études,

À mon binôme Rezki, avec qui j'ai partagé ce travail.

MOULLA Mohamed Cherif

Dédicaces :

Je dédie ce travail avec gratitude et respect :

À mes parents, ma sœur et tout les membres de ma famille qui m'ont soutenu et accompagné tout au long de mes études,

À tout mes amis, compagnons de cette aventure académique,

À mon binôme Mohamed Cherif, avec qui j'ai partagé ce travail.

À nos enseignants qui nous ont guidés et inspirés,

À tout les chercheurs et inventeurs, dont le travail a permis au notre d'exister,

À toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire.

NAIT LAZIZ Rezki

Notations

T-S	Takagi-Sugeno
PID	Proportionnel, Intégral, Dérivé
T	Température
t	Temps
IDE	Integrated Development Environment
$IDII$	Institut du Design d'Interaction d'Ivrea
E/S	Entrée / Sortie
PWM	Pulse Width Modulation
CAN	Conversion Analogique Numerique
USB	Universal Serial Bus
$ICSP$	In-Circuit Serial Programming
$JTAG$	Joint Test Action Group
CPU	Central Processing Unit
LCD	Liquid Crystal Display
MCC	Moteur à courant continu
DIY	Do it yourself

Table des matières

Table des figures

Liste des tableaux

Introduction générale

La régulation est un procédé complexe à mettre en place, nécessitant de nombreux calculs, de temps de conception et une représentation mathématique précise du système sur lequel elle est appliquée. Les systèmes automatiques, notamment les systèmes robotiques sont généralement des systèmes non linéaires complexes, avec une dynamique parfois inconnue, ce qui complique l'élaboration de systèmes pour les commander. De plus, la majorité des correcteurs classiques nécessitent la connaissance du modèle exact du système commandé pour délivrer de bonnes performances, or, nous ne disposons pas toujours de suffisamment de données sur le système en question.

Les correcteurs intelligents basés sur les méthodes d'intelligences artificielles ne nécessitent pas une connaissance parfaite du modèle précis du système. Cette intelligence est basée sur la logique floue qui permet une représentation mathématique du raisonnement humain, la rendant utilisable dans des processus tel que la régulation. Cependant, l'implémentation en pratique de ce type de régulateur nécessite des cartes de commande puissantes en raison de la complexité des calculs.

Le moteur à courant continu est l'élément le plus crucial dans les systèmes robotiques et il est utilisé pour diverses tâches, d'où l'importance d'avoir des processus de commande fiables.

Dans ce projet, nous allons commander un moteur à courant continu de $12V$ en vitesse par le biais d'un régulateur flou implémenté dans une carte Arduino Due.

Pour ce faire, nous allons commencer par citer et expliquer les principes de base de la logique floue, logique sur laquelle est basé le raisonnement de notre régulateur. Nous parlerons aussi de la commande floue et des types de régulateurs flous les plus connus, à savoir Mamdani et Takagi-Sugeno. Nous accorderons une attention particulière au régulateur Takagi-Sugeno qui fera l'objet de notre mémoire.

Dans le second chapitre, nous nous concentrerons sur la partie Hardware de notre projet, principalement la carte Arduino, nous comparerons les caractéristiques des cartes les plus répandues pour pouvoir choisir la mieux adaptée à notre projet, à noter que la régulation nécessite un microcontrôleur performant étant donné la charge de calcul et l'importance d'utiliser une fréquence élevée. Aussi, nous expliquerons les deux méthodes de programmation envisageables et celle pour laquelle nous opterons.

Dans le troisième chapitre, nous citerons les étapes suivies lors de l'identification approximative du moteur, de la conception, la simulation et l'implémentation du régulateur Takagi-Sugeno. Par la suite, nous suivrons la même démarche pour le régulateur PID qui servira de point de repère pour évaluer la qualité des résultats obtenus par l'utilisation du régulateur flou. Nous réaliserons des tests pour vérifier le bon fonctionnement du système.

Dans le dernier chapitre, nous présenterons les résultats obtenus des deux méthodes de régulation en simulation, mode externe et implémentation. Nous tracerons les courbes suivantes : la vitesse, l'erreur de vitesse et la tension de commande. Ensuite, nous analyserons les résultats obtenus en ayant comme critère la rapidité, la stabilité et l'efficacité.

Nous terminerons avec une conclusion globale où nous reverrons les points clés du projet.

Chapitre 1

Généralités sur la logique floue

1.1 Introduction

La logique floue est une partie importante de notre projet étant donné que l'élément essentiel (à savoir le régulateur flou) est basé sur cette dernière. Nous allons donc étudier les caractéristiques et éléments de cette logique qui nous permettront de comprendre le fonctionnement des régulateurs flous et de pouvoir les configurer pour obtenir le fonctionnement désiré.

Nous allons commencer par l'explication de la notion de logique floue et des éléments qui la composent, nous suivrons ensuite par les opérations possibles sur les ensembles flous. Puis, nous allons nous intéresser à la commande floue, son fonctionnement, l'explication de chaque étape de raisonnement et des types de régulateurs communément utilisés.

Enfin nous allons étudier plus en détail le régulateur Takagi-Sugeno qui sera par la suite implémenté dans la commande de notre système.

1.2 Logique floue

1.2.1 Définition

Le concept de logique floue est le processus d'affectation de valeurs de vérités variables réelles comprises entre 0 et 1 au lieu d'être vraies ou fausses (**degré de verité**).

Dans notre quotidien, nous utilisons de nombreuses expressions vagues et imprécises. Sur la base de ces expressions, il est difficile de faire une déduction ou une représentation du fait caractérisé.

À titre d'exemple, les expressions « une température est élevée » ou encore « un objet est volumineux » peuvent être interprétées différemment suivant la personne ou l'environnement dans lesquelles elles sont utilisées, il s'agit là de logique floue. Ces propositions ne sont pas toujours clairement définies comme vraies ou fausses car il ne s'agit pas de données précises.

Pour pouvoir exploiter ce type de données, il est nécessaire de les représenter sous une forme mathématique. Il s'agit d'une étape nécessaire dans la manipulation des déclarations floues.

Dans l'exemple cité plus tôt « une température est élevée », nous pouvons déterminer le degré de vérité de la proposition en nous basant sur la valeur précise de cette température. Le degré de vérité obtenu dépend de notre compréhension du concept de "température élevée",ce dernier ne peut pas être défini par un sous-ensemble ordinaire (0 ou 1). C'est ainsi que Lotfi Zadeh, fondateur de la logique floue a été dirigé vers celle-ci [12].

Degré de vérité :

En logique binaire, une affirmation est vraie ou fausse, son degré de vérité vaut 1 ou 0.

Valeur	Signification
1	Absolument vrai
0	Absolument faux

TABLE 1.1 – Degré de vérité en logique binaire

En logique floue, une affirmation est plus ou moins vraie (donc, plus ou moins fausse) ; son degré de vérité varie entre 0 et 1.

Degré de verité	Signification
0.0	Absolument faux
0.2	Plutôt faux
0.4	Quelque peu faux
0.6	Quelque peu vrai
0.8	Plutôt vrai
1.0	Absolument vrai

TABLE 1.2 – Degré de vérité en logique floue

Exemple :

Température T	Degré de vérité pour CHAUD
$< -10C$	0.0
$-5C$	0.1
$0C$	0.2
$5C$	0.3
$10C$	0.4
$15C$	0.5
$20C$	0.6
$25C$	0.7
$30C$	0.8
$35C$	0.9
$> 40C$	1.0

TABLE 1.3 – Exemple de degré de vérité pour dire qu'un objet est chaud.

1.2.2 Historique

Développement de la logique floue [16] :

1961 : Lotfi Zadeh affirme dans son article un nouveau type de mathématiques «floues».

1973 : Il détaille davantage la logique floue et introduit les fonctions d'appartenance.

1974 : E.Mamdani présente un régulateur flou pour contrôler une machine à vapeur.

1983 : La première application industrielle dans un four à ciment au Danemark.

1987 : Yamakawa présente le premier contrôleur analogique flou.

1988 : Togai implémente les premiers processeurs numériques flous.

1989 : LIFE : le laboratoire de recherche internationale sur le flou en ingénierie commence au Japon.

1992 : La première conférence internationale de l'IEEE sur le système flou.

1.2.3 Sous-ensemble flou

Un sous-ensemble classique A d'un ensemble U est défini par sa fonction d'appartenance $\mu_A(x)$ qui caractérise chaque élément x appartenant à U [12].

$$\mu_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} \tag{1.1}$$

Cette fonction d'appartenance ne peut prendre que deux valeurs possibles :

La valeur 1 si x appartient à A

ou

La valeur 0 si x n'appartient pas à A.

Un sous-ensemble flou A d'un ensemble U défini par une fonction d'appartenance $\mu_A(x)$ peut prendre plusieurs valeurs comprises entre 0 et 1. Le degré d'appartenance de l'élément x au sous-ensemble flou A appartient à l'intervalle $[0, 1]$ [7] .

$$\mu_A(x) \in [0, 1] \tag{1.2}$$

Ce sous-ensemble contient les éléments nécessaires à l'application de la logique booléenne pour une proposition donnée, mais aussi les valeurs intermédiaires. On en déduit que les sous-ensembles classiques sont un cas particulier de sous-ensembles flou [12].

1.2.4 Variable linguistique

Une variable linguistique décrit une situation imprécise par des mots ou expression tels que : [7]

- Peu, beaucoup, énormément.
- Rarement, fréquemment, souvent.
- Froid, tiède, chaud.
- Petit, moyen, grand.

Ces variables ne représentent pas des objets mathématiques, elles ne peuvent donc pas être traitées en tant que tel.

Exemple :

La variable linguistique « **température** » peut appartenir aux ensembles flous « **froid** », « **tiède** » ou « **chaud** ».

1.2.5 Fonction d'appartenance

Une fonction d'appartenance floue est notée $\mu_A(x)$ définie sur $[0, 1]$. Elle permet d'établir une relation entre le degré de vérité de la variable floue et la grandeur d'entrée correspondante, les sous-ensembles flous sont ceux caractérisés par une fonction d'appartenance floue.

Pour une même proposition, plusieurs fonctions différentes peuvent être considérées. Le choix de cette fonction est subjectif et dépend de la perspective ainsi que des paramètres pris en compte lors de la conception. Dans certains cas, le concepteur s'inspire du raisonnement humain pour tracer ces fonctions [5].

Il existe plusieurs types de fonctions d'appartenance, certaines sont conçues de façon à faciliter les calculs, parmi elles on trouve des formes paramétriques déterminées par le choix d'un nombre fini de paramètres.

Voici les fonctions d'appartenance les plus couramment utilisées :

a.Fonction Triangulaire :

$$u_A(x) = \begin{cases} 0 & x < a \\ \frac{x-a}{b-a} & a \leq x \prec b \\ \frac{c-x}{c-b} & b \leq x \prec c \\ 0 & x > c \end{cases} \tag{1.3}$$

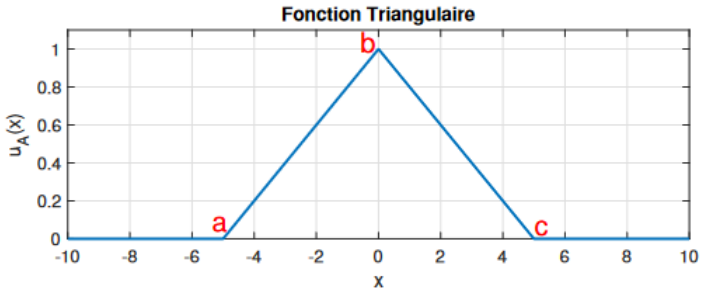


FIGURE 1.1 – Fonction Triangulaire.

b.Fonction Trapézoïdale :

$$u_A(x) = \begin{cases} 0 & x < a \\ \frac{x-a}{b-a} & a \leq x \prec b \\ 1 & b \leq x \prec c \\ \frac{d-x}{d-c} & c \leq x \prec d \\ 0 & x > d \end{cases} \tag{1.4}$$

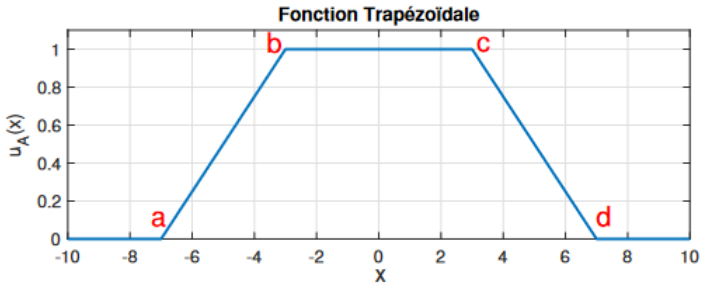


FIGURE 1.2 – Fonction Trapézoïdale.

c.Fonction Gaussienne :

$$u_A(x) = \exp(\frac{1}{2}(\frac{x-m}{\sigma})^2) \tag{1.5}$$

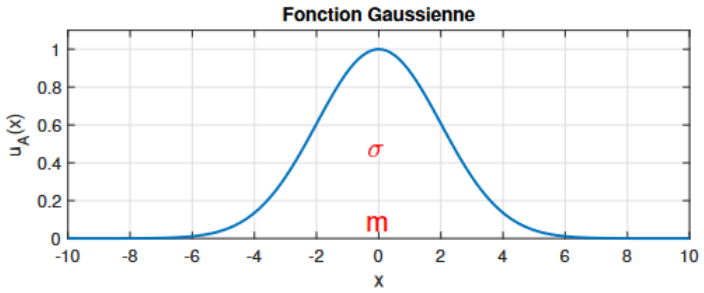


FIGURE 1.3 – Fonction Gaussienne.

d.Fonction Singleton :

$$u_A(x) = \begin{cases} 1 & x = a \\ 0 & x \neq a \end{cases} \tag{1.6}$$

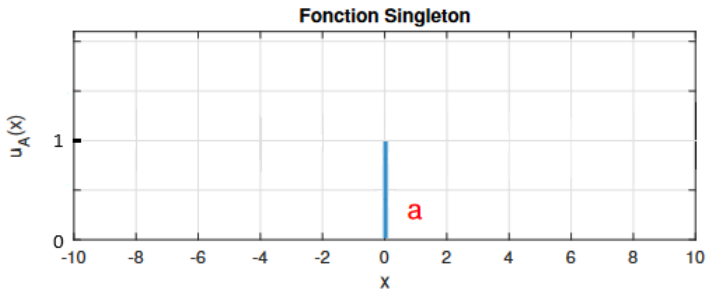


FIGURE 1.4 – Fonction Singleton.

Exemple :

Lors de l'automatisation d'un système de ventilation, le concepteur demande à un opérateur de faire varier manuellement la vitesse du ventilateur en fonction de la température et s'en inspire ensuite dans le choix des fonctions d'appartenance.

Lors de la sélection des fonctions d'appartenance dans un système, il est préférable d'avoir un chevauchement entre les fonctions voisines pour assurer un fonctionnement lisse et éviter des variations brusques de la sortie [5].

Voici un exemple de fonction d'appartenance à la proposition de "température élevée" :

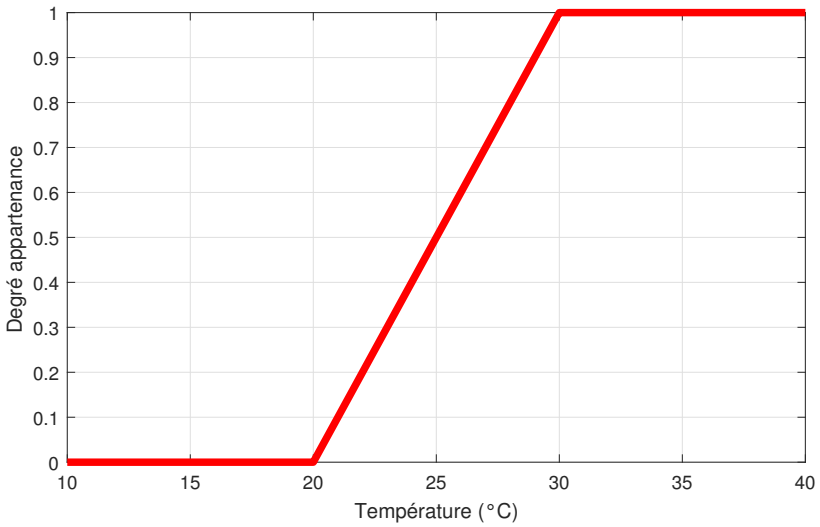


FIGURE 1.5 – Fonction d'appartenance relative à la proposition de "température élevée".

1.2.6 Univers de discours

Il s'agit du domaine de variation de la variable linguistique. En pratique, il représente le domaine de fonctionnement du système à commander, il faut donc le prendre en considération lors du réglage du régulateur [7].

L'univers de discours est important car il détermine les intervalles requis pour des opérations correctes, lorsque on change l'actionneur commandé, une mise à niveau doit être faite dans l'univers de discours pour assurer un fonctionnement correct [5].

1.2.7 Règle d'inférence

Les règles d'inférence sont des règles basées sur un raisonnement approximatif permettant d'aboutir à de vagues conclusions à partir de vagues propositions. Il s'agit donc du processus consistant à obtenir un ensemble flou de sortie étant donné les entrées du système et une collection de règles [14]. Les règles d'inférence sont conçues de manière à envisager tout cas de figure où le régulateur doit réagir.

Les règles d'inférences peuvent être décrites de plusieurs façons :

a. Linguistiquement

Les règles peuvent être exprimées sous la forme générale :

Si $condition1$ Alors $action1$ ou

Si $condition2$ Alors $action2$ ou

Si .....

Si $conditionN$ Alors $actionN$.

Les conditions peuvent dépendre de plusieurs variables liées entre elles par des opérateurs OU ou ET.

Exemple :

Si température froide ET hygrométrie importante *Alors* ouvrir la vanne d'admission d'air chaud.

b. Symboliquement :

Il s'agit d'une description linguistique où l'on remplace la désignation des ensembles flous par des abréviations.

Exemple :

Si TF et HI , Alors OV

où :

TF : représente la température froide.
HI : représente une hygrométrie importante.
OV : représente l'ouverture de la vanne d'admission d'air chaud.

Dans cette représentation symbolique, les termes linguistiques ont été remplacés par des abréviations, ce qui permet une formalisation de la règle d'inférence. Cela facilite notamment son utilisation dans des systèmes informatiques pour l'automatisation de la prise de décision.

c. Par matrice d'inférence

Elle rassemble toutes les règles d'inférences sous forme de tableau pour simplifier leurs représentation [3]. Prenons un tableau à deux dimensions, les entrées du tableau représentent les ensembles flous des variables d'entrées (température et humidité). L'intersection d'une colonne et d'une ligne donne l'ensemble flou de la variable de sortie définie par la règle. Il y a autant de cases que de règles.

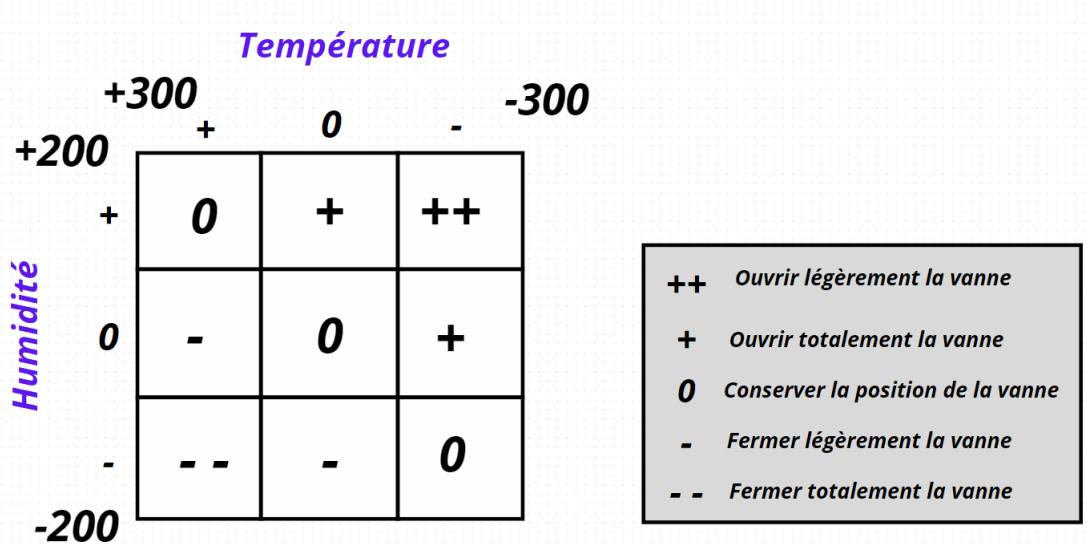


FIGURE 1.6 – Matrice d'inférence pour commande de vanne en fonction de la température et l'humidité.

1.2.8 Opérations sur les ensembles flous

Dans les ensembles classiques, les variables sont reliées à l'aide d'opérateurs logiques tels que : ET (Intersection), OU (Union) et NON (complément à 1). Dans la logique floue, ces opérateurs sont interprétés respectivement par : Minimum, Maximum et Complément à 1, et sont définis par leurs fonctions d'appartenance [11].

Affirmation	Degré de vérité (fonction d'appartenance)
X	$\mu(X)$
Y	$\mu(Y)$
X ET Y	$min(\mu(X), \mu(Y))$
X OU Y	$max(\mu(X), \mu(Y))$
NON (X)	$1 - \mu(X)$

TABLE 1.4 – Operations sur les ensembles flous

Exemples d'operations floues :

$X=$ « La température est élevée » $Y=$ « La couche de glace est fine »

		Logique binaire		Logique floue	
X	Y	X ET Y	X OU Y	X ET Y	X OU Y
1	1	1	1	1	1
1	0	0	1	0	1
0	1	0	1	0	1
0	0	0	0	0	0
0.7	1			0.7	1
0.7	0.8			0.7	0.8
0.7	0			0	0.7
0.5	0.5			0.5	0.5

TABLE 1.5 – Comparaison entre opérations floues et binaires.

1.3 Régulateur flou

Les techniques de contrôle classiques présentent beaucoup de limitations. Il est souvent difficile de modéliser par un ensemble d'équations mathématiques le comportement d'un système non linéaire, partiellement inconnu ou avec des incertitudes dynamiques imprévisibles. Nous faisons donc recours à la logique floue, où tous ces comportements imprévisibles peuvent être modélisés à l'aide de l'approche linguistique de Zadeh, selon un modèle proche du raisonnement humain [7].

L'utilisation de la logique floue dans le contrôle des systèmes permet d'obtenir une loi de commande considérablement efficace pour des systèmes complexes et fortement non linéaires sans avoir recours à un modèle mathématique bien défini, et cela par le biais d'inférences avec des règles floues basées sur des variables linguistiques [7].

Deux grands types de règles floues existent, à savoir les règles floues de Mamdani et les règles floues de Takagi-Sugeno [9].

Régulateur Mamdani : Le régulateur de Mamdani est le plus répandu, il a été proposé par Mamdani et Assilian dans le but de contrôler une combinaison machine à vapeur et chaudière en synthétisant un ensemble de connaissances linguistiques (règles de contrôle) obtenues auprès d'opérateurs humains expérimentés en s'inspirant de la théorie de Zadeh. La particularité de ce contrôleur est que les antécédents et conséquents sont tout deux des ensembles flous [10].

Régulateur Takagi-Sugeno : Le modèle de Takagi-Sugeno a été créé pour palier aux problèmes rencontrés lors de l'utilisation du modèle Mamdani sur certains systèmes [18], sur lesquels l'analyse des performances ainsi que l'étude de la stabilité du système pouvaient s'avérer difficiles. Il est représenté par un ensemble de sous-modèles linéaires pondérés par le biais de fonctions d'appartenance [2].

Dans le modèle T-S, l'antécédent de règle est toujours exprimé linguistiquement pendant que la conséquence est exprimée numériquement soit par un polynôme ou une fonction ou encore une équation différentielle.

1.4 Fonctionnement d'un régulateur flou

Les systèmes basés sur des modèles flous sont généralement composés de 5 blocs :

1. **Interface de fuzzification :** Transforme les entrées nettes en degrés d'accomplissements.
2. **Mécanisme d'inférence :** Effectue les opérations d'inférence contenues dans les règles.
3. **Base de règles :** Contient les règles « Si ... Alors... ».
4. **Interface de défuzzification :** Transforme les résultats flous en une sortie nette [9].
5. **Le système :** Processus à commander.

La majorité des régulateurs flous reçoivent des données de capteurs (variables d'entrée) et s'en servent pour commander le fonctionnement d'un actionneur à travers les variables de sortie.

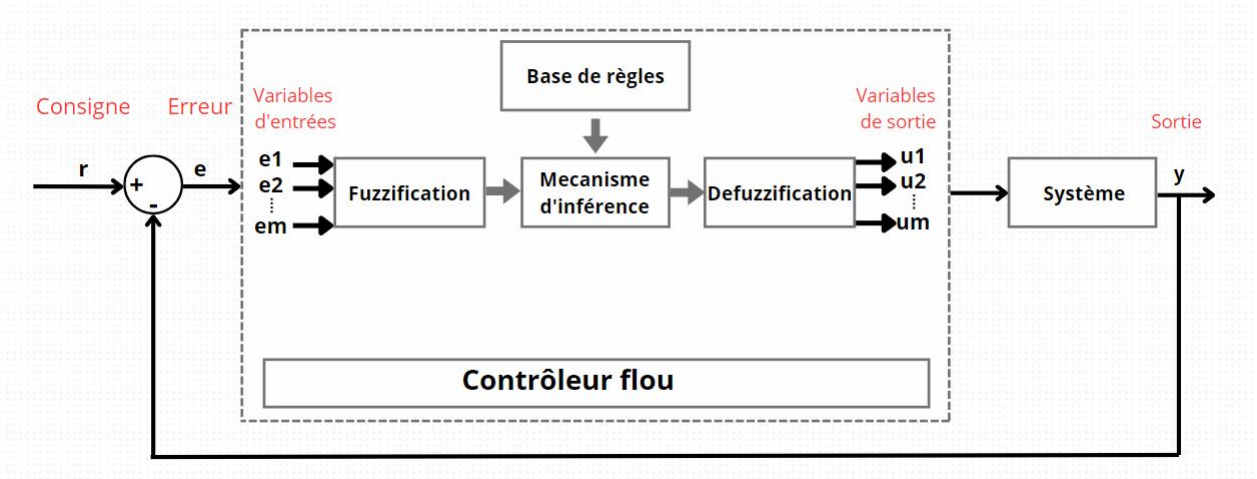


FIGURE 1.7 – Schéma de fonctionnement d'un modèle flou.

1.4.1 Fuzzification

C'est le mécanisme réalisant la conversion ''Numérique - linguistique". Les variables d'entrée et de sortie choisies pour modéliser ou commander un système sont des grandeurs numériques. L'étape de fuzzification consiste à transformer ces grandeurs réelles en variables linguistiques en vue d'un traitement d'inférence. Ainsi, à chaque variable d'entrée et de sortie sont associés des ensembles caractérisant les termes linguistiques pris par ces variables. Ces termes seront utilisés pour l'écriture des règles d'inférence [19].

Le bloc de fuzzification effectue les fonctions suivantes :

1. Définition des fonctions d'appartenance de toutes les variables d'entrées.
2. Transformation des grandeurs physiques (réelles ou numériques) en des grandeurs linguistiques ou floues.
3. Représentation d'échelle transférant la plage des variables d'entrées aux univers de discours correspondants.

Exemple :

Considérons une entrée température T définie par l'ensemble des variables linguistiques : FROID, FRAIS, MOYEN, TIÈDE, CHAUD.

L'univers de discours associé est : $[0, 32]$

Les fonctions d'appartenance de l'entrée sont :

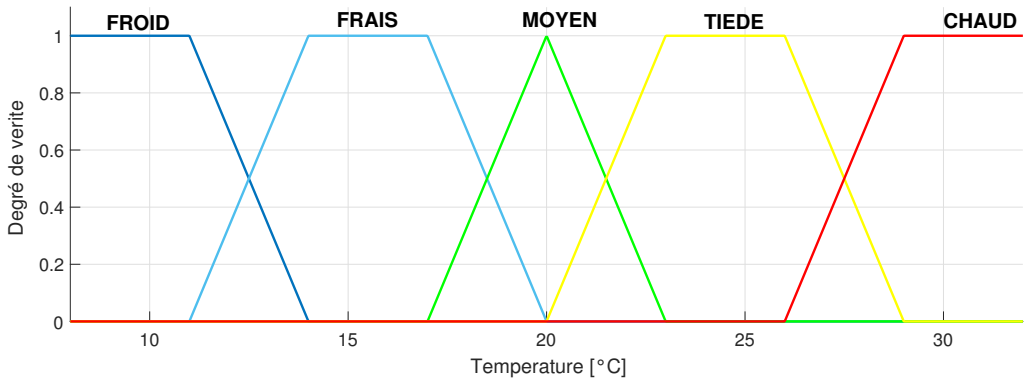


FIGURE 1.8 – Représentation de fonctions d'appartenance.

1.4.2 Base de règles

Une base de règles est constituée de toutes les règles d'inférence "Si...Alors" utilisées dans le processus de régulation. Ces règles sont définies par le concepteur.

Le régulateur utilise la base de règles pour déterminer la sortie, ce comportement est décrit par les règles floues sous la forme suivante [17] :

$$Si \text{ condition } 1 ET/OU \text{ condition } 2 \ ET/OU \ldots Alors \text{ action sur les sorties}$$

$$Si \text{ condition } 3 \ ET/OU \text{ condition } 4 \ ET/OU \ldots Alors \text{ action sur les sorties}$$

$$\ldots$$

$$Si \text{ condition } n \ ET/OU \text{ condition } n+1 \ ET/OU \ldots Alors \text{ action sur les sorties.}$$

Les entrées peuvent être une erreur, une variation de l'erreur ou une somme d'erreurs, et la sortie peut être l'action de contrôle ou l'action de variation de contrôle.

1.4.3 Mécanisme d'inférence

L'inférence floue ou la logique de prise de décision est le cœur du contrôleur flou qui possède la capacité de simuler les décisions humaines et de déduire (inférer) les actions de commande floue à l'aide de l'implication floue et des règles d'inférence de la logique floue [5].

La connexion entre une cause (antécédent) et un effet (conséquence) est faite par ce raisonnement.

Nous utilisons généralement des mécanismes d'inférence sous forme :

Si Cause1 est A ET Cause2 est B Alors Effet est C

Où A et B sont des variables linguistiques. Cela nous permet d'obtenir des résultats mathématiques basés sur une variable linguistique, ce qui rend le système intelligent dans un sens.

Il existe plusieurs méthodes d'inférence, parmi elles :

1. Méthode de Mamdani
2. Méthode de Larsen
3. Méthode de Takagi-Sugeno

Exemple :

Si "La température est Moyenne" *Alors* "mettre la vitesse du ventilateur sur Rapide"

Moyenne : est une fonction qui définit la température.
Rapide : est une fonction qui définit la vitesse du ventilateur.

1.4.4 Défuzzification

Après l'application du raisonnement flou, les résultats obtenus sont un ensemble de données sous forme linguistique (Régulateur Mamdani) ou numériques (Régulateur Takagi-Sugeno) et doivent donc être convertis en une seule sortie numérique pour pouvoir ensuite être exploités. Il existe différentes méthodes pour cela, certaines donnent une sortie globale calculée sur la base de tout les éléments résultants et de leurs degrés de vérité respectifs. D'autres sélectionnent uniquement l'élément relié au degré de vérité le plus élevé.

Il existe différentes méthodes de défuzzification, en voici trois :

a.Centre de gravité [5]

$$y = \frac{\sum_{i=1}^{N} u_{ci}(y)\omega_i}{\sum_{i=1}^{N} u_{ci}(y)} \tag{1.7}$$

où :

y : la valeur numérique de sortie.
N : le nombre de valeurs linguistiques de conséquence.
ω_i : hauteur des ensembles flous de conséquence.
u_{ci} : le degré d'appartenance.

b.Moyenne pondérée

$$y = \frac{\sum_{i=1}^{N} u_i \mu_i}{\sum_{i=1}^{N} \mu_i} \tag{1.8}$$

où :

y : la valeur numérique de sortie.

N : le nombre de règles.

μ_i : le degré de verité de la i-ème règle.

u_i : le conséquent de la i-ème règle.

c.Somme pondérée

$$y = \sum_{i=1}^{N} u_i \mu_i \tag{1.9}$$

où :

y : la valeur numérique de sortie.

N : le nombre de règles.

μ_i : le degré de verité de la i-ème règle.

u_i : le conséquent de la i-ème règle.

Exemple :

Pour illustrer le processus, considérons les degrés d'appartenance suivants pour le cas de la commande d'un moteur à courant continu :

$$\mu 1 = 0 \text{ pour } GN$$

$$\mu 2 = 0 \text{ pour } N$$

$$\mu 3 = 0 \text{ pour } Z$$

$$\mu 4 = 0 \text{ pour } N$$

$$\mu 5 = 0.6 \text{ pour } Z$$

$$\mu 6 = 0.2 \text{ pour } P$$

$$\mu 7 = 0 \text{ pour } Z$$

$$\mu 8 = 0.4 \text{ pour } P$$

$$\mu 9 = 0.2 \text{ pour } GP$$

Avec :

GN(Grand Négatif)=1 N(Négatif)=0.5 Z(Zero)=0 P(Positif)=0.5 GP(Grand Positif)=1

En appliquant la méthode de la moyenne pondérée, la valeur défuzzifiée u_{def} est calculée comme suit :

$$u_{def}=\frac{(0\times-1)+(0\times-0.5)+(0\times0)+(0\times-0.5)+(0.6\times0)+(0.2\times0.5)+(0\times0)+(0.4\times0.5)+(0.2\times1)}{(0.6+0.2+0.4+0.2)}$$

$$u_{def}=\frac{0.5}{1.4}$$

$$u_{def}=0.357$$

Alors, la sortie u à ajouter à la commande serait de 0.357, indiquant une légère augmentation de la puissance fournie à la machine.

1.4.5 Modèle de Takagi-Sugeno

Le modèle flou de Takagi-Sugeno est une représentation mathématique des systèmes non linéaires, appartenant à la famille des systèmes quasi-LPV (Linear-Parameter varying control) [18].

Ce modèle est représenté par un ensemble de sous-modèles linéaires pondérés par le biais de fonctions non linéaires appelées «Fonctions d'appartenance» [2]. L'activation de chaque sous-modèle est régie par les règles «Si ... Alors» .

Le modèle T-S (Takagi-Sugeno) est particulièrement adapté aux systèmes dont l'analyse des performances ainsi que l'étude de la stabilité peuvent s'avérer difficiles.

La particularité du modèle est que l'antécédent de règle est toujours exprimé linguistiquement pendant que la conséquence est exprimée numériquement soit par une constante, un polynôme, une fonction ou encore une équation différentielle.

Ces règles prennent la forme suivante :

$$R_i : \text{Si } x \text{ est } A_i \text{ Alors } y_i = f_i(x), i = 1, ..., z$$

où :

R_i est la i-ème règle du modèle
A_i : la fonction d'appartenance
z : le nombre de règles
x : la variable d'entrée
y : la variable de sortie

L'antécédent peut être exprimé comme une combinaison logique de propositions simples avec des sous-ensembles flous unidimensionnels :

$$R_i : \text{Si } x_1 \text{ est } A_{i1} \text{ et } x_2 \text{ est } A_{i2} \text{Alors } y_i = f_i(x), i = 1, ..., z$$

Les fonctions f_i sont choisies comme des fonctions paramétrées avec la même structure pour chaque règle où seul les paramètres varient. L'une des formes les plus utilisées est la forme affine Takagi-Sugeno suivante :

$$y_i = a_i^T x + d_i \tag{1.10}$$

où :

a_i : vecteur de paramètres $a_i \in R^p$
d_i : scalaire

Les conclusions des règles sont donc des sous-espaces linéaires p-dimensionnels dans R^{P+1}

Sortie : avant de sélectionner la sortie adéquate ; il faut d'abord calculer le degré d'accomplissement $\beta_i(x)$ de la condition.

Le degré d'accomplissement $\beta_i(x)$ est égal au degré d'appartenance de l'entrée multi-dimensionnelle x, c'est-a-dire :

$$\beta_i = \mu_{Ai}(x) \text{ avec } \mu_{Ai}(x) : R^P \to [0, 1]$$

Quand on a des connectives logiques, le degré d'accomplissement de l'antécédent est calculé comme une combinaison des degrés d'appartenance des propositions individuelles en utilisant les opérateurs de logique floue.

Dans la modélisation T-S, l'obtention de la sortie est réalisée à partir de la combinaison des opérations d'inférence et de défuzzification. La sortie finale est généralement

calculée comme la moyenne des sorties correspondant à R_i pondérées par le degré d'accomplissement respectif (Moyenne pondérée) selon l'expression [14] :

$$y = \frac{\sum \beta_i(x).y_i}{\sum \beta_i(x)} \tag{1.11}$$

1.4.5.1 Construction du modèle de Takagi-Sugeno

Il existe trois méthodes de réalisation de modèle sous forme T-S :

1. La première est basée sur des méthodes d'identification a partir de données d'entrée/sortie (modélisation floue classique) [18], cette méthode ne nécessite pas de modèle de connaissance ; elle convient aux installations qui ne peuvent pas ou sont trop difficiles à représenter par des modèles analytiques et/ou physiques.
2. La deuxième consiste à linéariser le système autour d'un nombre donné de points de fonctionnement soigneusement choisis dans le but de décrire la dynamique du système par un ensemble fini de modèles linéaires, ces derniers sont ensuite interconnectées par des fonctions non linéaires pour obtenir le modèle T-S.
3. La dernière approche est appelée « approche par secteur non linéaire », elle consiste a effectuer une transformation polytopique convexe des termes non linéaires du modèle [2].

Dans notre cas, on s'intéresse à la première méthode car elle correspond à notre cas de figure, il nous est nécessaire d'identifier approximativement le moteur sans obligatoirement connaître ses caractéristiques internes précises.

La procédure consiste principalement à identifier la structure et les paramètres.

1.5 Conclusion

Les systèmes basés sur la logique floue s'avèrent très utiles dans le développement de solutions de régulation efficaces .Ce succès est dû à l'utilisation d'expressions linguistiques à la place de complexes équations mathématiques, les expressions linguistiques permettent au concepteur d'avancer plus vite dans la réalisation en accordant davantage d'attention aux résultats obtenus plutôt qu'aux calculs mathématiques tout en préservant la qualité des résultats obtenus. Aussi, le régulateur Takagi-Sugeno permet une régulation sans modèle précis, ce qui correspond à notre cas. Ce processus doit être effectué par un microcontrôleur adapté à la charge de calculs, c'est ce que nous allons voir dans le chapitre suivant.

Chapitre 2

Généralités sur les cartes Arduino

2.1 Introduction

Dans ce chapitre, nous plongerons dans l'univers passionnant d'Arduino, qui est un élément essentiel de notre projet, la carte Arduino représente l'unité de calculs et de traitement de données qui assurera le fonctionnement de notre régulateur.

Nous explorerons les cartes Arduino les plus couramment utilisées, en mettant en lumière les caractéristiques distinctives de chacune et nous les comparerons.

Ensuite, nous examinerons de plus près les options de logiciel de développement, en mettant en avant l'IDE Arduino et MATLAB-Simulink.

Enfin, nous expliquerons le choix de la carte ainsi que l'environnement de conception utilisés dans notre projet qui ont permis d'assurer un fonctionnement correct, une optimisation des ressources et du temps.

2.2 Arduino

2.2.1 Description

La carte Arduino est un projet inventé en 2005 par une équipe italienne de l'Institut du Design d'Interaction d'Ivrea (IDII) désirant permettre à des étudiants en art et en design de créer tout type de projets.

Arduino est une plateforme électronique open-source (libre de droits) dont les principes de sa conception sont la facilité d'utilisation logicielle et matérielle. Les cartes Arduino sont capables de lire des entrées numériques ou analogiques, et produire une sortie suivant les instructions données dans le programme exécuté. Il existe différents modèles de cartes adaptées aux différentes utilisations comme l'électrotechnique industrielle, les systèmes embarqués (le modélisme et la domotique), l'art contemporain, le pilotage d'un robot, la commande des moteurs, l'automatique etc... [13], nécessitant des performances basiques ou élevées ainsi qu'un nombre d'entrées/sorties faible ou élevé.

Cependant, toutes les cartes Arduino fonctionnent en faible tension et courant (Max 5V & 900mA) ,on ne peut donc pas s'en servir comme alimentation électrique pour les composants commandés sauf dans le cas de composants à faible consommation d'énergie comme les capteurs.

Il est aussi utile de mentionner sa réutilisabilité, sur la plus part des modèles, les branchements ne nécessitent pas de soudure. Plus encore, il existe des supports (Shields) permettant d'enlever la carte tout en gardant le branchement individuel des pins. Elle devient alors facilement utilisable sur plusieurs projets simultanément.

2.2.2 Modèles de cartes

Depuis son invention, la carte Arduino a vu un grand nombre de versions. Actuellement, il existe plus de 20 versions cartes Arduino pour englober plusieurs utilisations ou mieux convenir à un type de projet relativement précis.

Ci-dessous, nous donnerons une brève définition ainsi que les caractéristiques de base de certaines versions les plus largement utilisées.

2.2.2.1 Arduino Nano

a.Définition

L'Arduino Nano est un condensé d'Arduino qui ne mesure que 1,9 cm sur 4,3 cm. Ces dimensions sont parfaites pour réduire les dimensions des projets. La Nano a toute la puissance de l'Arduino Uno, puisqu'elle utilise le même microcontrôleur ATmega328, mais ne fait qu'une fraction de sa taille. Elle tient à merveille sur une platine d'essai, ce qui la rend idéal pour le prototypage [13].

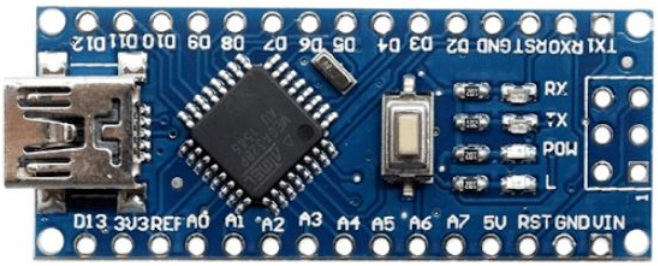


FIGURE 2.1 – Arduino Nano.

b.Caracteristiques techniques

Les principales caractéristiques de la carte Arduino Nano sont présentées dans le tableau suivant :

Catégorie	*Valeur*
Microcontrôleur	Atmega 168
Fréquence d'horloge	$16MHz$
Tension de service	$5V$
Tension d'entrée (recommandée)	$7-12V$
Tension d'entrée (limites)	$6-20V$
Ports numériques	14 E/S (Entrées/Sorties)
Ports-analogiques	8 entrées analogiques
Courant maxi. par broche d'E/S	$40mA$
Mémoire	$16Ko$ Flash
Interface	USB
Dimensions	$45 \times 18mm$

TABLE 2.1 – Caractéristiques techniques de la carte Arduino Nano.

c.Avantages et inconvénients

Avantages

1. Compacte : La Nano est extrêmement petite, ce qui la rend idéale pour les projets nécessitant un encombrement réduit.
2. Polyvalence : Elle offre une grande variété de fonctionnalités dans un format compact.
3. Faible consommation d'énergie : Elle consomme peu d'énergie, ce qui la rend adaptée aux projets alimentés par batterie.
4. Prix abordable : Elle est généralement moins chère que d'autres cartes Arduino.

Inconvénients

1. Moins de broches d'E/S : Elle dispose de moins de broches d'entrée/sortie que d'autres cartes Arduino, ce qui peut limiter les options de connectivité.
2. Puissance de calcul limitée : Elle a moins de puissance de traitement et de mémoire que certaines autres cartes Arduino [1].

2.2.2.2 Arduino Uno

a.Définition

L'Arduino Uno est l'une des cartes Arduino les plus populaires. C'est simplement de l'évolution du premier Arduino.

La carte Uno est très simple, elle est la plus utilisée et la plus documentée. La carte Uno est parfaite pour les débutants car elle n'est pas coûteuse, assez puissante pour commencer et dispose de suffisamment d'entrées et de sorties pour la plupart des projets.

De plus, Uno est compatible avec presque toutes les cartes d'extension, ce qui lui donne d'énormes possibilités d'évolution [15].

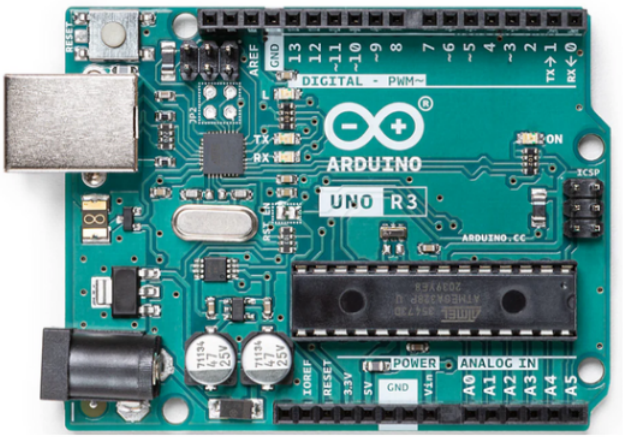


FIGURE 2.2 – Arduino Uno.

b.Caractéristiques techniques

Les principales caractéristiques de la carte Arduino Uno sont présentées dans le tableau suivant [4] :

Categorie	*Valeur*
Microcontrôleur	Atmega328
Fréquence d'horloge	$16MHz$
Tension de service	$5V$
Tension d'entrée (recommandée)	$7-12V$
Tension d'entrée (limites)	$6-20V$
Ports-numériques	14 E/S
Ports-analogiques	6 entrées analogiques
Courant maxi. par broche d'E/S	$40mA$
Courant maxi. par broche	$50mA$
Mémoire	$32Ko$ Flash
Interface	USB
Dimensions	$68 \times 53mm$

TABLE 2.2 – Caractéristiques techniques de la carte Arduino Uno.

c.Avantages et inconvénients

Avantages

1. Polyvalence : La Uno est largement utilisée et prise en charge, ce qui en fait un bon choix pour une variété de projets.
2. Grande communauté : Elle bénéficie d'une vaste communauté de développeurs et d'utilisateurs, ce qui facilite le partage de connaissances et de ressources.
3. Facilité d'utilisation : Elle est conviviale, même pour les débutants.
4. Abordable : Son prix est raisonnable pour les fonctionnalités qu'elle offre.

Inconvénients

1. Limitations matérielles : Comme la Nano, la Uno a des limitations en termes de puissance de traitement et de connectivité (peu d'entrées et mémoire insuffisante) [1].

2.2.2.3 Arduino Mega 2560

a.Définition

Comme son nom le suggère, la Méga 2560 est une carte plus grande que l'Uno. Elle est destinée à ceux qui en veulent plus : plus d'entrées, plus de sorties, et plus de puissance de calcul.

La Méga dispose de 54 broches numériques et de 16 broches analogiques, alors que l'Uno n'aligne que 15 broches numériques et 6 broches analogiques [13].

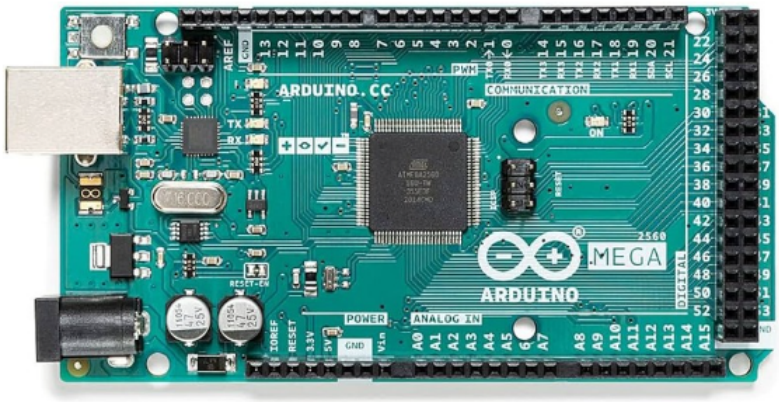


FIGURE 2.3 – Arduino Mega 2560.

b.Caractéristiques techniques

Les principales caractéristiques de la carte Arduino Mega 2560 sont présentées dans le Tableau 2.3 [13] :

Categorie	*Valeur*
Microcontrôleur	Atmega2560
Fréquence d'horloge	$16MHz$
Tension de service	$5V$
Tension d'entrée (recommandée)	$7-12V$
Tension d'entrée (limites)	$6-20V$
Ports numériques	54 E/S
Ports-analogiques	16 entrées analogiques
Courant maxi. par broche d'E/S	$40mA$
Courant maxi. par broche	$50mA$
Mémoire	$256Ko$ Flash
Interface	USB
Dimensions	$101 \times 53mm$

TABLE 2.3 – Caractéristiques techniques de la carte Arduino Mega 2560.

c.Avantages et inconvénients

Avantages

1. Plus de broches d'E/S : La Mega offre beaucoup plus de broches d'entrée/sortie que la Uno, ce qui la rend adaptée aux projets complexes nécessitant de nombreuses connexions.
2. Plus de mémoire : Elle dispose de plus de mémoire que la Uno, ce qui est utile pour les projets nécessitant un stockage de données important.
3. Puissance de calcul : Elle est plus puissante que la Uno, ce qui lui permet de gérer des tâches plus complexes.

Inconvénients

1. Encombrement : Sa taille plus grande peut être un inconvénient pour les projets nécessitant une empreinte plus petite.
2. Consommation d'énergie : Elle consomme plus d'énergie que les versions plus petites, ce qui peut ne pas convenir à certains projets alimentés par batterie.
3. Deux fois plus chère que l'Arduino Uno [1].

2.2.2.4 Arduino Due

a.Définition

L'Arduino Due est la nouvelle carte à microcontrôleur de la famille des cartes Arduino.

C'est la première carte basée sur un processeur ARM 32 bits, l' Atmel SAM3X8E ARM Cortex- M3. Il permet d'améliorer considérablement toutes les fonctionnalités standard et en ajoute de nouvelles.

Cette carte offre 54 entrées/sorties numériques (dont 12 utilisables en sorties PWM (Pulse Width Modulation) avec une résolution sélectionnable), 12 entrées analogiques de 12 bits de résolution, 4 UARTs (ports séries physiques), et 2 sorties analogiques CAN (Conversion Analogique Numérique), Oscillateur cristal 84 MHz, 2 connexions USB (Universal Serial Bus), un jack d'alimentation, un connecteur ICSP (In-circuit Serial Programming), un connecteur JTAG (Joint Test Action Group), et bouton Reset.

La tension maximale fournie ou supportée par les entrées/sorties est 3,3V. Travailler avec plus de tension, comme du 5V peut détruire la carte.

La carte Arduino Due dispose de 2 connecteurs USB, l'un au format micro-USB B, qui est le natif et fonctionne comme un USB host, c'est à dire que on peut connecter des périphériques externes compatibles USB comme une souris, un clavier, un smartphone. L'autre port USB au format type A est utilisé pour le debug.

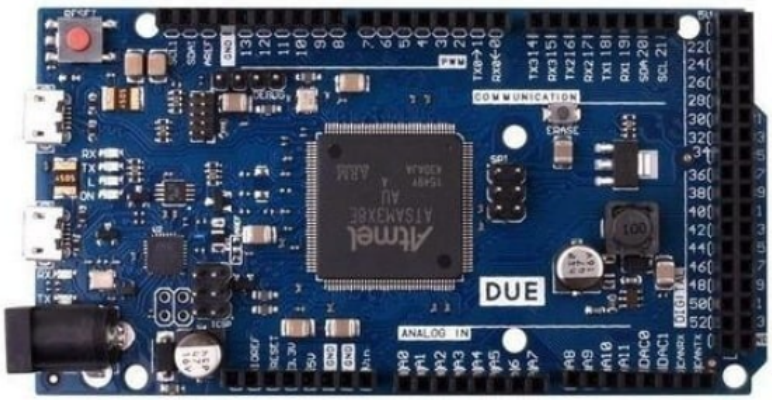


FIGURE 2.4 – Arduino Due.

b.Caractéristiques techniques

Les principales caractéristiques de la carte Arduino Due sont présentées dans le tableau suivant :

Categorie	*Valeur*
Microcontrôleur	AT91SAM3X8E
Fréquence d'horloge	$84MHz$
Tension de service	$3.3V$
Tension d'entrée (recommandée)	$7-12V$
Tension d'entrée (limites)	$6-20V$
Ports numériques	54 E/S (12 PWN)
Ports-analogiques	12 entrées 2 sorties (DAC)
Courant maxi. par broche d'E/S	$130mA$
Courant maxi. par broche	$800mA$
Mémoire	$512KB$ Flash
Interface	USB
Dimensions	$101 \times 53mm$

TABLE 2.4 – Caractéristiques techniques de la carte Arduino Due.

c.Avantages et inconvénients

Avantages

1. Puissance de traitement élevée : La Due est l'une des cartes Arduino les plus puissantes, avec un processeur ARM Cortex-M3 doté d'une fréquence de 84 MHz.
2. Plus de mémoire : Elle offre plus de mémoire que la plupart des autres cartes Arduino, ce qui est utile pour les projets nécessitant un stockage important ou une manipulation de données complexes.

Inconvénients

1. Prix : Elle est généralement plus chère que les autres cartes Arduino en raison de ses performances supérieures.
2. Complexité : Sa puissance accrue peut rendre la programmation et l'utilisation plus complexes pour les débutants.
3. Consommation d'énergie : Elle peut consommer plus d'énergie que les autres cartes Arduino en raison de sa puissance de traitement plus élevée [1].

2.2.3 Comparaison des types d'Arduino les plus populaires

Dans le tableau ci-dessous, nous mentionnons la plupart des types de cartes Arduino et montrons les différences entre eux :

Catégorie/Carte Arduino	*UNO R3*	*MEGA 2560*	*NANO*	*DUE*	*Esplora*	*Yun*
Microcontrôleur	ATmega 328	ATmega 2560	ATmega 328	AT31SA M3X8E	ATmega 32u4	ATmega 32u4
Fréquence d'horloge	16MHz	16MHz	16MHz	84MHz	16MHz	16MHz
Tension d'entrée (recommandée)	$7-12V$	$7-12V$	$7-12V$	$7-12V$	$7-12V$	$5V$
Tension de service	$5V$	$5V$	$5V$	$5V$	$5V$	$5V$
Ports numériques E/S	14/6	54/15	14/6	54/12	Non	20/7
Ports-analogiques E/S	6/0	16/0	8/0	12/2	Non	12/0
Mémoire vive(Flash)	$32ko$	$256ko$	$32ko$	$512ko$	$32ko$	$32ko$
Interface	USB-B male	USB-B male	Mini-USB	2 ports micro-USB	Micro-USB	Micro-USB
Carte SD	Non	Non	Non	Non	Non	Oui
Ethernet	Non	Non	Non	Non	Non	Oui
Wi-Fi	Non	Non	Non	Non	Non	Oui
Dimensions	$68 \times 53mm$	$101 \times 53mm$	$45 \times 18mm$	$101 \times 53mm$	$165 \times 60mm$	$68 \times 53mm$

Table 2.5 – Tableau de comparaison des cartes Arduino.

2.2.4 Choix de la carte Arduino

Dans les paragraphes précédents, nous avons examiné les caractéristiques techniques des cartes Arduino les plus populaires. Lorsqu'il s'agit de choisir la carte la plus adaptée à un projet spécifique, plusieurs critères doivent être pris en compte, notamment :

- La puissance de la CPU (Central Processing Unit).
- La capacité de stockage.
- La connectivité.
- La fréquence d'horloge.
- Le nombre d'entrées/sorties.

etc...

Après avoir évalué ces critères, nous avons opté pour la carte Arduino Due en raison de sa fréquence d'horloge élevée. Cette caractéristique garantit des performances de traitement supérieures, ce qui est essentiel pour notre projet vu les calculs complexes à effectuer lors d'une commande par régulateur T-S.

2.3 Logiciel de développement

Dans le domaine de l'électronique et du contrôle des systèmes, le choix du logiciel de développement peut jouer un rôle important dans la conception et la simulation de projets.

Les deux des outils les plus populaires utilisés par les ingénieurs et les amateurs sont l'IDE (Integrated Development Environment) Arduino et MATLAB-Simulink. Examinons en détail chacun de ces logiciels :

2.3.1 IDE Arduino

IDE Arduino est le logiciel officiel développé par Arduino.cc pour écrire, compiler et téléverser le code Arduino sur les cartes.

L'IDE Arduino est un environnement de développement open-source largement utilisé pour la programmation des microcontrôleurs Arduino .Il offre une interface convaincante, adaptée aux débutants comme aux utilisateurs expérimentés.

L'avantage du langage Arduino est qu'il est basé sur les langages C/C++ et supporte toutes les syntaxes standards du langage C et quelques-uns des outils du C++. En plus de très nombreuses librairies sont disponibles, gratuitement, pour communiquer avec le matériel connecté à la carte (Afficheurs LCD (Liquid Crystal Display), Afficheurs 7 segments, capteurs, servomoteurs... etc.).

Ce logiciel permet de :

1. Editer un programme : un programme est composé de croquis (sketch en Anglais), les programmes sont écrits en langage C.
2. Compiler ce programme dans le langage « machine » de l'Arduino, la compilation est une traduction du langage C vers le langage du microcontrôleur.
3. Téléverser le programme dans la mémoire de l'Arduino, le téléversement (upload) se passe via le port USB de l'ordinateur un fois dans la mémoire de l'Arduino.
4. Communiquer avec la carte Arduino grâce au terminal (ou moniteur série). pendant le fonctionnement du programme en mémoire sur l'Arduino, il peut communiquer avec l'ordinateur tant que la connexion est active (câble USB, ...)
5. Pour écrire un programme avec le langage Arduino, il faut respecter certaines règles. En effet, l'exécution d'un programme Arduino s'effectue de manière séquentielle, c'est-à-dire que les instructions sont exécutées les unes à la suite des autres, le compilateur vérifie l'existence de deux structures obligatoires à tout programme Arduino qui sont :
   - La partie initialisation et configuration des entrées/sorties ➔ la fonction setup()
   - La partie principale qui s'exécute en boucle ➔ la fonction loop()

Par contre, la partie déclaration des variables est optionnelle. La Figure 2.5 montre l'interface graphique de l'IDE ainsi que la structure d'un programme réalisé avec le langage Arduino [6].

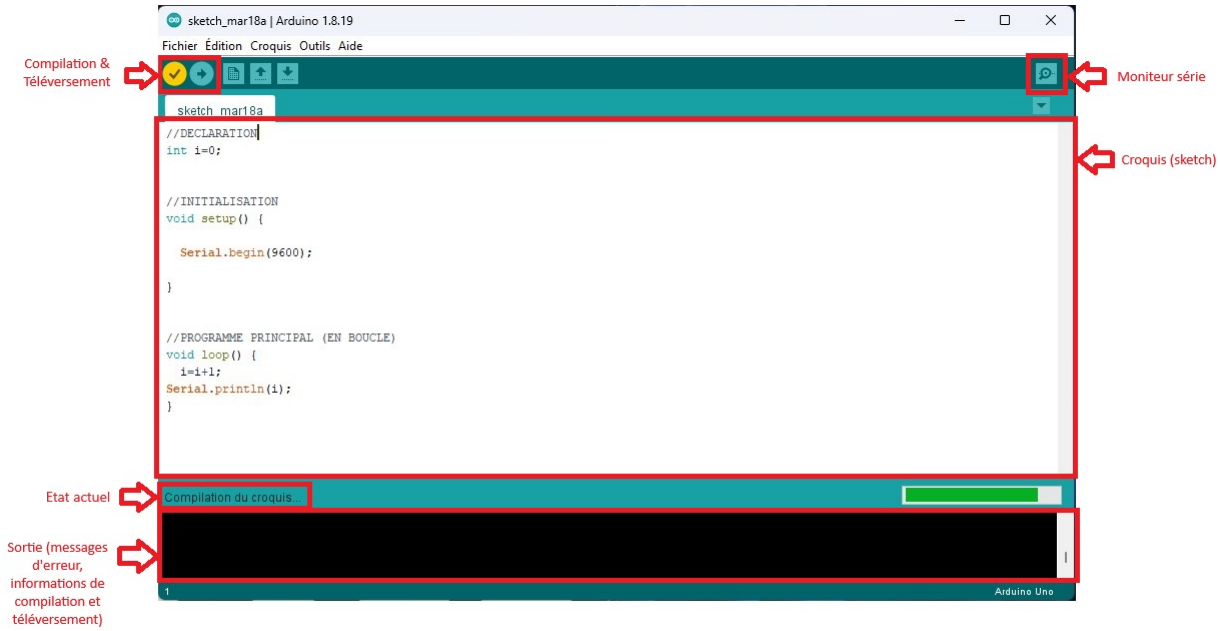


FIGURE 2.5 – Interface graphique du logiciel IDE Arduino.

Exemples de projets :

Dans cette partie, nous explorerons certains des projets les plus populaires réalisés avec l'IDE Arduino :

Station météo DIY : Les utilisateurs combinent souvent des capteurs de température, d'humidité et de pression avec un Arduino pour créer leur propre station météo. Ces projets peuvent afficher les données sur un écran LCD ou les transmettre via Wi-Fi pour un accès à distance.

Robot suiveur de ligne : Les passionnés de robotique explorent la création de robots capables de suivre des lignes tracées au sol en utilisant des capteurs de ligne infrarouges et des moteurs. L'Arduino contrôle les mouvements du robot en fonction des signaux des capteurs.

Système de contrôle d'éclairage automatisé : En combinant des relais et des capteurs de lumière, les utilisateurs peuvent créer un système qui ajuste automatiquement l'éclairage en fonction de la luminosité ambiante. Cela peut être utilisé pour économiser de l'énergie et améliorer le confort dans les maisons intelligentes.

2.3.2 MATLAB-Simulink de MathWorks

La programmation d'une carte Arduino est sensée être une tâche passionnante et agréable à accomplir, cependant, cela peut vite devenir désagréable dans le cas de collecte et traitement de données de capteurs, contrôle précis d'actionneurs ou réalisation de calculs complexes. Par conséquent, pour faciliter la programmation et travailler dans un environnement plus organisé, il serait plus judicieux d'opter pour l'utilisation de MATLAB-Simulink accompagné d'outils (add-ons) adaptés au projet.

MATLAB-Simulink est un environnement de modélisation et de simulation avancé utilisé pour la conception et l'analyse de systèmes dynamiques. Il offre une interface graphique convaincante, adaptée aux ingénieurs et aux chercheurs, leur permettant de modéliser des systèmes complexes de manière intuitive.

Les avantages de MATLAB-Simulink résident dans sa puissance de modélisation et sa polyvalence. En utilisant un paradigme de blocs fonctionnels interconnectés, les utilisateurs peuvent représenter visuellement les composants de leur système et définir les relations entre eux.

Ce logiciel offre des fonctionnalités qui permettent diverses applications, à savoir :

Modélisation des systèmes : Les utilisateurs peuvent modéliser des systèmes dynamiques en utilisant une variété de blocs fonctionnels prédéfinis pour représenter des composants tels que des systèmes mécaniques, électriques, hydrauliques, etc.

Simulation avancée : Simulink offre des fonctionnalités de simulation avancées, y compris la simulation en temps réel, la simulation hors ligne et la capacité à modéliser des systèmes complexes avec des comportements non linéaires.

Validation du système : Les utilisateurs peuvent valider leurs modèles en simulant le comportement du système dans différentes conditions et en comparant les résultats de simulation avec des données expérimentales ou théoriques.

Génération de code : Une fois que le modèle est validé, Simulink peut générer automatiquement du code C/C++ à partir du modèle, permettant une implémentation rapide et fiable sur des plates-formes matérielles telles que des microcontrôleurs.

Pour écrire un modèle avec MATLAB-Simulink, il faut suivre une approche similaire à l'IDE Arduino en respectant les règles de modélisation. Il est nécessaire de définir les entrées et les sorties du système, configurer les blocs fonctionnels pour représenter les composants du système, et définir les conditions initiales et les paramètres de simulation.

Ensuite, il est possible d'exécuter la simulation pour analyser le comportement du système et ajuster les paramètres si nécessaire.

La Figure 2.6 montre l'interface graphique de MATLAB-Simulink :

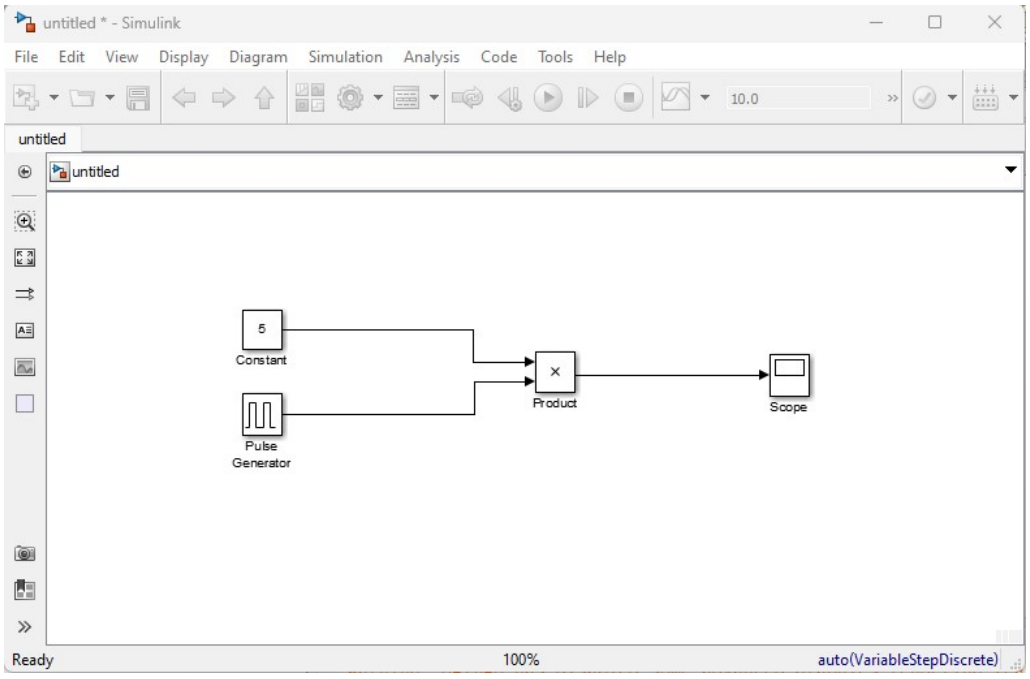


FIGURE 2.6 – Interface graphique de Matlab Simulink.

2.3.2.1 Simulink support package pour Arduino

Pour assurer la communication entre Simulink et la carte, il est nécessaire d'ajouter le paquet "Simulink Support Package for Arduino".

Ce paquet s'obtiens sur le site de MATLAB, il suffit de télecharger l'installeur du paquet et de le lancer dans MATLAB. La création d'un compte Mathworks est nécessaire.

Ce paquet permet de développer des algorithmes sous Simulink à l'aide de blocs pour la modélisation de systèmes ainsi que leurs commande, les blocs sont utilisés pour configurer et accéder aux périphériques connectés à la carte Arduino, nous pouvons même ajouter un bloc représentant un programme MATLAB.

Après avoir réalisé le modèle Simulink, nous pouvons le simuler et régler les paramètres de l'algorithme pour obtenir le fonctionnement voulu. Suite à ça, il est possible de charger l'algorithme final dans la mémoire de la carte Arduino pour un fonctionnement indépendant (standalone), ce qui signifie qu'il ne sera plus nécessaire de gérer le fonctionnement via ordinateur [8].

Exemples de projets

Système de contrôle de drone : Les ingénieurs utilisent MATLAB Simulink pour modéliser et simuler des systèmes de contrôle de drones, y compris la stabilisation, la navigation et la gestion de la batterie. Cela permet de tester et d'optimiser les algorithmes de contrôle avant le déploiement réel.

Système de gestion de la production : Les professionnels de l'industrie utilisent Simulink pour modéliser et simuler des systèmes de gestion de la production, y compris les lignes de fabrication, les robots industriels et les processus automatisés. Cela permet d'analyser les performances du système et d'optimiser les opérations.

Système de contrôle de véhicule autonome :MATLAB Simulink est utilisé pour modéliser et simuler des systèmes de contrôle de véhicules autonomes, y compris la perception, la planification et le contrôle. Les ingénieurs peuvent tester différents scénarios de conduite et valider les algorithmes de contrôle avant les essais sur route.

2.3.3 Tableau comparatif

Dans le tableau suivant, nous comparons les caractéristiques pertinentes des deux logiciels cités (IDE Arduino et MATLAB Simulink) :

Critères	IDE Arduino	MATLAB-Simulink
Modélisation et Simulation	Limitée à la programmation et au contrôle	Interface graphique puissante dédiée à la modélisation et la simulation avancée des systèmes dynamiques
Convivialité	Convivial pour les débutants	Nécessite un certain rythme d'apprentissage en raison de sa complexité
Développement Direct	Développement direct sur les microcontrôleurs Arduino	Pas de développement direct, mais possibilité de générer le code pour Arduino
Simulation Avancée	Non disponible	Simulation en temps réel, validation du contrôleur, analyse des performances
Génération de Code	Génération directe du code pour Arduino	Génération du code C/C++ prêt pour Arduino à partir des modèles de simulation
Précision des Modèles	Limitée en termes de modélisation et de précision	Permet une modélisation précise et une simulation avancée des systèmes dynamiques
Optimisation du Contrôleur	Limitée aux paramètres de contrôle dans le code	Permet l'optimisation des paramètres du contrôleur avec différents outils spécifiques
Cas D'utilisation Courants	Projets électroniques DIY (Do it yourself), prototypage rapide	Modélisation et simulation de systèmes dynamiques, contrôle avancé, applications industrielles
Communauté et Support	Grande communauté d'utilisateurs et support actif en ligne (forums de discussion)	Support technique robuste, documentation étendue, forums de discussion

Table 2.6 – Tableau de comparaison entre l'IDE Arduino et MATLAB Simulink.

2.3.4 Choix du logiciel

Dans les titres précédents, nous avons examiné les caractéristiques de chacun des logiciels. Pour le choix du logiciel le plus adapté à notre projet, nous avons pris en compte plusieurs critères, notamment :

- La génération automatique de code pour Arduino
- La modélisation et simulation intuitive des systèmes complexes.
- La simplicité d'utilisation.
- La simulation en temps réel

etc...

Après avoir étudié la convenabilité des logiciels pour notre utilisation précise, nous avons opté pour MATLAB-Simulink car nous bénéficions d'une modélisation précise, d'une simulation avancée, d'une validation efficace du contrôleur et d'une génération de code automatisée, ce qui nous permet d'obtenir des résultats fiables et optimisés dans un délai plus court et dans un environnement de travail organisé et intuitif.

2.4 Conclusion

Après avoir exploré en profondeur l'univers d'Arduino, nous pouvons conclure que cette plateforme de prototypage électronique offre une multitude de possibilités pour les amateurs et les professionnels de l'électronique. Nous avons découvert différentes gammes de cartes Arduino, chacune offrant des fonctionnalités uniques adaptées à divers projets. De la petite et économique Arduino Nano à la puissante Arduino Due. De plus, nous les avons comparées pour nous aider à choisir celle qui convient le mieux à notre projet.

Ensuite, nous avons examiné de près les options de logiciel de développement, mettant en évidence l'IDE Arduino et MATLAB-Simulink comme deux choix principaux pour la programmation des cartes Arduino.

Enfin, nous avons choisi la carte Arduino Due pour sa puissance et polyvalence parfaitement adaptée aux exigences de notre projet de commande d'un moteur à courant continu. De plus, en sélectionnant MATLAB-Simulink comme logiciel de développement, nous avons accès à des outils avancés de modélisation, de simulation et de génération de code, nous permettant de concevoir et de tester notre système avec précision et efficacité.

Chapitre 3

Implémentation d'un régulateur flou sur une carte Arduino Due

3.1 Introduction

Dans ce chapitre, nous allons présenter les techniques utilisées pour concevoir le régulateur Takagi-Sugeno ainsi que sa structure. Puis nous l'implémenterons sur la carte Arduino en utilisant l'environnement Simulink de MATLAB, afin de commander notre moteur. L'objectif de ce chapitre est d'implémenter un réglateur flou sur une carte Arduino pour la commande d'un moteur à courant continu.

D'abord ,Nous allons identifier le système en utilisant le "System Identification Toolbox" de MATLAB pour créer un modèle mathématique approximatif sous forme de fonction de transfert. Puis, nous allons réaliser un montage permettant à la fois de commander le moteur et de visualiser les données requises. De plus, pour comparer les résultats, nous concevrons un régulateur PID (Proportionnel Intégral Dérivé) à l'aide d'un outil MATLAB.

Ensuite, nous allons procéder à des simulations et des tests pratiques en utilisant des schémas de commande pour évaluer le comportement du système et ajuster les paramètres des régulateurs.

Enfin, nous implémentons les deux régulateurs sur la carte Arduino pour comparer leurs réactions.

3.2 Structure du régulateur

La structure du régulateur est composée de quatre parties élémentaires : la fuzzification, l'établissement des règles floues, l'inférence et la défuzzification.

Nous allons passer par chacune de ces étapes dans le contexte de la régulation de vitesse d'un moteur à courant continu.

3.2.1 Univers de discours

Comme dit dans le premier Chapitre, l'univers de discours représente intervalle sur lequel s'étend la variable linguistique. Ce dernier sera de $[-1, 1]$.

3.2.2 Entrées du régulateur

Pour la conception de notre régulateur, nous commençons par choisir ses entrées.

Généralement, dans le cas d'une régulation de ce type, l'erreur e représente l'entrée du régulateur et la commande u en est la sortie. Cependant, pour nous permettre d'obtenir de meilleurs résultats, nous ajoutons la dérivée de l'erreur de aux entrées. De cette façon, le régulateur possède de plus de données, cela nous permet de mieux contrôler la commande et d'augmenter la précision.

a.Erreur de Vitesse (e) :

La différence entre la vitesse souhaitée (référence) et la vitesse actuelle du moteur. Cette variable indique si la vitesse actuelle est inférieure ou supérieure à la vitesse souhaitée.

Une erreur positive signifie que la vitesse actuelle est inférieure à la vitesse de consigne, nécessitant une augmentation de la vitesse.

Une erreur négative indique que la vitesse actuelle est supérieure à la vitesse de consigne, nécessitant une réduction de la vitesse.

b.Variation de l'Erreur de Vitesse (de) :

La dérivée de l'erreur de vitesse par rapport au temps, représente le changement de l'erreur de vitesse.

Une variation positive de de signifie que l'erreur augmente avec le temps, suggérant que la vitesse actuelle s'éloigne de la vitesse de consigne.

Une variation négative signifie que l'erreur diminue, suggérant que la vitesse actuelle se rapproche de la vitesse de consigne.

Exemple :

Lorsque le moteur n'a pas atteint la vitesse exigée mais qu'il est en phase d'accélération, l'erreur est positive et la dérivée de l'erreur négative. Dans ce cas, deux possibilités s'offrent à nous, nous pouvons programmer le régulateur de sorte à ce que sa sortie soit positive pour atteindre la vitesse souhaitée plus vite ou une sortie nulle pour laisser le moteur l'atteindre plus lentement.

Par contre, si nous avions seulement l'erreur comme entrée, nous n'aurions qu'une seule possibilité et ce serait la sortie positive.

3.2.3 Sortie du régulateur

La sortie du régulateur flou est calculée par la défuzzification qui prend en compte les résultats des inférences et des fonctions linéaires ou des constantes pour obtenir une seule valeur.

Dans notre cas, nous devons avoir trois constantes pour couvrir tous les cas de figure envisageables (sortie positive, négative ou nulle). Pour une meilleure précision, nous choisissons d'ajouter deux valeurs intermédiaires en plus des trois requises.

Les différentes constantes de sortie sont définies comme suit :

- GN (Grand Négatif) = -1
- N (Négatif) =-0.5
- Z (Zero) = 0
- P (Positif) = 0.5
- GP (Grand Positif) = 0

Une fois la valeur de la sortie calculée, elle est ajoutée à la commande de l'itération précédente pour devenir la nouvelle valeur de la commande où :

Une sortie positive peut indiquer une augmentation de l'énergie fournie à la machine pour augmenter sa vitesse.

Une sortie négative peut indiquer une réduction de l'énergie fournie ou l'application d'un freinage pour réduire la vitesse.

3.2.4 Fonctions d'appartenance

En second lieu, nous choisissons les fonctions d'appartenance pour l'erreur e et sa dérivée de.

Dans notre cas, il est plus convenable d'utiliser des méthodes basées sur le raisonnement humain et sur les conditions d'utilisation de ces fonctions, puis d'effectuer des ajustements en fonction des résultats pratiques obtenus [9].

Pour chaque variable d'entrée (Erreur de vitesse (e) et Variation de l'erreur de vitesse (de), nous définissons plusieurs ensembles flous pour couvrir chacune des situations possibles.

a.Erreur de Vitesse (e)

N (Négative)

$$\mu_N(x) = \begin{cases} 1 & x < -1 \\ x & -1 \leq x < 0 \\ 0 & x \geq 1 \end{cases} \tag{3.1}$$

Z (Zéro)

$$\mu_Z(x) = \begin{cases} 0 & x < -1 \\ x+1 & -1 \leq x < 0 \\ 1-x & 0 \leq x < 1 \\ 0 & x > 1 \end{cases} \tag{3.2}$$

P (Positive)

$$\mu_P(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ 1 & x \geq 1 \end{cases} \tag{3.3}$$

b.Variation de l'Erreur de Vitesse (de)

Nous appliquons le même raisonnement pour le choix des fonctions pour de.

N (Négative)

$$\mu_N(x) = \begin{cases} 1 & x < -1 \\ x & -1 \leq x < 0 \\ 0 & x \geq 1 \end{cases} \tag{3.4}$$

Z (Zéro)

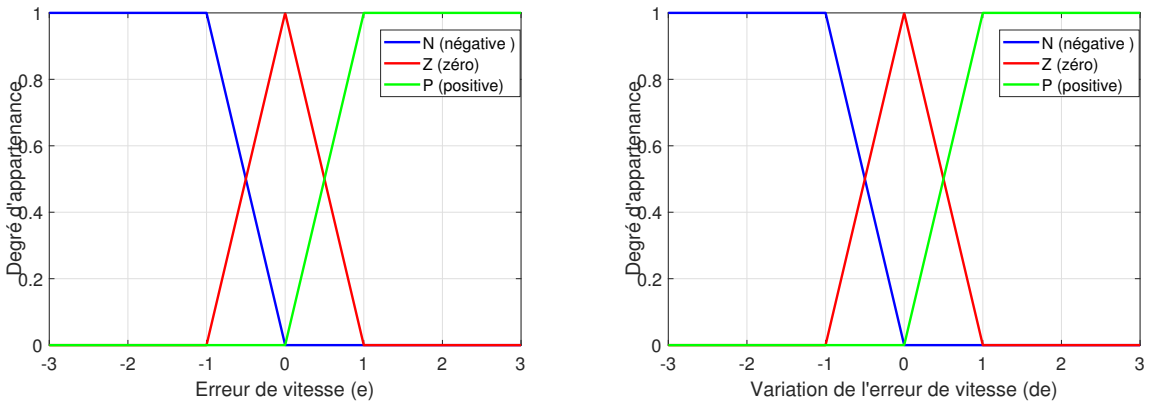
$$\mu_Z(x) = \begin{cases} 0 & x < -1 \\ x+1 & -1 \leq x < 0 \\ 1-x & 0 \leq x < 1 \\ 0 & x > 1 \end{cases} \tag{3.5}$$

P (Positive)

$$\mu_P(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ 1 & x \geq 1 \end{cases} \tag{3.6}$$

Comme dit dans le premier chapitre, le chevauchement des fonctions d'appartenance permet une commande plus lisse et robuste. En ce qui concerne les deux fonctions N et P, nous les prenons sous forme trapézoïdale, la fonction Z quant a elle sera triangulaire.

La figure ci-dessous montre le tracé des fonctions d'appartenance :



(a) (b)

FIGURE 3.1 – Fonctions d'appartenance de : (a) l'erreur de vitesse et (b) la dérivée de l'erreur de vitesse.

3.2.5 Règles d'inférence

Compte tenu des deux variables d'entrée, chacune appartenant à trois fonctions d'appartenance, le nombre total de combinaisons possibles est $3 \times 3 = 9$. Nous devons donc concevoir neuf règles d'inférence pour couvrir toutes les combinaisons.

Elles peuvent être symboliquement représentées comme suit :

Si (e est N) ET (de est N) *Alors* (u est GN)

Si (e est N) ET (de est Z) *Alors* (u est N)

Si (e est N) ET (de est P) *Alors* (u est Z)

Si (e est Z) ET (de est N) *Alors* (u est N)

Si (e est Z) ET (de est Z) *Alors* (u est Z)

Si (e est Z) ET (de est P) *Alors* (u est P)

Si (e est P) ET (de est N) *Alors* (u est Z)

Si (e est P) ET (de est Z) *Alors* (u est P)

Si (e est P) ET (de est P) *Alors* (u est GP)

3.2.6 Défuzzification

Il existe plusieurs méthodes de défuzzification, mais nous nous concentrerons ici sur la méthode de la moyenne pondérée (Weighted Average), qui est couramment utilisée en raison de sa simplicité et de son efficacité.

Voici donc sa formule mathématique ;

$$S = \frac{\sum_{i=1}^{N} c_i \mu_i}{\sum_{i=1}^{N} \mu_i} \tag{3.7}$$

où :

S : la valeur numérique de sortie.

N : le nombre de règles.

μ_i : le degré de vérité.

c_i : le conséquent.

Dans cette méthode, chaque valeur de sortie de la fonction d'appartenance est pondérée par son degré d'appartenance correspondant, puis une moyenne pondérée est calculée pour obtenir la valeur de sortie défuzzifiée.

3.3 Application expérimentale

Le moteur que nous avons utilisé est un moteur à courant continu conçu pour une utilisation dans des applications nécessitant une lecture précise de la position, comme le feedback des mouvements d'un robot. Il est équipé d'un encodeur à effet Hall monté sur l'arbre, fournissant une sortie en quadrature pour une précision optimale, cet encodeur génère 230 impulsions par tour. Cela signifie qu'il génère deux signaux décalés de 90 degrés, permettant de déterminer la direction et la position angulaire (vitesse angulaire) de l'arbre du moteur.

Caractéristiques Techniques :

- Tension Nominale : $12V$.
- Courant Nominal : $200mA$.
- Rapport de Transmission : 1 : 19.
- Couple : $295g - cm$ $(0,029Nm)$.
- Encodeur : Effet Hall avec une résolution de 230 impulsions par tour.
- Dimensions (sans arbre) : $22mm$ de diamètre x $68mm$ de longueur.
- Dimensions de l'Arbre : $4mm$ de diamètre x $12mm$ de longueur (forme en D).
- Trous de Montage : 4 trous $M2$ x $4,0mm$ espacés de $18mm$ (centre à centre).

FIGURE 3.2 – Moteur à courant continu $12V$ avec encodeur.

3.3.1 Identification du système

La période échantillonnage a un effet direct sur le temps de réponse du régulateur, nous devons donc choisir la période la plus courte possible.

Pour la choisir, nous sommes conditionnés par le microcontrôleur et l'encodeur. Si la période est trop courte, notre système risque de dysfonctionner occasionnellement ou même de ne pas fonctionner du tout. Après de nombreux tests, nous avons choisi une période échantillonnage de $Fe = 0.02$ car elle représente le compromis idéal.

Pour commander ce moteur, nous devons installer le package "Arduino Hardware Support". Ce dernier ajoute des blocs permettant d'envoyer ou de lire des signaux sur les pins de la carte.

L'image ci-dessous montre les blocs les plus utilisés de ce package :

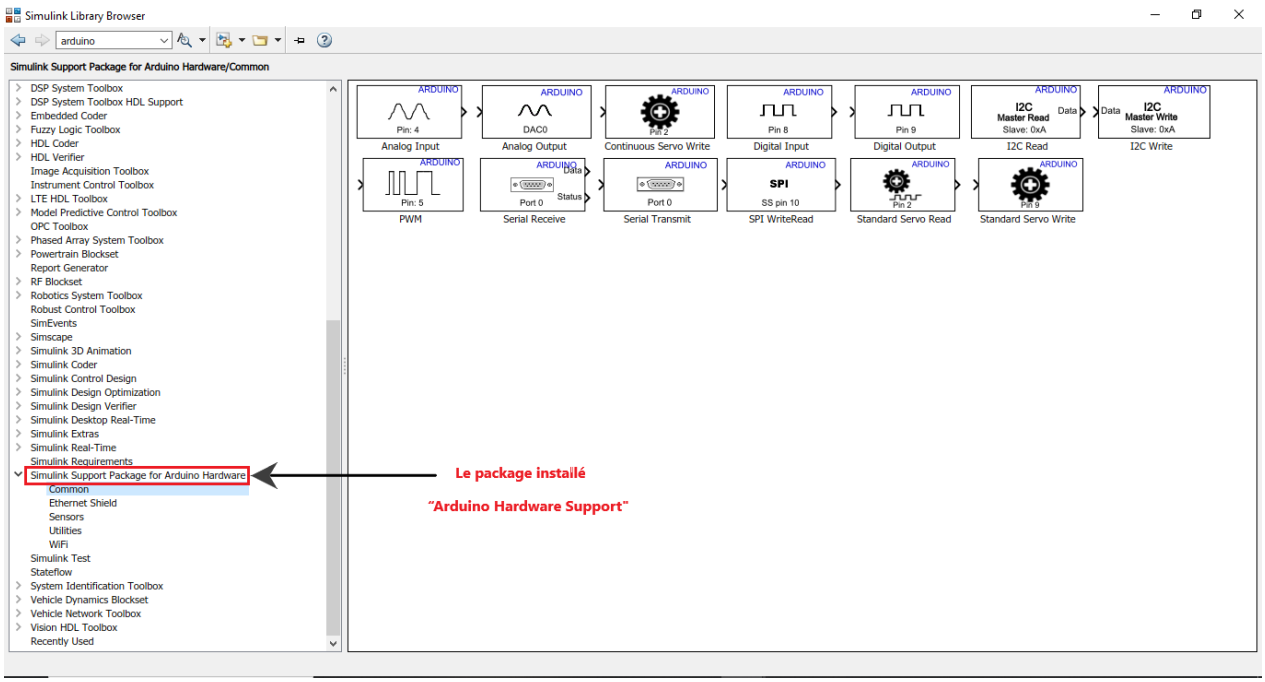


FIGURE 3.3 – Blocs ajoutés par le package "Arduino Hardware Support".

A l'aide de ces blocs, nous programmons la partie commande de notre fichier d'abord. D'après le branchement, le pin 9 transmet le signal PWM et les pins 10 et 11 le sens de rotation.

L'image suivante illustre le schéma de commande :

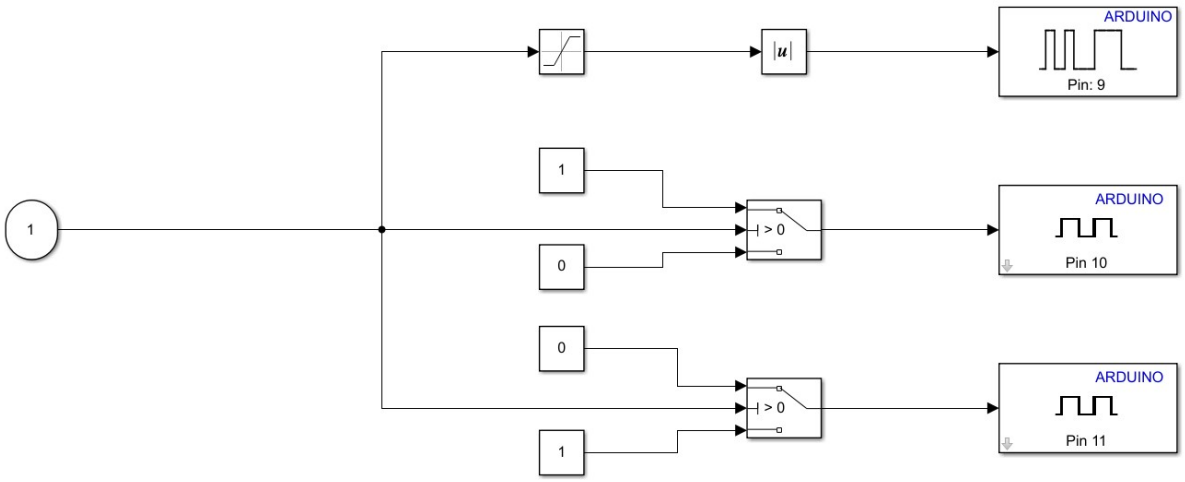


FIGURE 3.4 – Schéma de commande.

De plus, MATLAB met à notre disposition plusieurs fonctions mathématiques, d'ingénierie et de traçage intégré pour analyser et visualiser les données collectées à partir de notre carte Arduino.

Pour obtenir une fonction de transfert représentant approximativement le comportement de notre moteur. Nous avons envoyé une commande u au moteur, en mesurant la vitesse v atteinte par le moteur en parallèle, de cette façon, nous possédions suffisamment de données pour en déduire une fonction de transfert.

L'image ci-dessous montre le schéma utilisé pour l'identification du système :

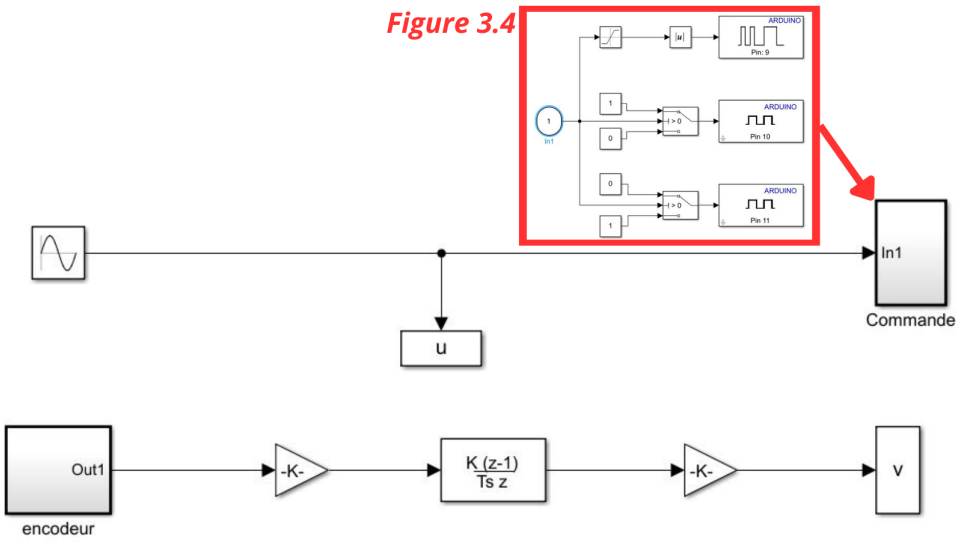


FIGURE 3.5 – Schéma fonctionnel du processus d'identification sur Simulink.

Après avoir collecté les données u et v, nous avons utilisé le "System Identification Toolbox" de MATLAB pour calculer un modèle mathématique de notre système sous forme de fonction de transfert. Ce processus comprend l'importation des données, l'estimation des paramètres de la fonction de transfert et la validation du modèle en comparant les sorties simulées avec les données réelles.

L'image ci-dessous montre cette interface :

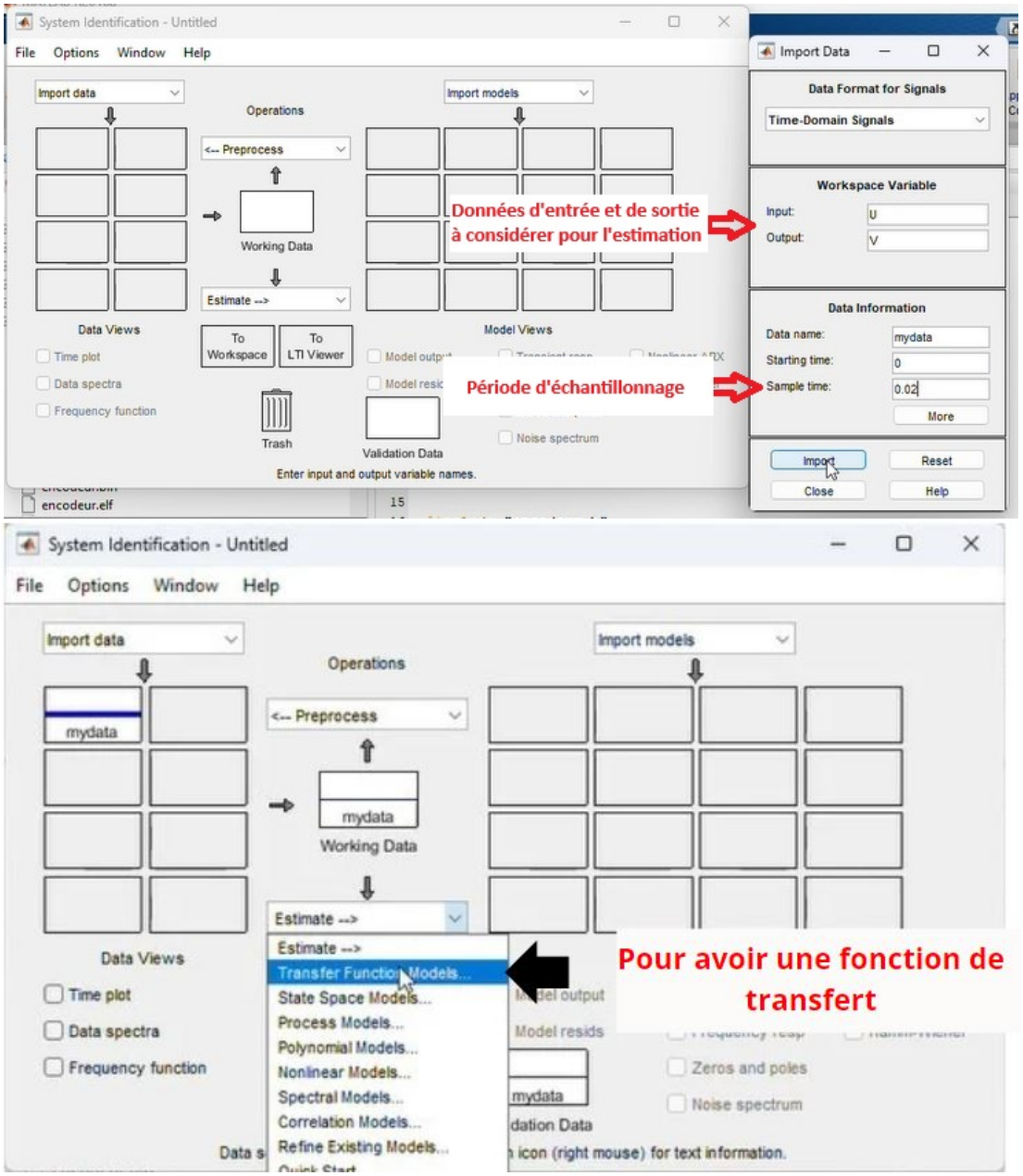


FIGURE 3.6 – L'interface de "System Identification Toolbox"

Après identification, nous avons obtenu la fonction de transfert suivante :

$$F(z) = \frac{0.0001546z + 0.0001525}{z^2 - 1.958z + 0.9599} \tag{3.8}$$

3.3.2 Conception du régulateur flou sur Simulink

a.Schéma de simulation

Une fois que nous avons obtenu le modèle mathématique de notre système sous forme de fonction de transfert, nous l'avons utilisé pour concevoir et simuler une boucle de régulation floue de type Takagi-Sugeno.

L'outil "Fuzzy Logic Designer" permet de définir les variables d'entrée et de sortie, de tracer les fonctions d'appartenance, d'établir les règles floues et de simuler les performances du contrôleur flou avant sa mise en œuvre.

L'image suivante montre cette interface :

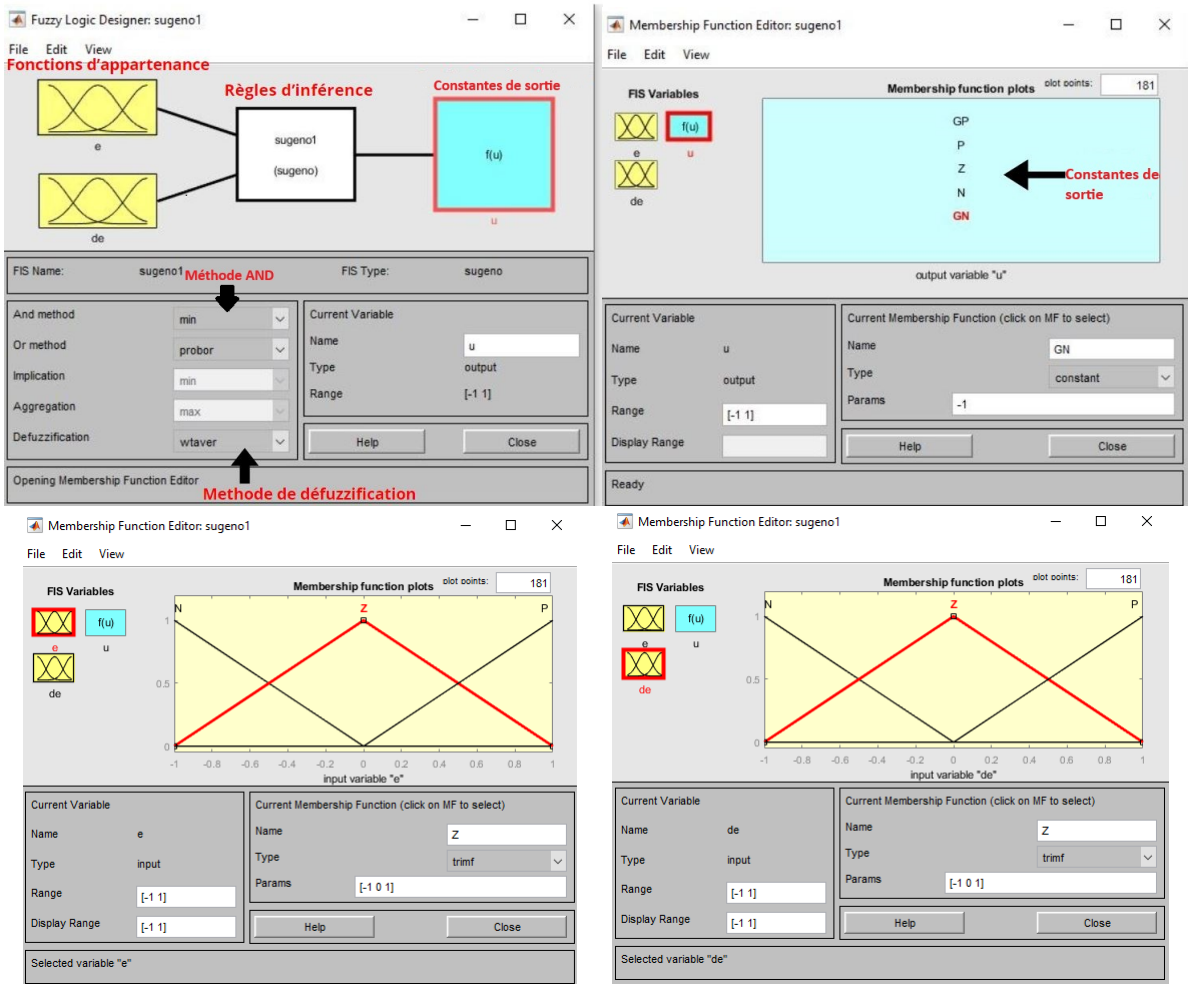


FIGURE 3.7 – Interface de l'outil "Fuzzy Logic Designer".

En ajoutant des gains de normalisation, nous avons rendu la manipulation des entrées et de la sortie du régulateur plus simple. Cela se traduit par un réglage plus rapide. Grâce à ces gains, le régulateur est mieux ajusté pour réagir de manière appropriée aux variations du système. Les gains choisis pour e, de et la sortie du régulateur sont ; 0.05, 1 et 2 respectivement.

L'image ci-dessous montre le schéma de simulation du régulateur Takagi-Sugeno :

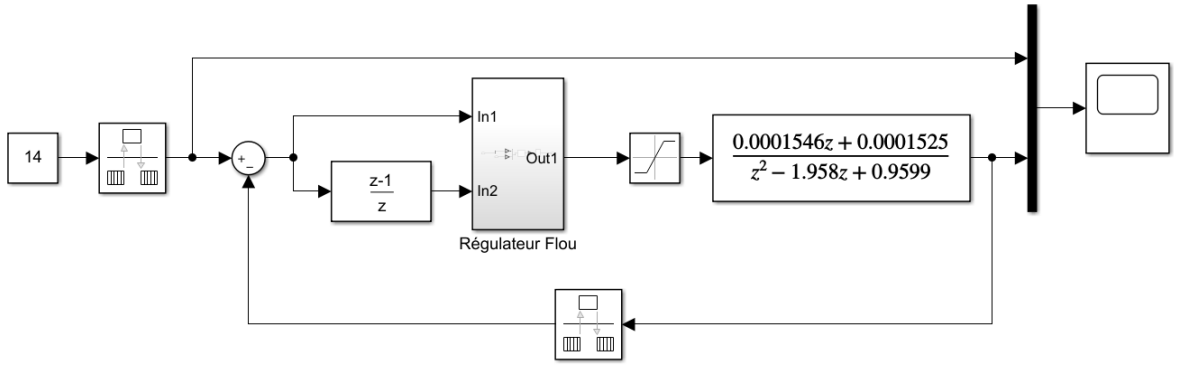


FIGURE 3.8 – Schéma de simulation du régulateur flou de type Takagi-Sugeno.

b.Schéma de commande

Une fois que la boucle de simulation est établie dans Simulink, la prochaine étape consiste à intégrer ce régulateur dans un schéma de commande exécuté par la carte Arduino Due qu'on a choisit dans le deuxième chapitre grâce a sa fréquence d'horloge élevée.

Afin d'exécuter le système de commande en mode externe, nous avons paramétré le fichier Simulink pour que le code généré soit adapté à la carte Arduino Due, et que le solveur soit discret "discrete (no continuous states)" avec un pas fixe "Fixed-step".

L'image ci-dessous montre la fenêtre de paramétrage :

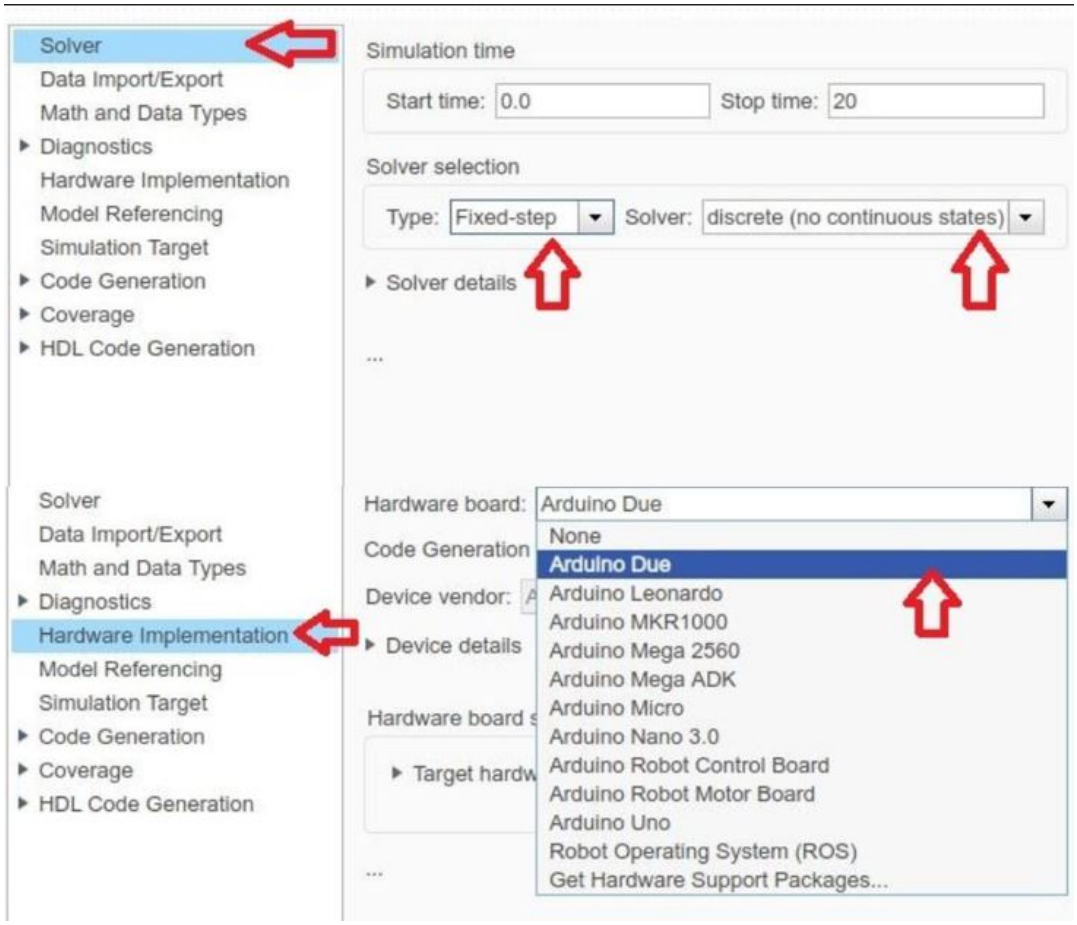


FIGURE 3.9 – Fenêtre de paramétrage des fichiers Simulink.

Ensuite, nous avons conçu un nouveau schéma de commande permettant une application expérimentale qui utilise les mesures de l'encodeur pour calculer la vitesse du moteur et se base sur cette dernière pour calculer l'erreur. Nous y avons inséré le même bloc "Fuzzy Logic Controller" utilisé précédemment pour tester le fonctionnement puis réajusté les gains de normalisation pour convenir à l'utilisation pratique.

L'image ci-dessous montre le schéma de commande appliqué sur le moteur réel :

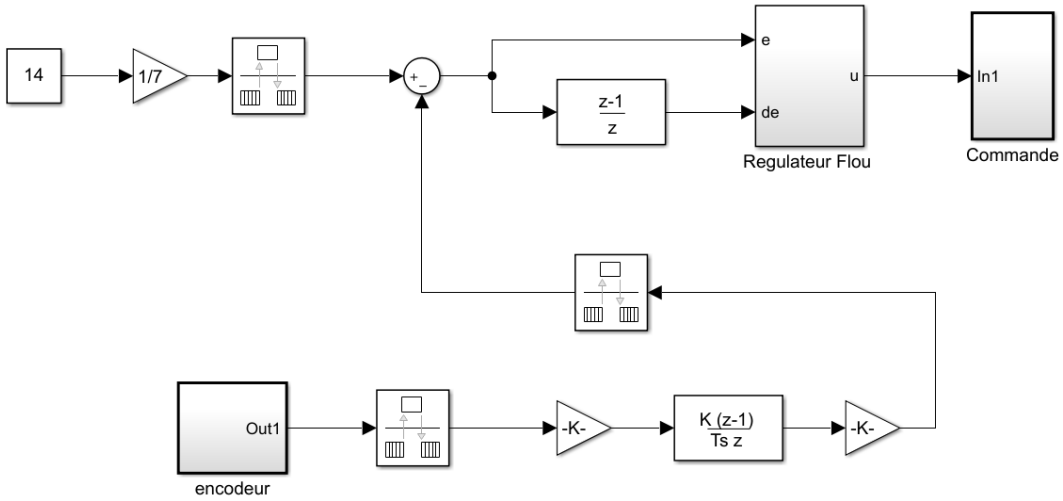


FIGURE 3.10 – Schéma de commande floue appliqué au moteur.

Le bloc "Commande" est identique à celui utilisé en identification, voir Figure 3.4.

3.3.3 Conception du régulateur PID

a.Schéma de simulation

Après avoir conçu et testé avec succès le schéma de contrôle Takagi-Sugeno, nous avons opté pour un contrôleur classique PID afin de comparer les performances des deux méthodes de régulation. Cela en remplaçant le contrôleur T-S par un contrôleur PID.

L'image ci-dessous montre le schéma de simulation du regulateur PID :

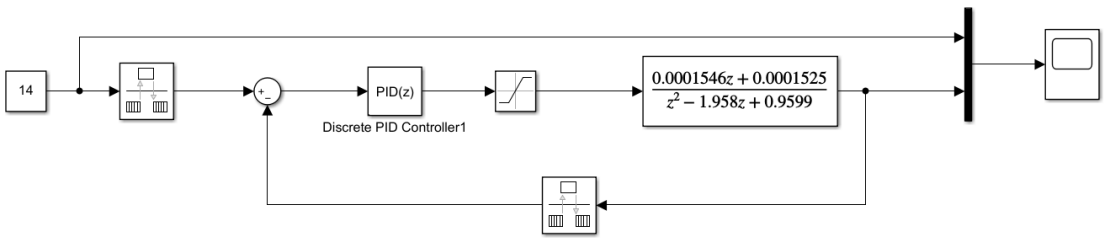


FIGURE 3.11 – Schéma de simulation du régulateur PID.

Pour le réglage du contrôleur, nous avons utilisé l'outil "PID Tuner" dans Simulink. Les paramètres du PID peuvent être ajustés en temps réel à l'aide des boutons et curseurs de l'interface. Pendant cette étape, on voit immédiatement comment le système simulé réagit aux nouveaux paramètres. Ces paramètres sont :

- Proportionnel (P) : 0.635857522081545.
- Intégral (I) : 63.5857522081546.
- Dérivé (D) : 0.
- Avec un filtre de coefficient (N) : 100.

L'image ci-dessous montre cette interface :

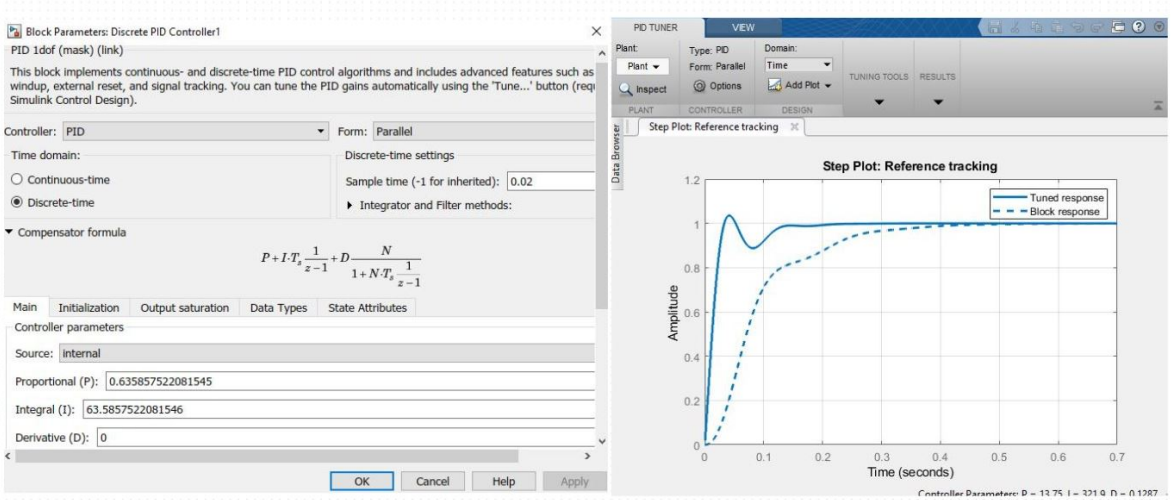


FIGURE 3.12 – Fenêtre de paramétrage de l'outil "PID Tuner".

b.Schéma de commande :

Après avoir conçu le régulateur PID, nous reprenons la boucle de commande utilisée précèdemment en remplaçant le bloc "Fuzzy Logic Controller" par "Discrete PID Controller" pour le tester en pratique.

L'image ci-dessous montre le schéma de commande du régulateur :

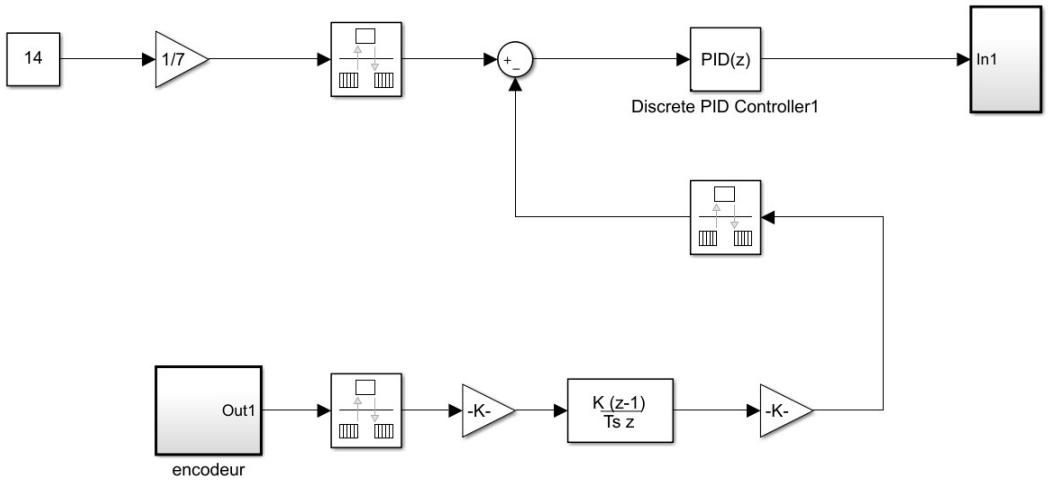


FIGURE 3.13 – Schéma de commande PID sur Simulink.

3.3.4 Implémentation sur la carte Due

Pour l'implémentation, nous utilisons les mêmes boucles que pour la commande.

Étant donné que le bloc "Fuzzy Logic Controller" utilisé précédemment ne peut pas être implémenté sur la carte, nous avons dû programmer le régulateur en utilisant des blocs basiques de Simulink.

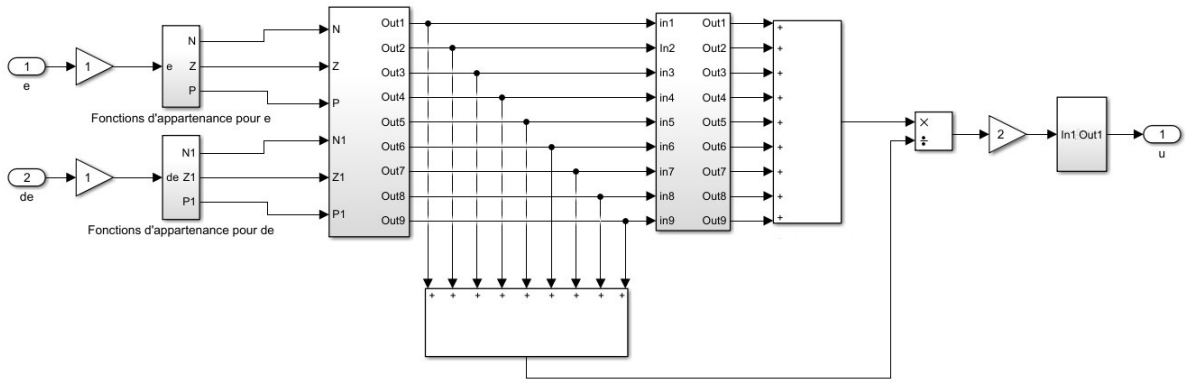


FIGURE 3.14 – Régulateur flou "Takagi-Sugeno" conçu avec les blocs de Simulink.

Nous avons intégré une deuxième carte Arduino (Arduino Uno) pour lire les résultats étant donné qu'il n'est pas possible de le faire avec un programme implémenté, de plus, cela réduit la charge sur la carte responsable des calculs (Arduino Due).

Les deux cartes partagent les connexions suivantes :

1. L'encodeur pour la transmission des données de position.
2. La GND pour assurer une référence commune pour les signaux.

Nous connectons le PC1 à la carte Due pour l'implémentation des algorithmes de contrôle, tandis que le PC2 et la Arduino Uno sont utilisés pour visualiser les résultats.

L'image ci-dessous montre le schéma de branchement entre les deux cartes Arduino et le moteur :

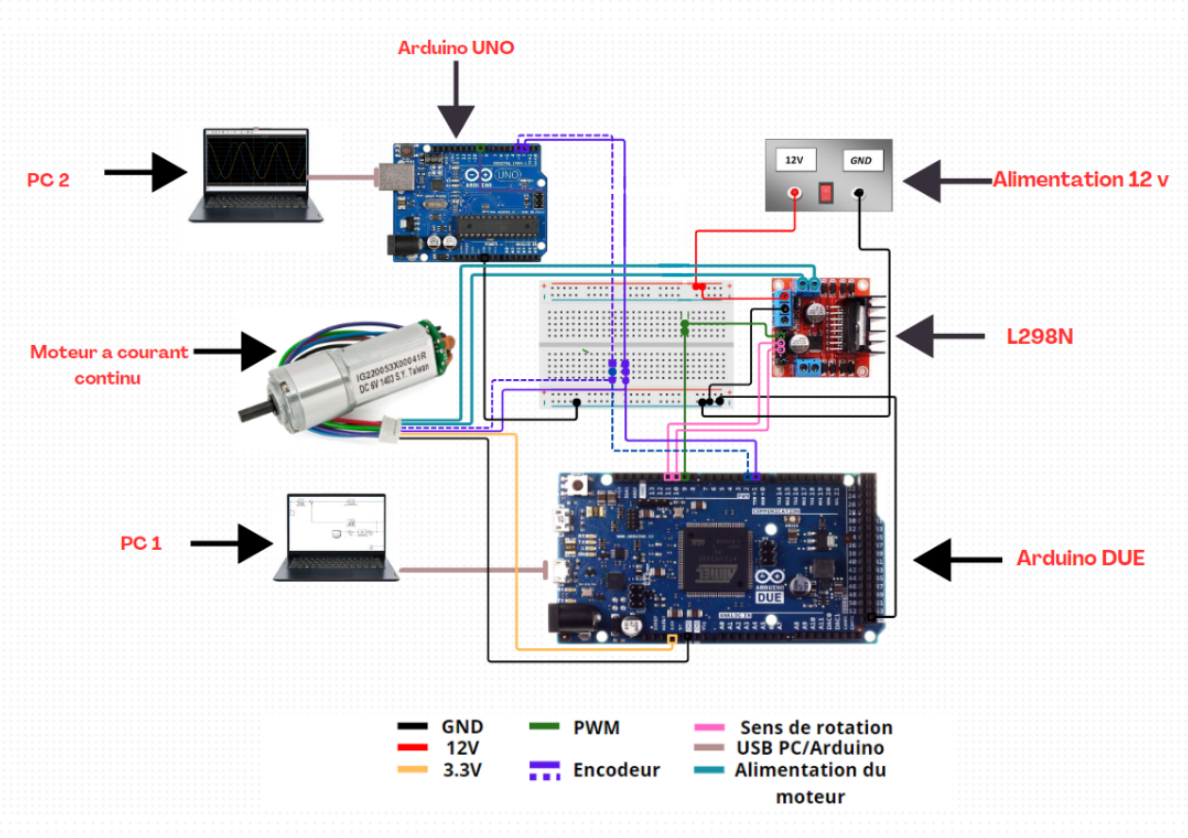


FIGURE 3.15 – Schéma de câblage du moteur avec les cartes Arduino.

L'image suivante montre le montage réalisé :

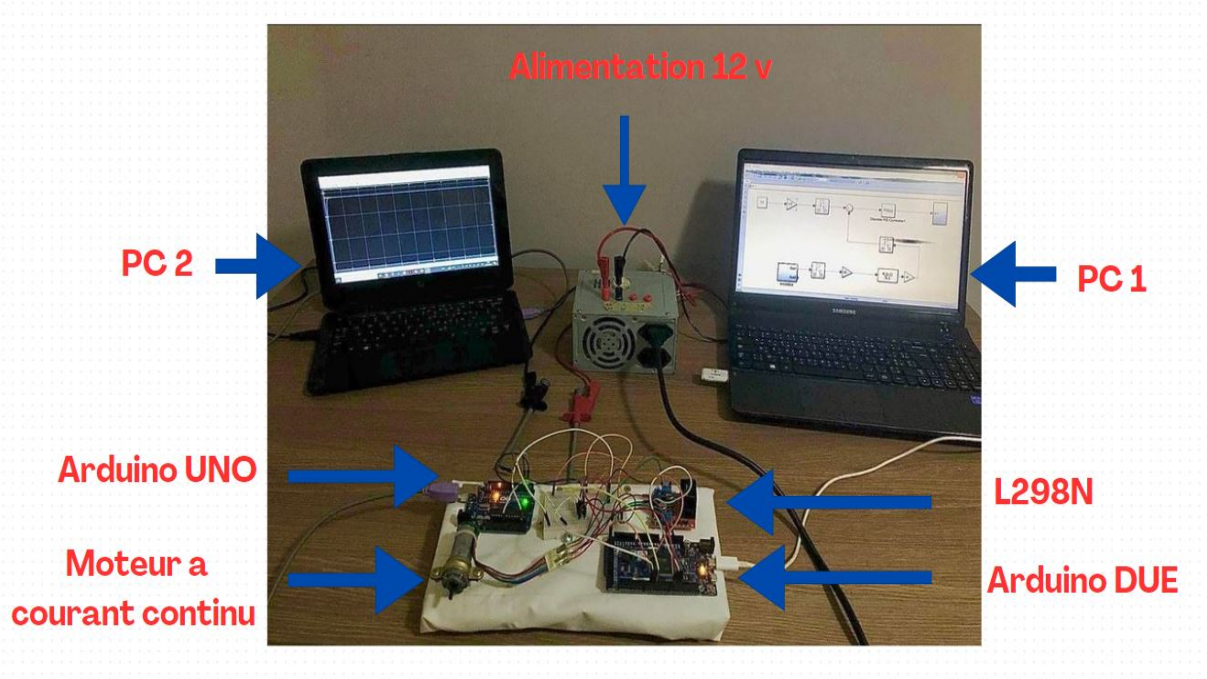


FIGURE 3.16 – Dispositif expérimental réalisé.

Nous avons configuré le fichier Simulink utilisé comme fait avec le fichier de commande pour que le code généré soit adapté à la carte Arduino Due, et que le solveur soit discret "discrete (no continuous states)" avec un pas fixe "Fixed-step".

Après avoir configuré les fichiers de commande dans MATLAB, on appuie sur le bouton "Deploy to Hardware" pour implémenter le programme sur l'Arduino Due connectée au PC1. Cette carte prend en charge les calculs et l'exécution du programme.

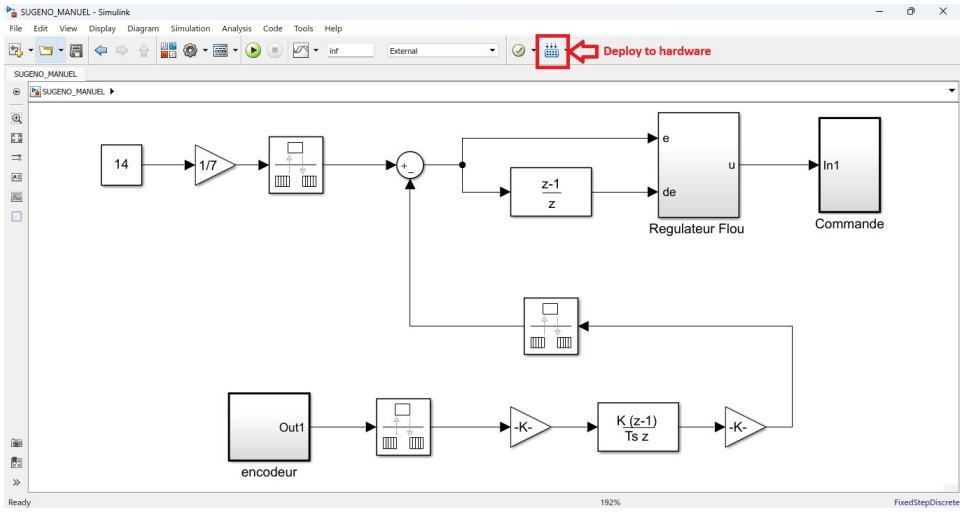


FIGURE 3.17 – Implémentation via "Deploy to Hardware".

Pendant ce temps, l'autre carte connectée au PC 2 lit les résultats des calculs effectués par l'Arduino Due. Les données collectées par l'Arduino Uno en mode externe sont ensuite envoyées à MATLAB, où elles sont visualisées sous forme de graphes pour analyser les résultats en temps réel à l'aide du bloc "Scope" ou les sauvegarder en utilisant "To Workspace".

Nous utilisons un programme Simulink pour l'Arduino Uno qui lira continuellement les valeurs provenant de l'Arduino Due. À chaque itération de cette boucle, l'Arduino Uno récupère une nouvelle valeur transmise par l'Arduino Due. Une fois la valeur récupérée, elle est immédiatement transmise à l'espace de travail de MATLAB.

L'image suivante illustre le programme Simulink utilisé pour lire les valeurs transmises :

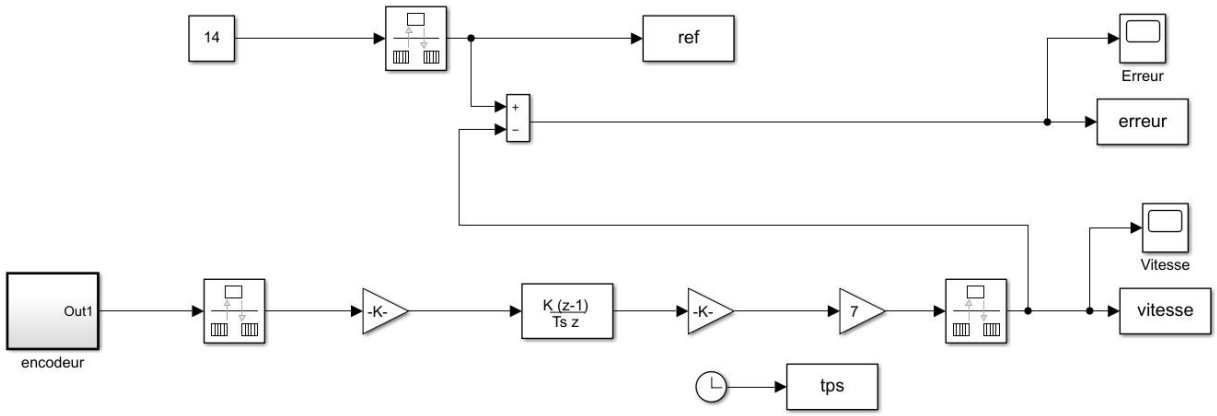


FIGURE 3.18 – Programme Simulink utilisé pour l'acquisition des données de mesure.

3.4 Conclusion

Dans ce chapitre, nous avons commencé par identifier notre système à l'aide de l'outil "System Identification Toolbox".

Ensuite, nous avons développé et testé un régulateur flou de type Takagi-Sugeno pour la régulation de vitesse d'une machine à courant continu.

La conception s'est déroulé en quatre étapes essentielles : la fuzzification, l'établissement des règles floues, l'inférence et la défuzzification.

En parallèle, nous avons aussi conçu un régulateur PID pour comparer les résultats.

Chapitre 4

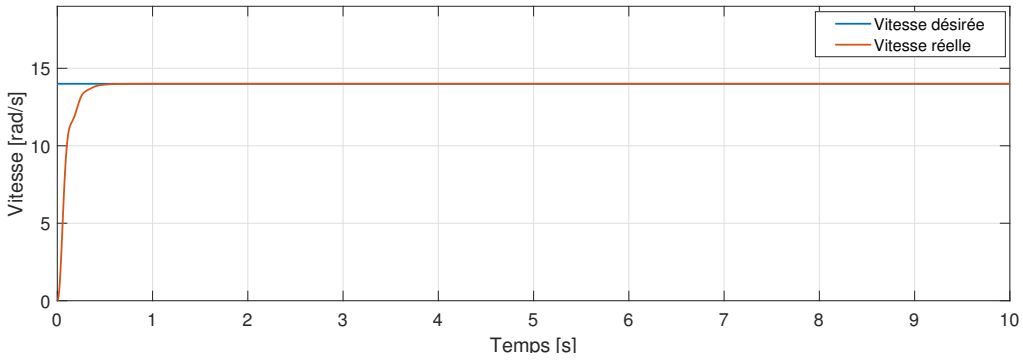
Résultats et discussions

4.1 Introduction

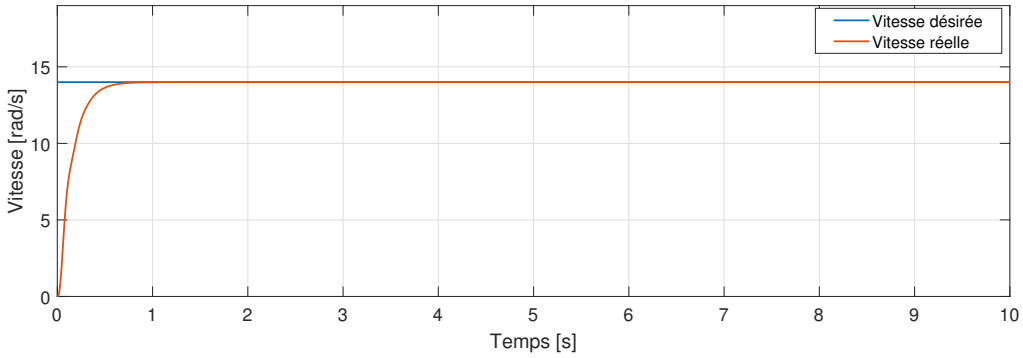
Dans ce chapitre, nous allons analyser les résultats de simulation et expérimentaux obtenus par les deux types de régulateurs : PID (Proportionnel, Intégral, Dérivé) et le régulateur flou de type Sugeno lors de leurs application pour la commande en vitesse d'un moteur à courant continu. Pour pouvoir faire cela, nous avons visualisé les courbes de la vitesse, l'erreur de vitesse ainsi que la tension de commande dans les différents cas.

Nous nous baserons sur ces courbes pour évaluer la rapidité, la stabilité et l'efficacité de ces deux régulateurs.

4.2 Résultats de simulation



(a)

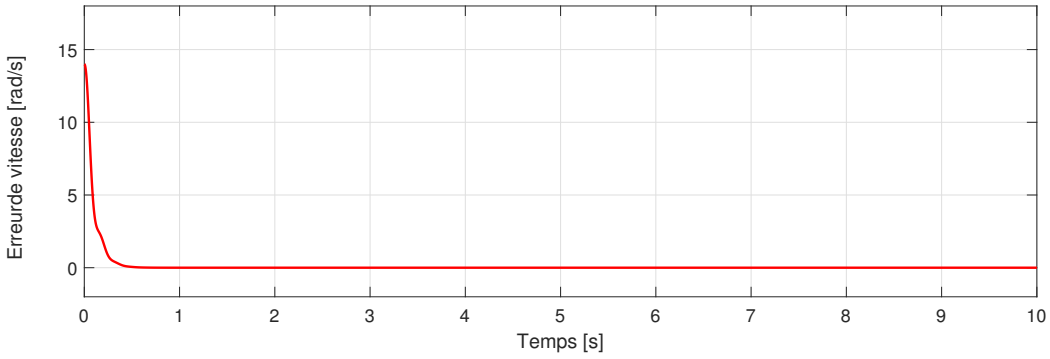


(b)

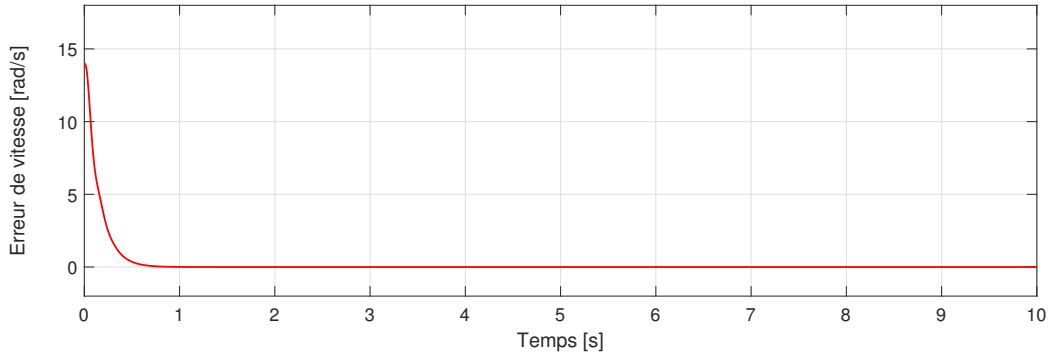
FIGURE 4.1 – Courbes des vitesses obtenues en simulation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

En simulation, les régulateurs Takagi-Sugeno et PID réagissent de manière presque identique, seul le temps de réponse du régulateur flou est légèrement plus long.

En leur donnant la même référence de $14 rad/s$, la vitesse réelle est atteinte avec le Sugeno et le PID en $0,75s$ et $0.5s$ respectivement, sans observer ni dépassement ni retard.



(a)

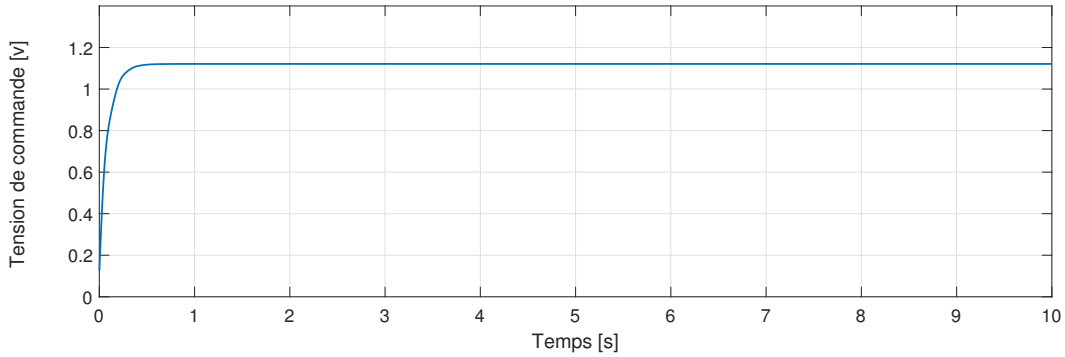


(b)

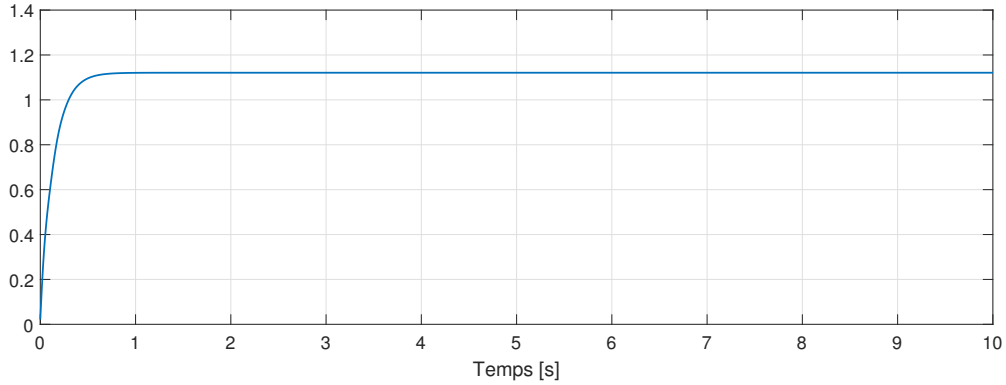
FIGURE 4.2 – Courbes des erreurs de vitesse obtenues en simulation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

En simulation, les régulateurs T-S et PID ont produit les mêmes résultats. L'erreur était initialement de $14 rad/s$, puis s'est annulée à $0,75s$ pour le PID et $0.5s$ pour le T-S puis reste nulle pendant toute la simulation.

On en déduit que les deux régulateurs sont tout aussi efficaces pour atteindre et maintenir la vitesse désirée de $14 rad/s$ sans dépassement ni retard.



(a)

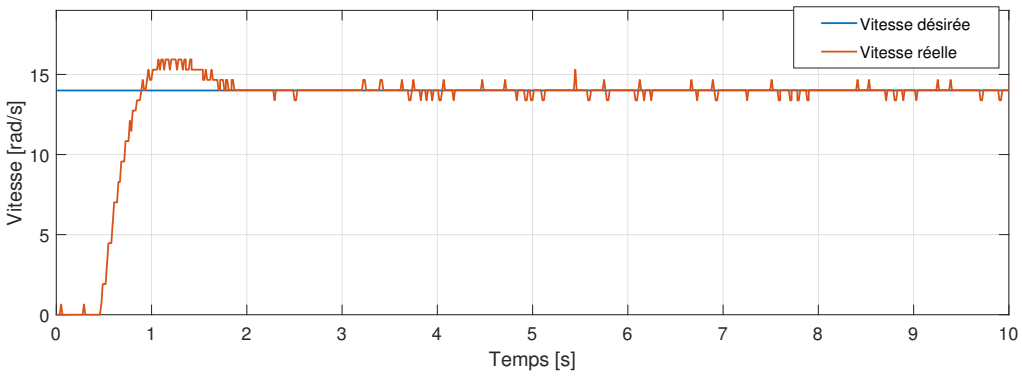


(b)

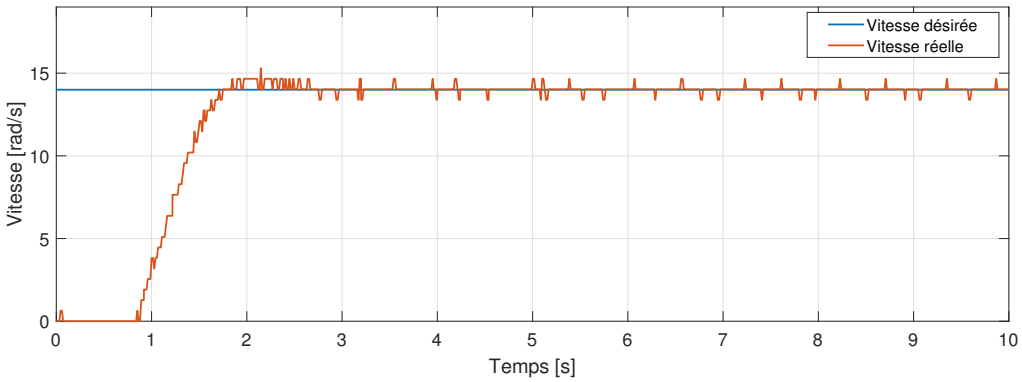
FIGURE 4.3 – Courbes des tensions de commande injectées en simulation par le régulateur : (a) PID, (b) Takagi-Sugeno.

Tout comme pour l'erreur et la vitesse, la tension de commande est très légèrement différente. Celle du régulateur PID augmente plus vite atteignant une valeur crête de $1,1V$ qui est ensuite maintenue. Cette concordance indique que les deux régulateurs sont tout aussi efficaces dans cette situation, assurant une réponse rapide et stable sans oscillations ou dépassements au niveau de la vitesse.

4.3 Résultats de commande en mode externe



(a)



(b)

FIGURE 4.4 – Courbes des vitesses obtenues en mode externe avec le régulateur : (a) PID, (b) Takagi-Sugeno

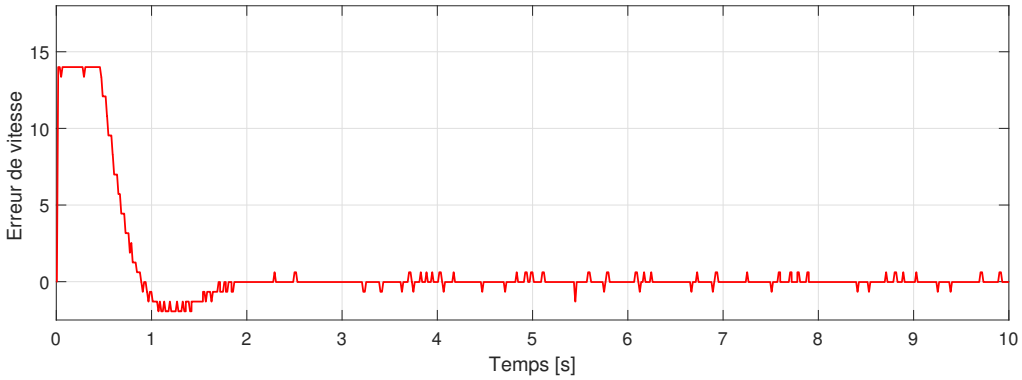
Lors de la commande du moteur avec une référence de vitesse fixée à $14 rad/s$, les deux régulateurs ont été comparés.

Le régulateur PID a démontré une réactivité élevée, atteignant rapidement la vitesse désirée. Cependant, son action a entraîné un dépassement de $2 rad/s$, avant de se stabiliser à la vitesse cible $2s$ après.

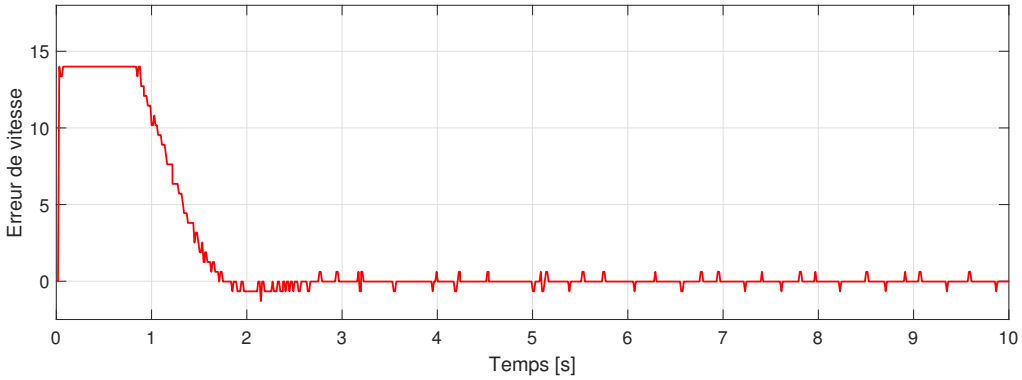
En revanche, le régulateur T-S a présenté un léger retard d'une seconde, bien qu'il n'ait pas montré de dépassement, Sa réponse progressive a abouti à la stabilisation de la vitesse désirée également $2s$ après.

On peut remarquer que ces résultats sont différents de ceux obtenus en simulation, les différences entre les réponses des deux régulateurs sont plus prononcées.

Ces observations soulignent que lors du choix du régulateur pour une application donnée, la sélection se fait selon le critère le plus important, à savoir la rapidité ou la précision.



(a)



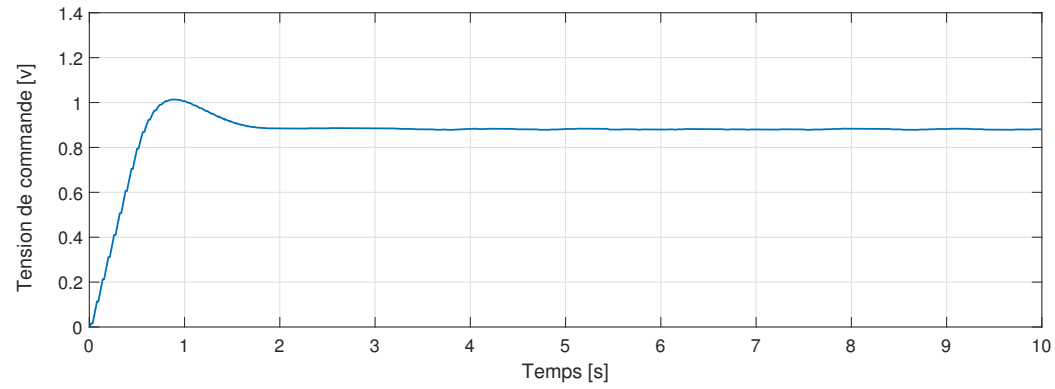
(b)

FIGURE 4.5 – Courbes des erreurs de vitesse obtenues en mode externe avec le régulateur : (a) PID, (b) Takagi-Sugeno.

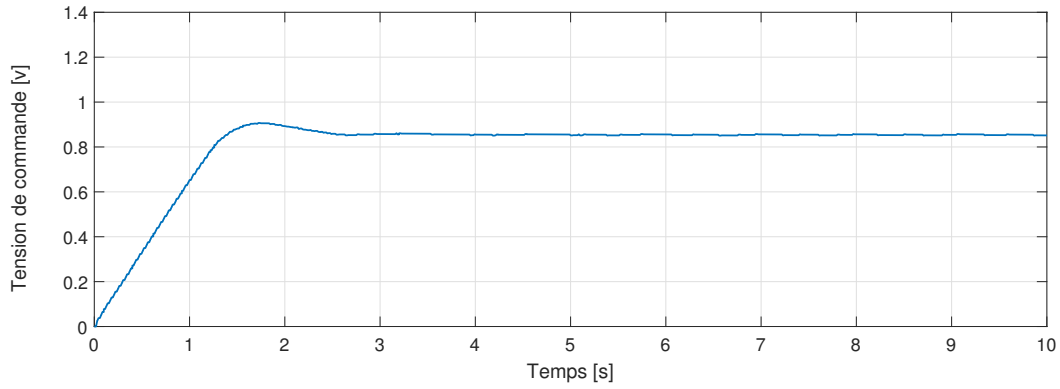
Pour le régulateur PID, à $t = 0s$, l'erreur de vitesse est de $14rad/s$. Il y a un léger retard jusqu'à $0,25s$ avant que le régulateur ne commence à agir. À $1,25s$, le dépassement de la vitesse a provoqué une erreur négative atteignant $-1,4rad/s$. Finalement, à $2s$, l'erreur se stabilise à $0rad/s$.

Quant au régulateur T-S, au premier instant, l'erreur de vitesse est également de $14rad/s$. Un retard d'une seconde est observé avant que le régulateur ait un effet. À $t = 1,9s$, l'erreur se stabilise à $0rad/s$.

Ces observations montrent que les deux régulateurs finissent par stabiliser la vitesse désirée après un certain temps, malgré des différences dans leur comportement initial et leur temps de réponse.



(a)



(b)

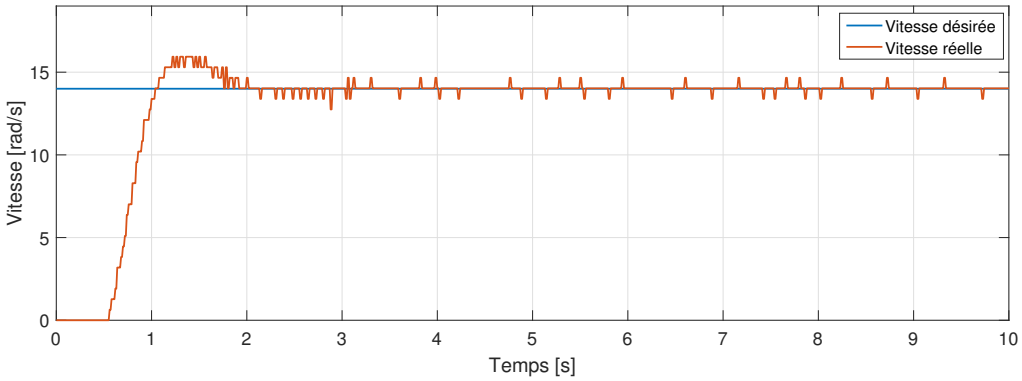
FIGURE 4.6 – Courbes des tension de commande injectées en mode externe par le régulateur : (a) PID, (b) Takagi-Sugeno.

Pour la tension de commande, nous pouvons remarquer qu'elle commence à augmenter dès le premier instant dans les deux cas malgré la différence du temps de démarrage observée sur le graphe de la vitesse.

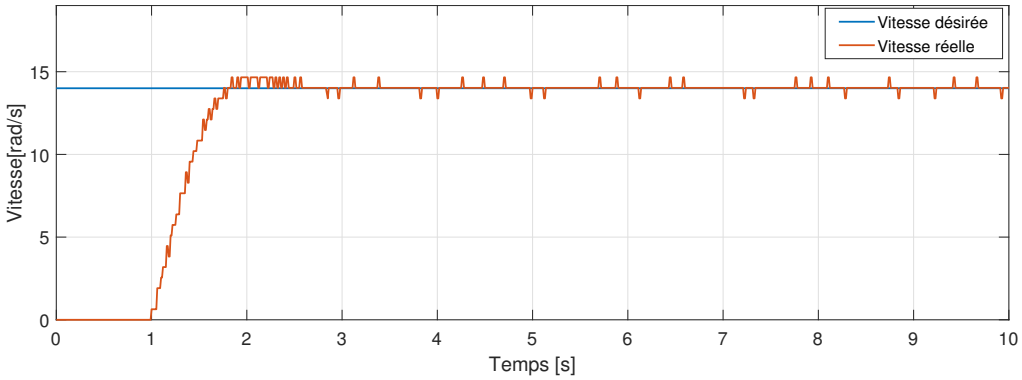
Le régulateur PID montre une stabilisation à $u = 0,9V$ après $2s$. Avant cette stabilisation, un dépassement de $0,1V$ est observé, atteignant un pic de $1V$ à $t = 1s$.

En revanche, pour le régulateur T-S, la stabilisation se produit à partir de $t = 3s$ à $0,83V$. Avant celle-ci, un dépassement de $0,07V$ est constaté,en atteignant une valeur crête de $0,9V$ à $2s$.

4.4 Résultats d'implémentation



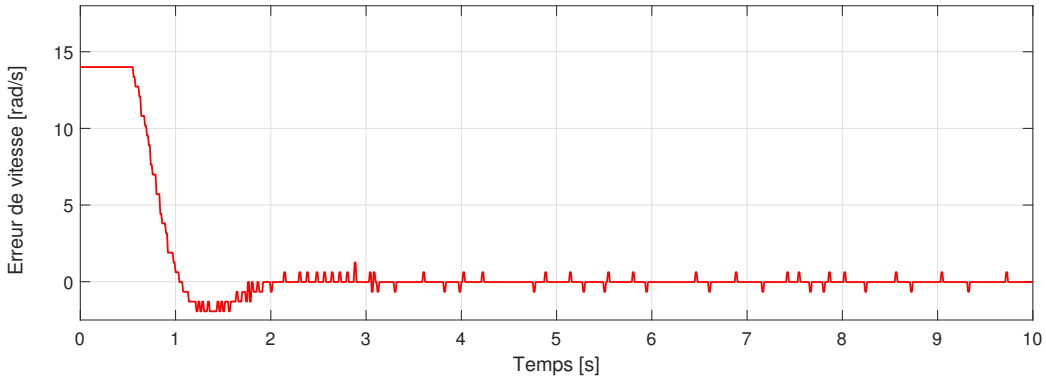
(a)



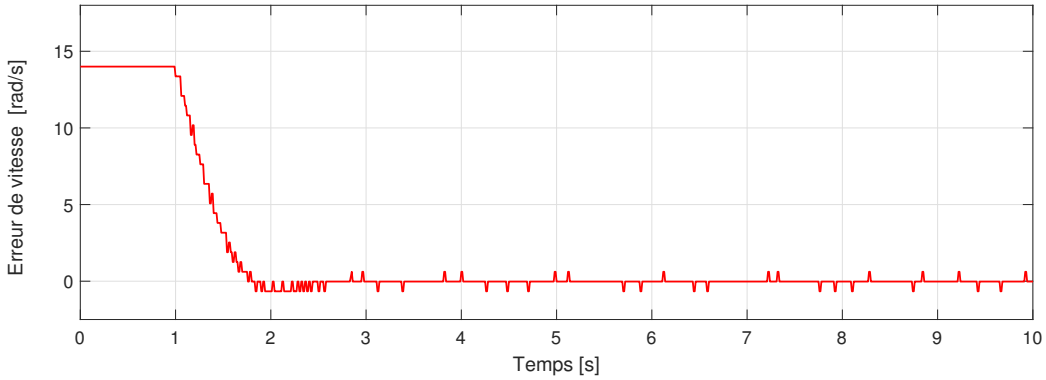
(b)

FIGURE 4.7 – Courbes des vitesses obtenues en implémentation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

Pour la vitesse obtenue en implémentation, elle est quasiment identique à celle du mode externe, ce qui prouve que le régulateur que nous avons programmé a un comportement identique au bloc "Fuzzy Logic Controller" de MATLAB.



(a)



(b)

FIGURE 4.8 – Courbes des erreurs de vitesse obtenues en implémentation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

L'erreur est également presque identique à celle obtenue en mode externe étant donné que les régulateurs sont les mêmes.

4.5 Rejet de perturbations

Afin de tester la robustesse des contrôleurs proposés, nous avons inséré à l'axe du moteur une charge à $t \simeq 5s$.

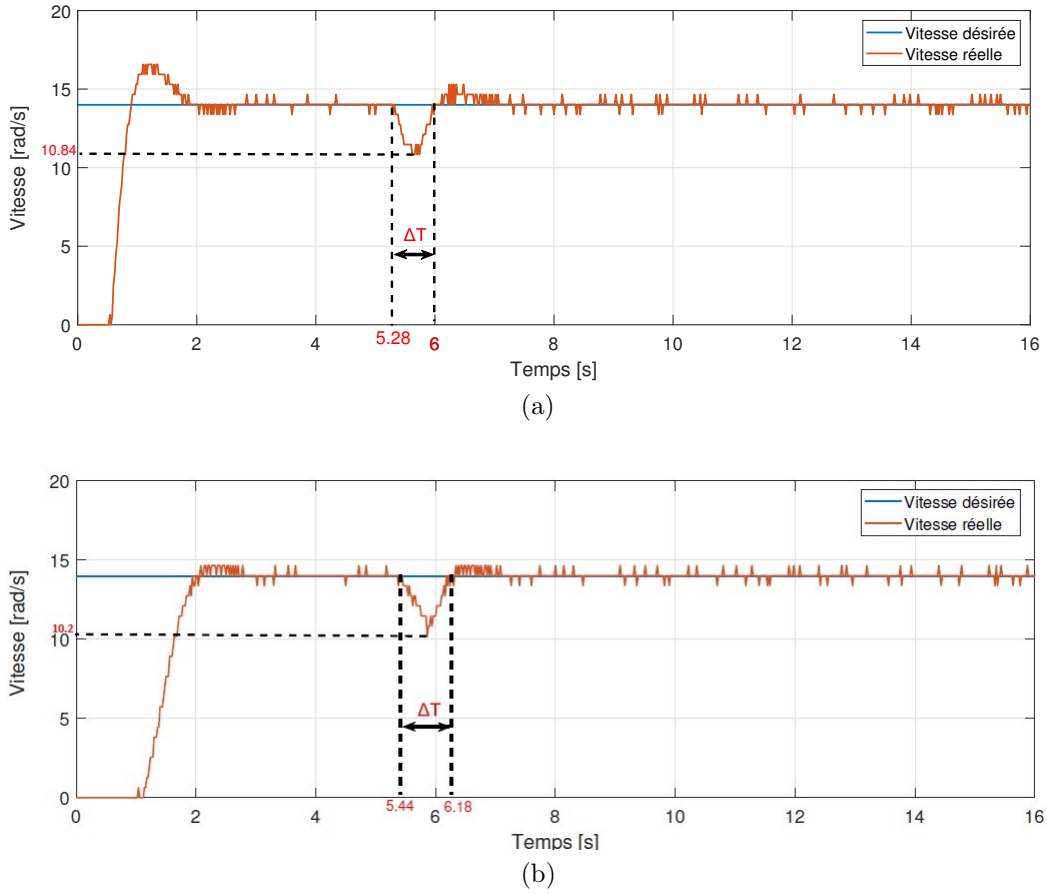
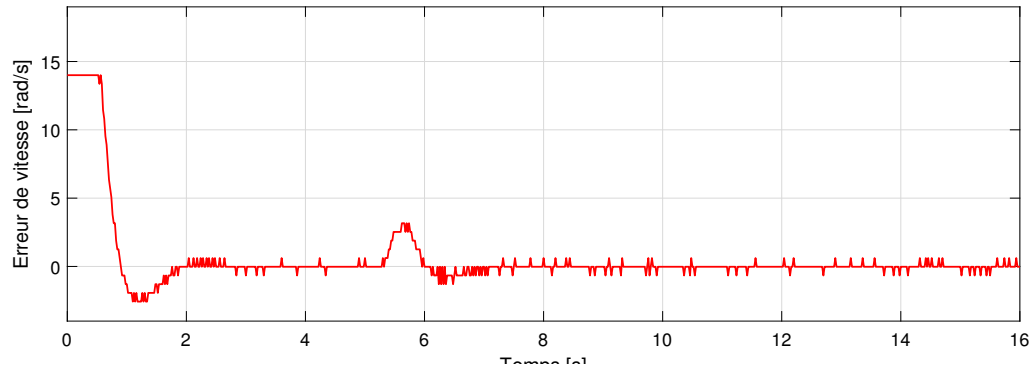
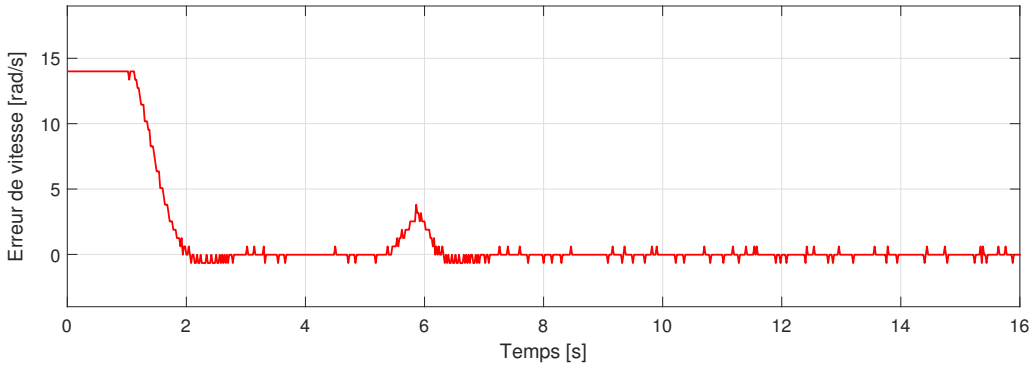


FIGURE 4.9 – Courbes des vitesses obtenues en présence de perturbation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

Comme nous pouvons le constater dans la Figure 4.9, un ralentissement du moteur a été observé dès l'insertion de la charge. La vitesse a baissé à 10.84 rad/s et 10.20rad/s pour le régulateur PID et Takagi-Sugeno respectivement, elle est ensuite revenue à la vitesse désirée. Il y'a une différence négligeable au niveau de intervalle de temps où la vitesse a été perturbée. Nous avons relevé un intervalle de $\triangle t = 0.72s$ et $\triangle t = 0.74s$ pour les systèmes utilisant le contrôleur : PID et T-S respectivement. Cependant, en reprenant la vitesse désirée, le régulateur PID provoque un dépassement important comparé au régulateur T-S.



(a)



(b)

FIGURE 4.10 – Courbes des erreurs de vitesse obtenues en présence de perturbation avec le régulateur : (a) PID, (b) Takagi-Sugeno.

Pour le régulateur PID, dès l'insertion de la charge, l'erreur de vitesse augmente jusqu'à atteindre $3.16 rad/s$. Il réagit pour l'annuler, elle diminue jusqu'à $-1.29 rad/s$ puis s'annule et reste constante.

Pour le régulateur T-S, on observe un différent comportement, l'erreur augmente dès l'insertion de la charge pour atteindre $3.8 rad/s$ et diminue ensuite jusqu'à atteindre $-0.66 rad/s$ puis s'annule et reste stable.

De manière globale, nous constatons que les deux régulateurs ont permis l'obtention de résultats de commande satisfaisants. Le régulateur PID se distingue par sa rapidité mais aussi par un dépassement relativement élevé alors que le régulateur T-S donne des courbes plus lisses tout en gardant une précision élevée ainsi qu'un rejet de perturbations aussi rapide que le contrôleur PID.

4.6 Conclusion

En conclusion, cette étude comparative a permis de mettre en évidence les caractéristiques distinctes de ces deux approches de contrôle. Le régulateur PID offre une réactivité élevée au prix d'un dépassement, tandis que le régulateur flou garantit une stabilité sans dépassement au détriment du temps de réponse. Le choix entre ces deux régulateurs dépendra donc des exigences et contraintes spécifiques de chaque application et des compromis acceptables entre réactivité et stabilité, particulièrement la présence ou non du modèle du système.

L'implémentation pratique sur des cartes Arduino a montré l'efficacité des contrôleurs proposés, de la méthodologie de conception et d'implémentation adoptée.

Conclusion générale

Le travail présenté dans le cadre de ce mémoire a pour objectif d'aborder les techniques de l'intelligence artificielle, plus particulièrement les techniques floues appropriées à la commande des moteurs. Celles-ci sont utilisées pour surmonter les contraintes rencontrées lors de l'utilisation des techniques de commande classiques afin d'obtenir les performances voulues.

Dans ce travail, nous avons proposé un contrôleur flou de type Takagi-Sugeno pour commander un moteur à courant continu en utilisant une carte Arduino Due choisie pour sa fréquence élevée, son utilisation a permis d'obtenir un système efficace qui atteint et maintiens la vitesse exigée sans dépassement et s'adapte également en cas de présence d'une charge.

Ce mémoire a été divisé en quatre parties.

Dans la première partie, nous avons présenté les éléments de la logique floue nécessaires à la conception de notre régulateur ainsi que les techniques de régulation floues les plus communes.

Dans la deuxième partie, nous avons présenté les différents modèles de cartes Arduino connues, particulièrement le modèle Due utilisé dans ce travail. Ainsi que les différents logiciels et outils permettant sa programmation.

Dans la troisième partie, nous avons commencé par obtenir un modèle approximatif du moteur pour la simulation. Ensuite, nous avons proposé un régulateur Takagi-Sugeno visant à résoudre la problématique en question, nous l'avons programmé dans l'environnement Simulink puis implémenté. Enfin, nous avons appliqué la même procédure pour la conception d'un régulateur PID dans le but de comparer leurs performances.

La dernière partie de ce travail a été consacrée à l'étude des résultats obtenus et la comparaison des deux méthodes de contrôle. Ces résultats nous ont permis de tirer des conclusions pertinentes à propos du régulateur proposé dans la partie précédente.

Nous constatons que l'utilisation de régulateurs basés sur l'intelligence artificielle s'avère efficace et annonce même un avenir prometteur vu les fenêtres d'évolution possibles. Comme attendu, le régulateur a permis la commande d'un moteur a courant continu sans modèle exact avec succès, ce qui permet un gain de temps considérable étant donné

qu'une fois le régulateur conçu, il ne reste plus qu'à l'adapter à son environnement d'utilisation. En cas de commande d'un autre moteur à courant continu, il suffit de changer les gains de normalisation. Nous pouvons donc dire que le régulateur Takagi-Sugeno convient pour un large éventail d'utilisations et présente de nombreux avantages comme l'absence de dépassement et l'obtention de courbes lisses sans variation brusque ou encore sa capacité à effectuer une régulation là où d'autres régulateurs auraient du mal à fonctionner correctement. De plus, ce dernier réagit de manière efficace aux perturbations, ce qui le rend utilisable dans un contexte où différentes charges sont appliquées au moteur.

Cependant, il existe tout de même des cas où l'utilisation d'un autre type de régulateur s'avèrerait plus judicieux. Comme nous avons pu le constater dans le quatrième chapitre, le régime transitoire dure plus de temps qu'avec un régulateur PID. Suivant le domaine d'utilisation du moteur, ce délais pourrait être un inconvénient, le supprimer nécessiterait des essais pour chacun des moteurs sur lequel il serait utilisé. Des innovations futures pourraient libérer le développeur de ces tâches d'adaptation par un programme effectuant un étalonnage automatique lors du branchement d'un nouveau moteur, l'implémentation d'un régulateur flou sur une carte FPGA pour la commande d'un M.C.C ouvrirait aussi les portes à des innovations plus sophistiquées étant donné la puissance de la plateforme FPGA.

Bibliographie

[1] Arduino : Utilisation et fonctionnement. *arduino-france.com.*

[2] Benyamine Allouche. *Modélisation et commande des robots : nouvelles approches basées sur les modèles Takagi-Sugeno.* PhD thesis, Université de Valenciennes et du Hainaut-Cambresis Automatique / Robotique NNT : 2016VALE0021 tel-01469234, 2016.

[3] Lotfi Baghli. *Contribution à la commande de la machine asynchrone, utilisation de la logique floue, des réseaux de neurones et des algorithmes génétiques.* PhD thesis, Université Henri Poincaré - Nancy 1, 1999.

[4] Erik Bartmann. *Le grand livre d'Arduino.* EYROLLES, 2013.

[5] Dimiter Driankov, Hans Hellendoorn, and Michael Reinfrank. *An Introduction to Fuzzy Control.* Springer Berlin Heidelberg, 1996.

[6] Tunisie Institut Superieur des Etudes Technologiques (ISET) de Bizerte.

[7] Hocine Khati. *Commande d'une Architecture de Téléopération par la Carte FPGA.* PhD thesis, Université Mouloud Mammeri de Tizi Ouzou, 2020.

[8] MathWorks. Arduino programming with matlab and simulink. *Mathworks.com.*

[9] Kamyar Mehran. Takagi-sugeno fuzzy modeling for process control. *Industrial Automation, Robotics and Artificial Intelligence (EEE8005) School of Electrical, Electronic and Computer Engineering*, 2008.

[10] Patricia Melin, Oscar Castillo, Janusz Kacprzyk, Marek Reformat, and William Melek, editors. *Fuzzy Logic in Intelligent System Design.* Springer eBook Collection. Springer, Cham, 2018.

[11] Hung T. Nguyen, Nadipuram R. Prasad, Carol L. Walker, and Elbert A. Walker. A first course in fuzzy and neural control.

[12] Hung T. Nguyen, Carol Walker, and Elbert A. Walker. *A First Course in Fuzzy Logic.* CRC Press, 2018.

[13] John Nussey. *Arduino pour les Nuls poche.* First Interactive, 2017.

[14] Victor Hugo Grisales Palacio. *Modélisation et commande floues de type Takagi-Sugeno appliquées àun bioprocédé de traitement des eaux usées.* PhD thesis, Auto-

matique / Robotique Université Paul Sabatier -Toulouse III Français. NNT : tel-00136382, 2007.

[15] Jean-Christophe Quetin. *Arduino : Apprivoisez l'électronique et le codage pour donner vie à vos projets.* ENI, 2021.

[16] Claude Rosental. Histoire de la logique floue une approche sociologique des pratiques de démonstration. *Revue de Synthèse*, 119(4) :575–602, 1998.

[17] Nazmul Siddique. Intelligent control, 2014.

[18] M. Takagi, T.and Sugeno. *Fuzzy identification of systems and its applications to modeling and control. Systems, Man and Cybernetics,IEEE Transactions on (1) :116–132.* IEEE, 1985.

[19] Nakoula Y. *Apprentissage des Modèles linguistiques flous, par jeu de règles pondérées.* PhD thesis, Université de Savoie, France, 1997.

Annexe

Circuit L298N

Le module L298N est préactionneur composé de deux ponts-H et permet la commande en vitesse et en sens de deux moteurs à courant continu simultanément. Le module prend en charge les moteurs à courant continu fonctionnant sous des tensions allant de 5 à 35V avec un courant maximal de 2A. Il est important de noter que ce module provoque une chute de tension de 2V. Par conséquent, lorsque on commande notre moteur de 12V, seul 10V pourront être fournies, ce qui signifie qu'il ne sera pas possible d'atteindre sa vitesse maximale.

Les pins OUT1 et OUT2 sont connectés à l'alimentation du moteur. Le module est alimenté par les pins 12V et GND. Les pins IN1 et IN2 reçoivent la commande du sens du moteur et ENA la PWM provenant de l'Arduino.

La Figure 4.11 illustre le schéma du circuit électronique du L298N.

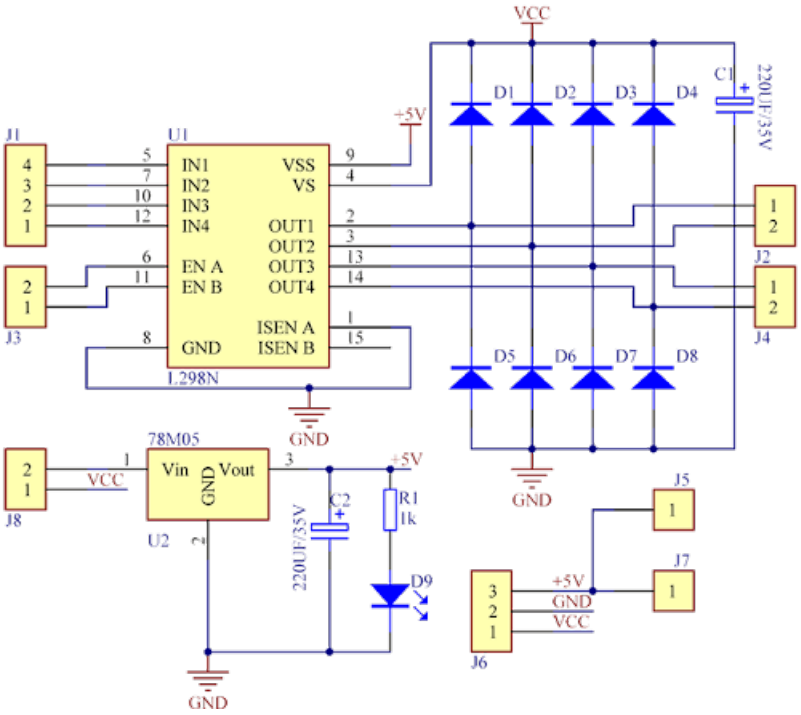


FIGURE 4.11 – Schéma du circuit électronique du L298N.

Caractéristiques

- Tension de fonctionnement 4V à 46V.
- Courant total (DC) jusqu'à 4 A.
- Basse tension de saturation.
- Protection contre la surchauffe.
- Régulateur embarqué 5V à faible chute de tension, accessible par l'utilisateur.
- Diodes de protection.
- Alimentation logique : 5V.
- Puissance maximale : 25W.
- Deux voyants de direction du moteur.
- Température de stockage : -25 à +135.

La Figure 4.12 présente le module L298N.

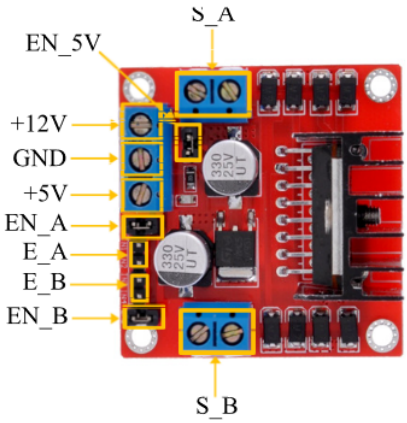


FIGURE 4.12 – Module L298N.

Le Tableau 4.1 présente les fonctions des pins du circuit L298N.

Nom de la pin	Description
S_A	Moteur A
S_B	Moteur B
$+12V$	Entrée d'alimentation $12V$
GND	Terre
$+5V$	Sortie $5V$
E_A	2 Entrées logiques pour le moteur A
E_B	2 Entrées logiques pour le moteur B
EN_A	Contrôle du moteur A
EN_B	Contrôle du moteur B
EN_$5V$	Activation $5V$

TABLE 4.1 – Fonctions des pins du L298N.