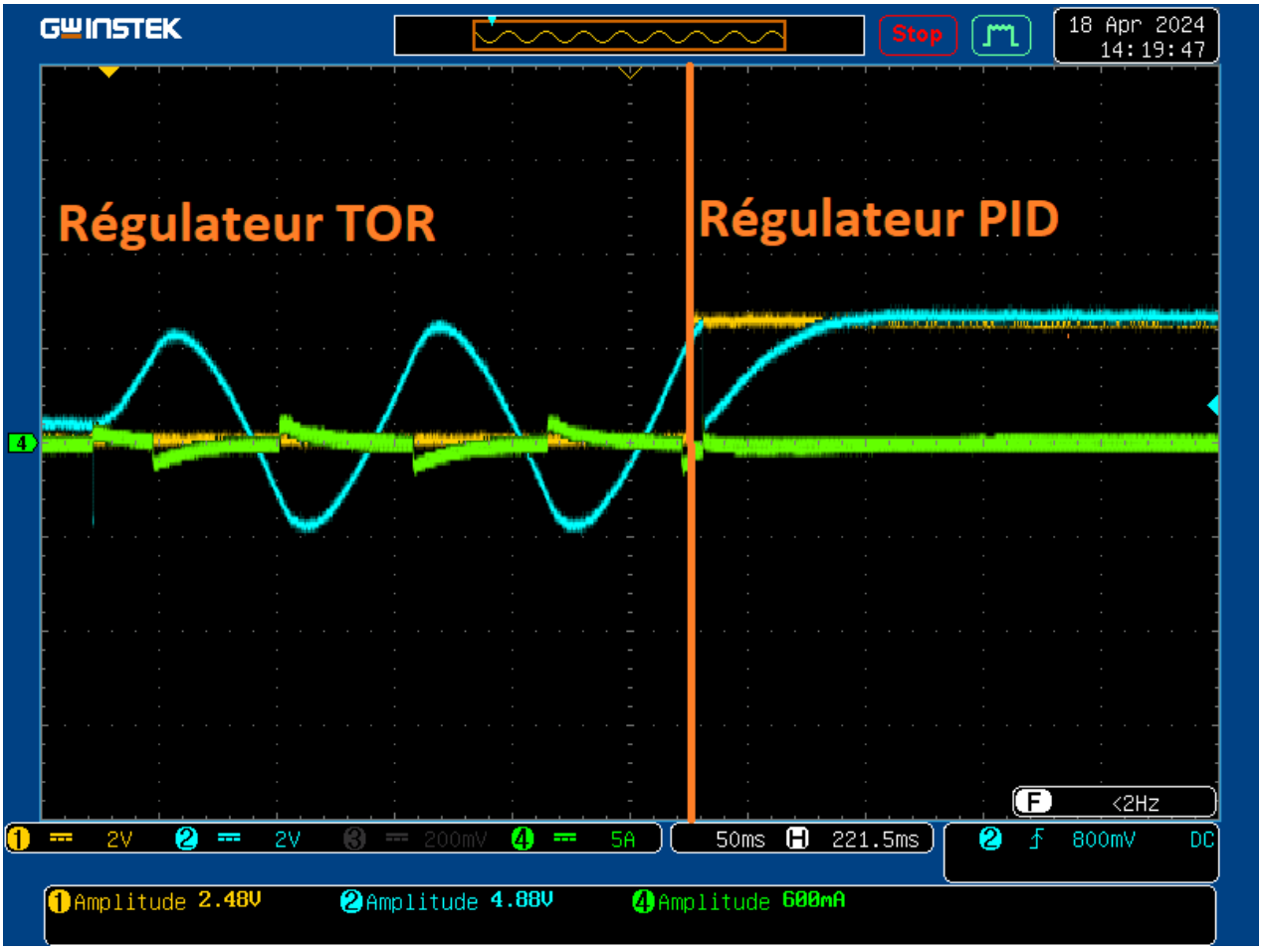
Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Projet de semestre 6

Auto-ajustage de régulateur PID (ARP)

Maillard Dimitri



Superviseur: Moncef Justin Lalou
Durée du projet: 20.02.2024 – 16.05.2024
Filière: Génie électrique, HEIA-FR

Abstract/Résumé

Ce rapport présente le projet "Auto-ajustage de régulateur PID" visant à explorer et appliquer les principales méthodes de réglage pour créer un système auto-régulé fonctionnel. L'étude se concentre sur l'utilisation du régulateur PID, largement répandu dans le domaine de la régulation automatique. L'objectif final est de développer un système capable de s'adapter aux variations de son environnement et de maintenir des performances optimales.

La première maquette présente des inconvénients logiciels qui permettent de réaliser chacune des applications séparément, mais pas simultanément. En effet l'application auto-régulée est fastidieuse à réaliser avec des blocs fonctionnels (langage que prévoit Saia pour son automate). Il n'a pas été réalisé dans ce projet.

Les résultats de la deuxième expérimentation montrent que le système auto-régulé répond de manière efficace aux sauts de consigne et parvient à réguler le système rapidement et avec précision. La régulation se fait en 60ms sans prendre en compte la phase d'oscillations (250ms). Des améliorations et optimisations peuvent être réalisées dans le code.

Bien que la méthode Åström et Hägglund offre un bon point de départ pour le réglage des paramètres, elle ne constitue pas une solution universelle. Cependant, elle représente un point de départ dans la recherche des coefficients, offrant une méthode d'auto-ajustage autonome et adaptable à différents systèmes.

This report presents the "PID controller self-tuning" project, which aims to explore and apply the main tuning methods to create a functional self-tuning system. The study focuses on the use of the PID controller, which is widely used in the field of automatic control. The ultimate aim is to develop a system capable of adapting to variations in its environment and maintaining optimum performance.

The first model has software drawbacks that allow each of the applications to be carried out separately, but not simultaneously. In fact, the self-regulating application is tedious to create using function blocks (the language that Saia provides for its PLC). It was not implemented in this project.

The results of the second experiment show that the self-regulating system responds effectively to setpoint jumps and manages to regulate the system quickly and accurately. Regulation is achieved in 60ms without taking into account the oscillation phase (250ms). Improvements and optimisations can be made to the code.

Although the Åström et Hägglund method provides a good starting point for parameter tuning, it is not a universal solution. However, it does provide a starting point in the search for coefficients, offering a self-tuning method that can be adapted to different systems.

Table des matières

Liste des figures

Liste des tableaux

Liste des codes

Abréviations

Abréviation	Définition
HEIA-FR	Haute école d'ingénierie et d'architecture de Fribourg
AH	Karl Johan Åström et Tore Hägglund ; Deux Suédois publiant en 1984 la méthode étudiée qui porte leurs noms
ZN	méthode de Ziegler & Nichols
PID	Régulateur comprenant le terme proportionnel,intégral et dérivé
PI	Régulateur comprenant le terme proportionnel et intégral
PD	Régulateur comprenant le terme proportionnel et dérivé
API	Automate programmable industriel
Kp	Gain proportionnel
Ki	Gain intégrateur
Kd	Gain dérivateur
PWM	Pulse Width Modulation
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
TOR	Tout ou rien (Commande de régulateur)
T_c	Période critique
K_c	Gain critique
AC	Alternativ Current
DC	Direct Current
B	Gain pour al méthode AH qui est multiplié avec Kp
Y_{rp}	Commande proportionnelle numérique
Y_{ri}	Commande intégrale numérique
Y_{rd}	Commande dérivée numérique
M_S	Marge de sensibilité
LCD	Liquid crystal display

1. Introduction

Dans un monde où les systèmes automatisés jouent un rôle de plus en plus prépondérant, la nécessité de concevoir et de réguler efficacement ces systèmes prend une importance cruciale. C'est dans ce contexte que s'inscrit le projet intitulé "Auto-ajustage de régulateur PID". Ce rapport se concentre sur l'étude et la mise en œuvre du régulateur PID (Proportionnel, Intégral, Dérivé), une technique largement utilisée dans le domaine de la régulation automatique.

L'objectif de ce projet est de créer un système auto-régulé fonctionnel, capable de s'adapter de manière autonome aux variations de son environnement tout en maintenant des performances optimales dans des conditions variables. Ainsi, ce rapport détaillera les différentes étapes de notre démarche, de l'étude théorique des méthodes de réglage à leur application pratique, en mettant en lumière les résultats obtenus au cours des expérimentations.

1.1. Le projet

Pour répondre à ce problème, il s'agit de développer un outil simple pour l'ajustement automatique des régulateurs PID. Les différents tests et essais sont effectués avec un API Saia PCD2 programmé à l'aide de logiciel PG5 et son langage FUPLA. En deuxième partie, ils seront faits sur un microprocesseur DSP TMS320F28335. L'environnement de développement est fait avec Code Composer Studio en "C".

1.2. Objectif

L'objectif de ce projet est, d'ici le 16 mai 2024, de réaliser un régulateur avec la méthode d'autoajustement de Åström et Hägglund en fonction de l'amortissement voulu.

1.3. Organisation du rapport

Le rapport est structuré en plusieurs sections distinctes.

Il débute par une introduction qui met en lumière le contexte général du projet, l'objectif, la planification ainsi que le cahier des charges.

Ensuite, la présentation de diverses méthodes de réglage sont décrites au point ajustage des paramètres d'un régulateur PID.

La suite du rapport est consacrée à la description et à l'analyse de deux expérimentations. La description de chacune est faite au début. Divers schémas et figures illustrent le déroulement de la régulation.

Dans la section 'Synthèse', les principaux résultats des expérimentations sont récapitulés et leur signification est discutée dans le contexte plus large de l'automatique et du contrôle des systèmes. Des enseignements sont tirés de ce travail et des perspectives pour des développements futurs sont envisagées.

Enfin, le rapport se conclut en résumant les points clés abordés et en soulignant les défis rencontrés.

Le rapport se termine avec une liste de références bibliographiques utilisées dans le travail, ainsi que des annexes fournissant des informations supplémentaires pour une meilleure compréhension du rapport.

1.4. Cahier des charges

Les débuts de ce projet consistent à étudier les principales méthodes de réglage. La méthode de Åström et Hägglund sera, elle, mise en avant afin de réaliser un système auto régulé. Une fois la théorie bien comprise, à l'aide de modèles de laboratoire, des tests seront effectués afin de valider

la méthode. Une première expérience sera réalisée à l'aide de l'automate Saia PCD2.M5540 et le logiciel PG5. Le "code" est écrit grâce à des blocs fonctionnels. Une deuxième maquette est à étudier. Elle comporte un moteur et est régulée par une carte Delfino dont le code est écrit en "C". Un rapport présentant les méthodes, les expérimentations, les descriptions, les tests ainsi que les résultats est écrit.

1.5. Planification du projet

Le projet débute la semaine du lundi 19 février 2024 (notée semaine 1) et se termine la semaine du lundi 20 mai 2024 (notée semaine 13). Le présent rapport est à rendre au plus tard le mercredi 15 mai 2024 à 23h59. Le rendu de la slide TV ainsi que le support Power-Point pour la défense orale est à rendre lundi 20 mai 2024. La défense orale de ce travail se fera le mardi 21 mai 2024. En bleu, le planning estimé, en jaune le planning effectif et en vert la rédaction du rapport. En bleu → Planifié | en vert → Documentation | En jaune → temps effectif | En rouge → Deadline.

Semaines du projet	26.02-01.03 1	04.03-08.03 2	11.03-15.03 3	18.03-22.03 4	25.03-29.03 5	01.04-05-04 6	08.04-12.04 7	15.04-19.04 8	22.04-26.04 9	29.04-03.05 10	06.05-10.05 11	13.05-17.05 12	20.05-24.05 13
Rendu Rapport												15	
Rendu Slide TV et Oral													20
Planification et Objectif													
Cahier des charges													
État de l'art													
Expérimentation 1 (SAIA)													
Programmation PG5													
Expérimentation 2 (Delfino)													
Code régulateur à relais													
Code régulateur AH													
Code Auto-tunning													
Essai Laboratoire													
Comparaison et synthèse													
Rédaction Rapport													
Relecture& correction													

TABLE 1 – Planification du projet

2. Ajustage des paramètres d'un régulateur PID :)

Les régulateurs PIDs sont largement répandus dans l'industrie, avec une présence dépassant les 90%. Malgré des décennies d'expérience, les réglages des paramètres Kp, Ki et Kd ne sont souvent pas optimaux pour les processus à contrôler.

L'histoire des régulateurs remonte loin, dès 1788, avec l'installation d'un contrôleur de vitesse sur une machine à vapeur de James Watt, considéré comme l'un des premiers régulateurs de l'histoire.

Aujourd'hui, diverses méthodes sont utilisées pour régler les régulateurs PID, qu'elles soient basées sur des approches mathématiques telles que le placement des pôles, ou empiriques, comme la méthode de Ziegler et Nichols. Chacune de ces approches présente ses propres avantages et inconvénients. Dans cette section, nous examinerons plusieurs de ces méthodes afin de les comparer. Pour une compréhension approfondie du fonctionnement et des composants d'un régulateur PID, explorons en détail les caractéristiques et le fonctionnement de ce dispositif.[1]

2.1. Description d'un régulateur PID

Le régulateur PID remplit trois fonctions :

- Il fournit un signal de commande $Y_{cm}(t)$ en tenant compte de l'évolution du signal de sortie $Y(t)$ par rapport à la consigne $Y_c(t)$.
- Il élimine l'écart permanent grâce au terme intégrateur.
- Il anticipe les variations de la sortie grâce au terme dérivateur.

Le schéma fonctionnel d'un système ou processus à régler est illustré à la figure 1 . Ce schéma ne présente pas la perturbation qui serait un terme à additionner au système. La description temporelle du régulateur se donne par :

$$Y_{cm}(t) = Kp \cdot \left(e(t) + Ki \int_0^t e(t)dt + Kd \frac{de(t)}{dt} \right) \tag{1}$$

avec l'erreur définie comme :

$$e(t) = Y_c(t) - Y_m(t) \tag{2}$$

Dans le domaine de Laplace, sa fonction de transfert s'écrit :

$$H_r(s) = \frac{Y_{cm}(s)}{E(s)} = kp + Kd \cdot s + \frac{Ki}{s} \tag{3}$$

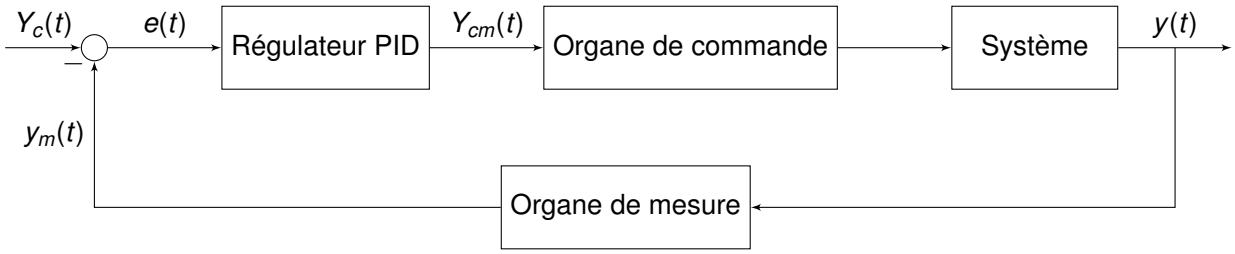


FIGURE 1 – Schéma fonctionnel d'un processus à régler par un PID classique

Où :
Régulateur PID = $H_r(s)$
Organe de commande = $H_{cm}(s)$
Système = $H_s(s)$
Organe de mesure = $H_m(s)$

2.2. Méthode de réglage des régulateurss PID

Afin de répondre au mieux aux exigences de chaque processus à régler, il existe un bon nombre de méthodes, en voici quelques-unes.

- Méthode itérative à trois phases
- Méthode de Ziegler et Nichols
- Métahode de Åström et Hägglund

2.2.1. Méthode itérative à trois phases

Cette méthode empirique est très utilisée du fait de sa simplicité d'exécution. Il s'agit d'effectuer des essais en trois phases observant les effets prévisibes des trois corrections Kp,Ki et Kd. Dans tous les essais, on utilise un saut indiciel comme consigne.

- **Phase 1 :** Commencer par mettre Ki et Kd à 0. puis introduire un gain Kp de faible valeur et l'augmenter pour accélérer le processus réglé tant que les oscillations et les dépassements sont acceptables. Cette valeur de Kp permet de charger correctement l'organe de commande.
- **Phase 2 :** On ajoute la correction Kd en débutant à 0 jusqu'à obtenir un bon amortissement de la réponse $Y_m(t)$. Le réajustement de Kp peut être nécessaire.
- **Phase 3 :** Enfin on ajoute la correction Ki en débutant à 0 afin que $Y_m(t)$ atteigne la consigne rapidement. Si les oscillations reprennent ou ne sont plus acceptables, augmenter Kd ou diminuer Kp.

Si le réglage est cohérent sans l'une ou l'autre correction (Ki ou Kd) alors, nous les laisserons à 0.

2.2.2. Méthode de Ziegler et Nichols

La méthode de Ziegler et Nichols appelée ZN dans ce document, est une approche basée sur leurs expériences. Elle consiste premièrement à n'utiliser qu'un régulateur P, en amenant le gain Kp jusqu'à oscillation permanente du système. A ce stade, on obtient la limite de stabilité. On attribue la valeur de Kp à Kc qui correspond au gain critique. On mesure la période d'oscillation appelée T_c. Grâce aux travaux de John G. Ziegler et Nathaniel B. Nichols le tableau ci-dessous donne les gains des différents régulateurs.[2]
Remarque sur la méthode de ZN. Les essais doivent être réalisés hors saturation ou limite afin que

Régulateur	Kp	Ki	Kd
P	$0.5 \cdot kc$	-	-
PI	$0.45 \cdot Kc$	$1.2 \cdot \frac{Kp}{Tc}$	-
PID	$0.6 \cdot Kc$	$2 \cdot \frac{Kp}{Tc}$	$0.125 \cdot Kp \cdot Tc$

TABLE 2 – Gains méthode ZN

les réponses reflètent le comportement propre du processus. L'expérience montre que les gains Kp proposés sont élevés et conduisent à des dépassements. On peut aisément les diviser par deux pour obtenir une réponse bien amortie. On peut admettre que le réglage ZN est un réglage primaire et que l'ajustage itératif plus fin peut être nécessaire (fine-tunning).

2.2.3. Méthode de Åström et Hägglund

Enfin cette méthode qui nous intéresse pour la suite du projet ressemble à la méthode ZN. Pour la suite du projet, la méthode AH sera utilisée, mais adjointe d'un régulateur à hystérésis ou tout ou rien (TOR). Ceci pour faire de l'auto-ajustage (Auto-tunning). Le schéma fonctionnel montré sur la figure 3. L'idée est de chercher le gain critique Kc à l'aide du régulateur à relais qui entrainera une oscillation de la grandeur réglée $Y_m(t)$. La réponse nous donnera trois paramètres pour calculer

les gains du régulateur. A_r= Amplitude crête du relais, A_m= Amplitude crête de la mesure et Tc la période d'oscillation. À savoir qu'un coefficient "b" est ajouté avant le gain Kp. Les autres paramètres sont calculés à l'aide des équations ci-dessous. [3]

$$K_c = \frac{4 \cdot A_r}{\pi \cdot A_m} \tag{4}$$

$$K_0 = \frac{A_m}{A_r} \tag{5}$$

$$\kappa = \frac{1}{k_0 \cdot K_c} \tag{6}$$

Finalement, en connaissant ceci, les coefficients du régulateur peuvent être donnés par une seule fonction :

$$f(\kappa) = a_0 \cdot e^{(a_1 \cdot \kappa + a_2 \cdot \kappa^2)} \tag{7}$$

Les paramètres a_0, a_1 et a_2 sont donnés par le tableau 3
Rappel : le coefficient Ms "mesure" la sensibilité du réglage par rapport à la perturbation. Lorsque Ms=1.4, le régulateur aura un meilleur amortissement. [4]

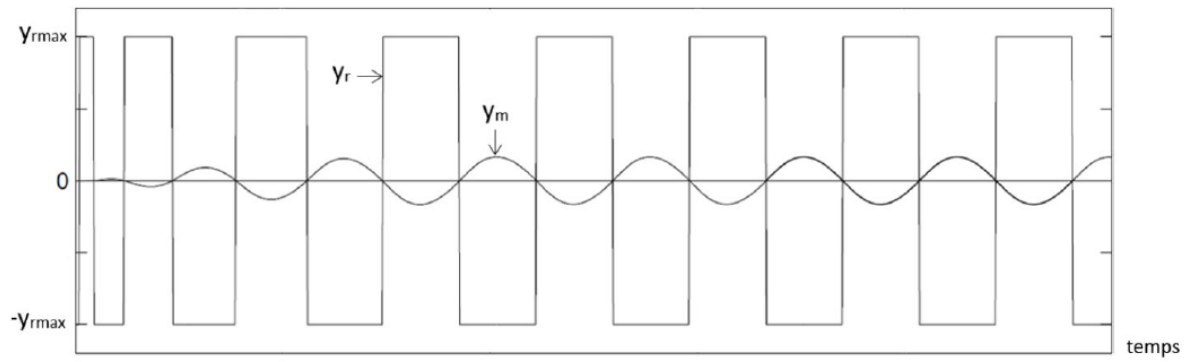


FIGURE 2 – Réponse théorique d'un régulateur TOR pour identification des paramètres

		Ms=2			Ms=1.4		
		a_0	a_1	a_2	a_0	a_1	a_2
Kp	$a_0 \cdot e^{(a_1 \cdot \kappa + a_2 \cdot \kappa^2)} \cdot K_c$	0.72	-1.6	1.2	0.33	-0.31	-1.0
Ki	$Kp/a_0 \cdot e^{(a_1 \cdot \kappa + a_2 \cdot \kappa^2)} \cdot T_c$	0.59	-1.3	0.38	0.76	-1.6	-0.36
Kd	$T_c \cdot a_0 \cdot e^{(a_1 \cdot \kappa + a_2 \cdot \kappa^2)} \cdot K_c$	0.15	-1.4	0.56	0.17	-0.46	-2.1
b	$a_0 \cdot e^{(a_1 \cdot \kappa + a_2 \cdot \kappa^2)}$	0.25	0.56	-0.12	0.58	-1.3	3.5

TABLE 3 – Paramètres selon AH

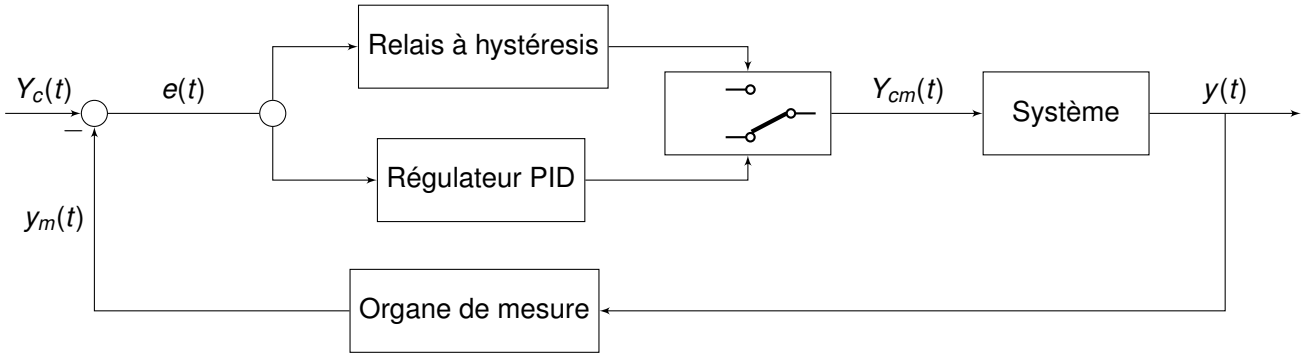


FIGURE 3 – Schéma de principe du régulateur PID pour l'auto-ajustage selon Åström et Hägglund

2.2.4. Résumé

Pour résumer, il existe une multitude de méthodes comportant des avantages et des inconvénients. La table 4 reprend l'essentiel établi dans ce rapport.

Il existe notamment une méthode appelée "placement des pôles" très convoitée dans les écoles. Le désavantage avec celle-ci, est qu'il faut connaître ou déterminer les fonctions de transfert du processus à régler. Cette méthode exige des connaissances mathématiques et/ou physique bien spécifiques. Le code en annexe 5 permet de calculer, de placer et de ploter la réponse du système à régler. La manière itératives est très efficace pour obtenir des résultats rapide mais peut être sujette à quelques mauvaises interprétations.

	Avantage	Inconvénient	Commentaire
Itération en trois phases	Facile à mettre en œuvre, adaptable, permet un contrôle fin	sensible aux erreurs humaines	Méthode empirique, Difficile d'optimiser
Ziegler & Nichols	simplicité, rapidité, robustesse	sensible aux variations, manque de précision, limité	Excellent point de départ, utile si la modélisation du système est très complexe
Åström et Hägglund	Bonne performance, robustesse, auto-ajustage	Plus complexe à mettre en œuvre,sensible aux perturbations, limité	Idéal pour des modèles linéaire, méthode rapide et robuste

TABLE 4 – Résumé des régulateurs

3. Expérimentation 1

Pour cette première partie, le système à régler est une balle de ping-pong à l'intérieur d'un tube (Colonne à lévitation). Un ventilateur fonctionnant comme un aspirateur commandé par une tension de 0 à $\pm 10 V_{dc}$ et alimenté par un amplificateur linéaire. L'air dans le tube est dépressurisé par aspiration en fonction de la vitesse de rotation du ventilateur (approximatif). Enfin un laser situé au sommet du tube permet la mesure de la hauteur de la balle. La tension de mesure est de 0 à $10 V_{dc}$. (figure 4). Le régulateur qui règle la hauteur de la balle est implémenté sur l'API (SAIA-PCD2). L'acquisition de la mesure se fait sur l'A/D (input analogique) et la commande se fait via un D/A (sortie analogique). La source de tension est limitée à $\pm 1.5 A$. L'objectif est que la balle de ping-pong atteigne la hauteur de 1m grâce à la méthode de Åström et Hägglund (auto-tunning).

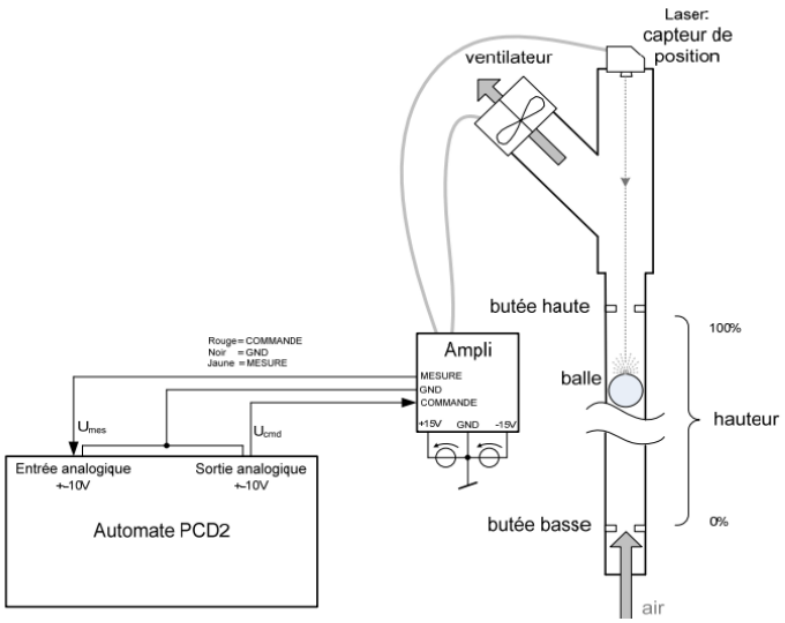


FIGURE 4 – Schéma de l'expérimentation n°1

3.1. Réalisation de la régulation

Travaux préparatoires

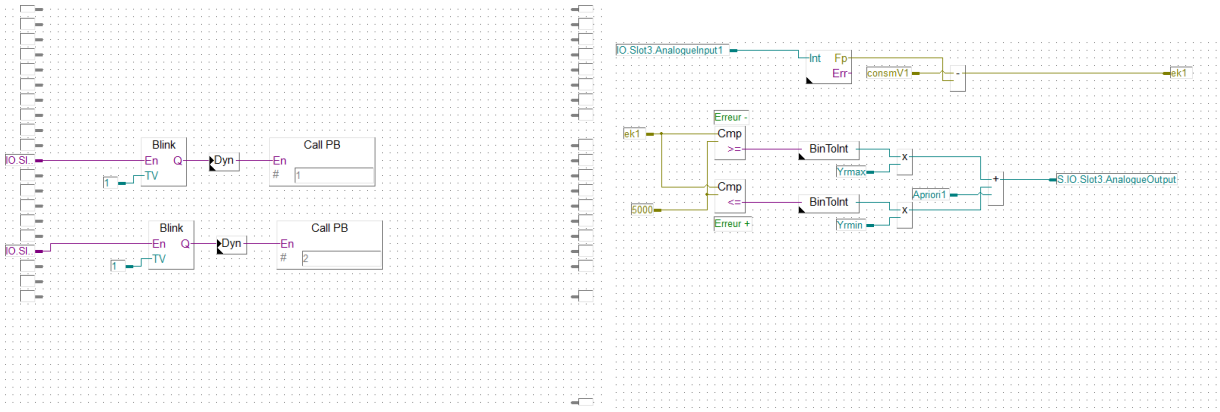
Quelques manipulations doivent être exécutées avant même l'implémentation du programme. Premièrement, il faut donner une valeur de consigne préalable au ventilateur pour compenser le poids de la balle (commande à priori). Celle-ci doit être à peine amorcée. Deuxièmement, il faut connaitre les caractéristiques du laser. Ici de simples tests ont permis de formuler la caractéristique en fonction de la hauteur : $f(x) = 50 \cdot x + 80$.

Environnement de programmation

Comme cité en introduction, le code est réalisé en langage FUPLA (bloc fonctionnel) propriété de Saia. Le code contient un programme principal (figure 5a) qui appellera les différentes fonctions. Une fonction dénommée "relais" (figure 5b) comporte les instructions pour faire osciller la balle et ainsi trouver les 3 paramètres cruciaux pour la régulation. Une fonction "PID" (figure 6) qui permet au système d'obtenir le résultat attendu.

Programme Main :

La fonction main comporte deux entrées numériques (switch on-off) qui commandent un bloc Blink chacun. La période de scintillement est de 200ms. Le bloc "Call PB 1" appelle la fonction Relay et "Call PB 2" appelle la fonction PID.



(a) Programme Main **(b)** Fonction Relay

FIGURE 5 – Fonction Main et Relay Saia

Fonction Relay :

L'entrée analogique lit la valeur du laser et la soustrait à la valeur de consigne du relais qui donne l'erreur. Cette erreur est comparée avec la caractéristique médiane du laser (au centre) ; si elle est plus petite, on la multiplie avec une constante (Yrmax). Si elle est plus grande, on la multiplie par l'opposé de cette constante (Yrmin=-Yrmax). En additionnant les deux sorties de multiplication (dont l'une des deux vaudra 0 en fonction de la condition) et la commande à priori, on obtient la sortie qui correspond à la commande à envoyer sur l'amplificateur et le ventilateur.

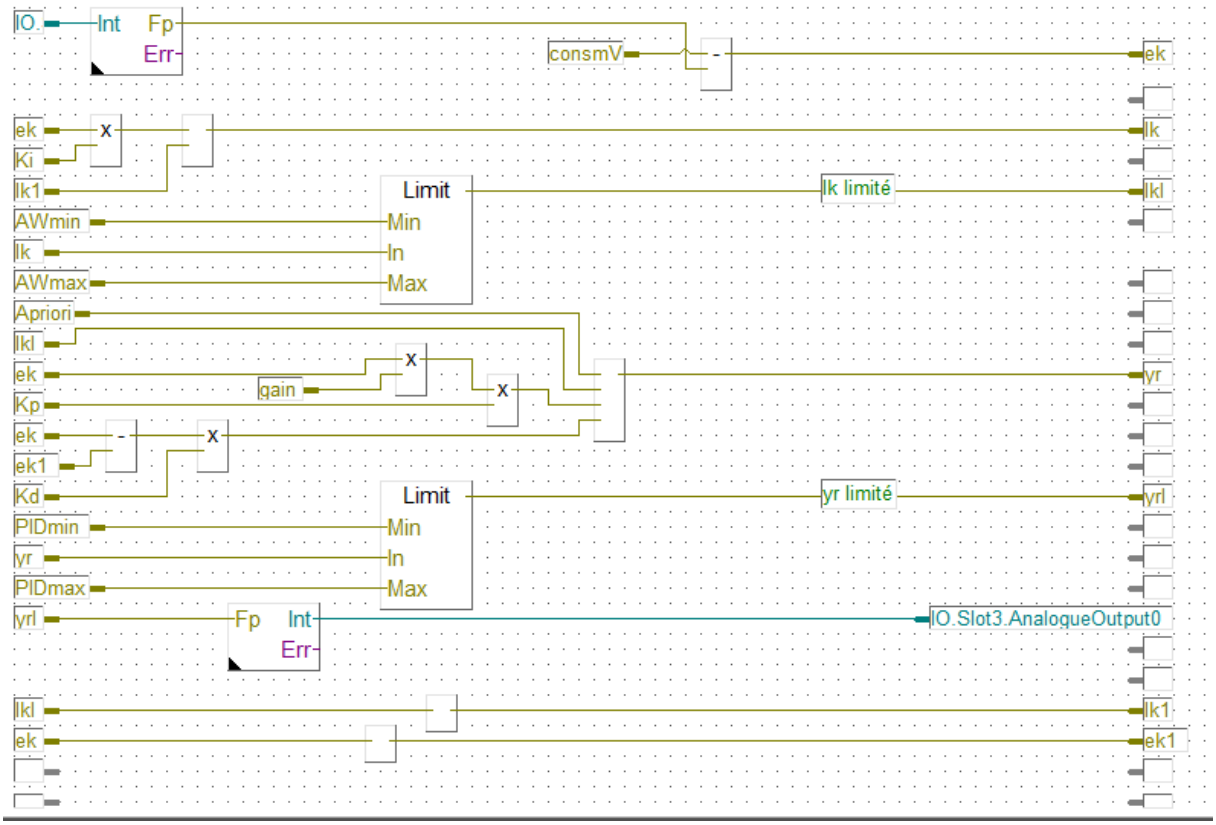


FIGURE 6 – Fonction PID

Fonction PID :

Pour la fonction PID, un organigramme est présenté en annexe figure 21, page C-2

	AH	ZN	itérative
Kp	0.0804	0.279	0.5
Ki	0.9597	0.2232	0.022
Kd	0.1038	0.0871	0.6

TABLE 5 – résumé des paramètres

3.2. Résultats

À ce stade, il est judicieux d'analyser l'oscillogramme de l'API. On remarque que le régulateur fait bien une commande TOR (tout ou rien). On peut alors extraire les données, pour l'instant manuellement.

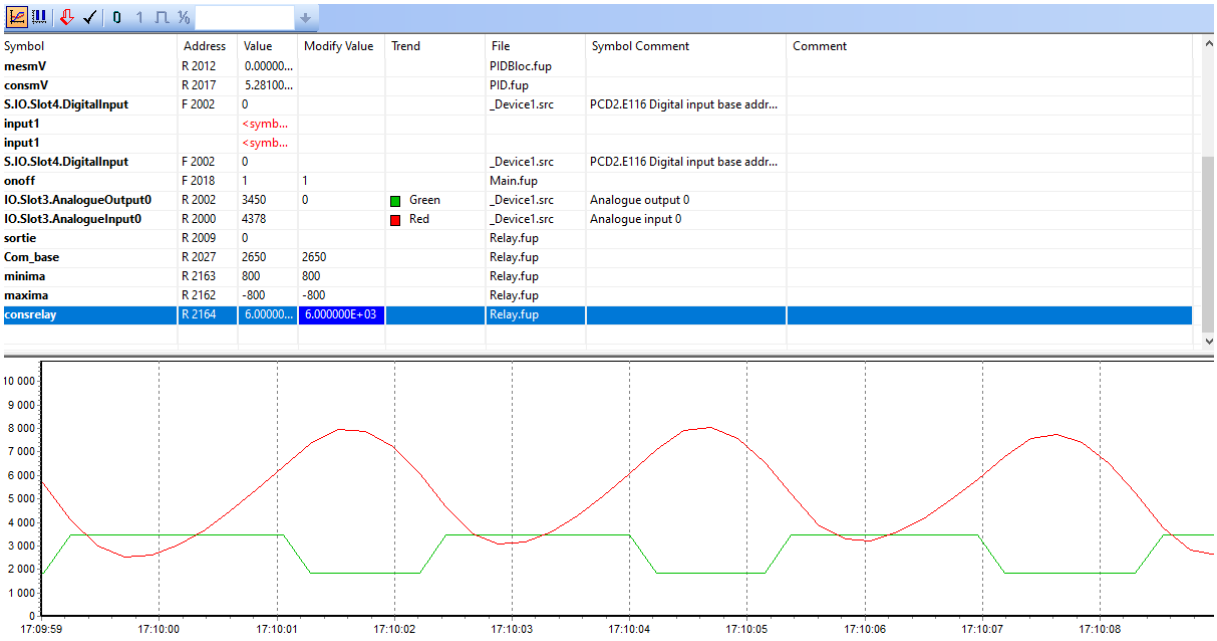


FIGURE 7 – Oscillogramme régulateur à relais (En rouge le signal de la mesure, en vert, la commande du ventilateur.)

On observe une période d'oscillation de 3,2 secondes, une amplitude du signal de mesure de 3000 et une amplitude de la réponse de 750. Grâce à ceux-ci, on peut déterminer les paramètres Kp,Ki, kd et b selon Åström et Hägglund. Dans ce cas, ils valent : Kp=0.0804, Ki=0.9597, Kd= 0.1038 (Où Kp=kp*b). A ce stade il semble judicieux de tester ces paramètres dans le régulateur PID implémenter dans l'API avant de réaliser l'algorithme auto-tunning. La réponse obtenue est à la figure 8. La régulation ne s'effectue pas correctement. La balle de ping-pong tape en butée sur le haut et le bas de la maquette. Afin de se faire une idée de la valeur des paramètres satisfaisants, il est utile d'utiliser une autre approche pour justifier des changements. Pour la suite, la méthode itérative est privilégiée. La figure 9 illustre la réponse. Les valeurs des paramètres sont largement différentes de celles obtenues avec la méthode AH. La méthode ZN présente elle aussi une mauvaise réponse avec la balle qui s'envole vers le sommet du tube et à l'inverse vers le bas. Le tableau ci-dessous résume les paramètres obtenus. (On remarque que pour la méthode itérative, le temps de réglage peut être amélioré).

En résumé, on obtient des gains très différents selon la méthode utilisée ; table 5. Afin d'obtenir une réponse viable mais convenable pour la méthode AH, il est nécessaire d'ajuster les paramètres. Ce qui est contradictoire au thème de ce projet. C'est pourquoi il est primordial de se questionner. La maquette est-elle adaptée à cette méthode ? La fréquence d'échantillonnage est elle suffisante ? On peut prétendre répondre par la négative à ces deux questions. Le système est très voir trop oscillant. De plus, la fréquence maximale à laquelle l'automate travaille est de 100 ms. Cette expérimentation

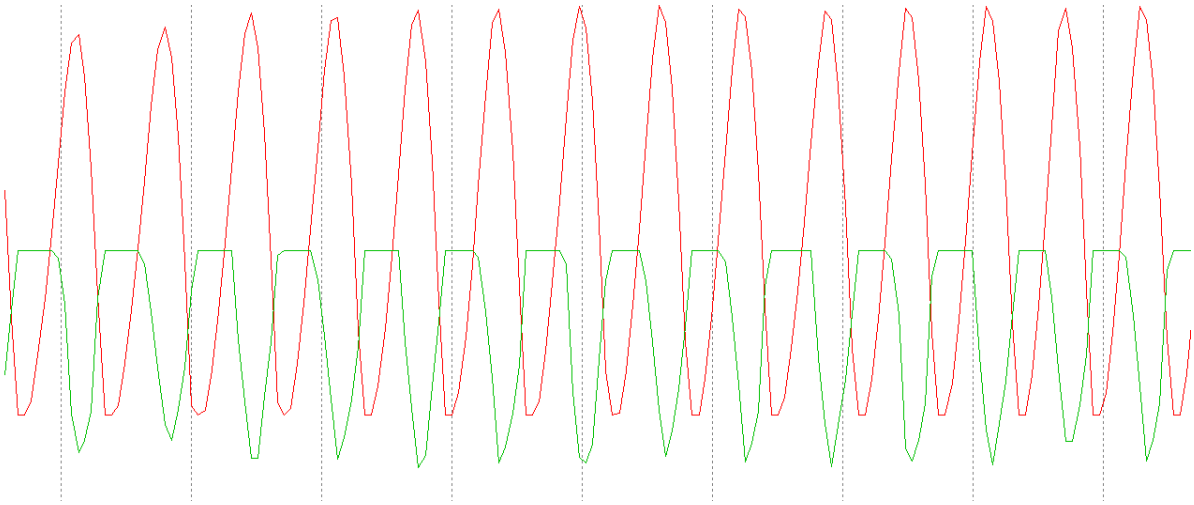


FIGURE 8 – Réponse du système avec les paramètres selon la méthode AH

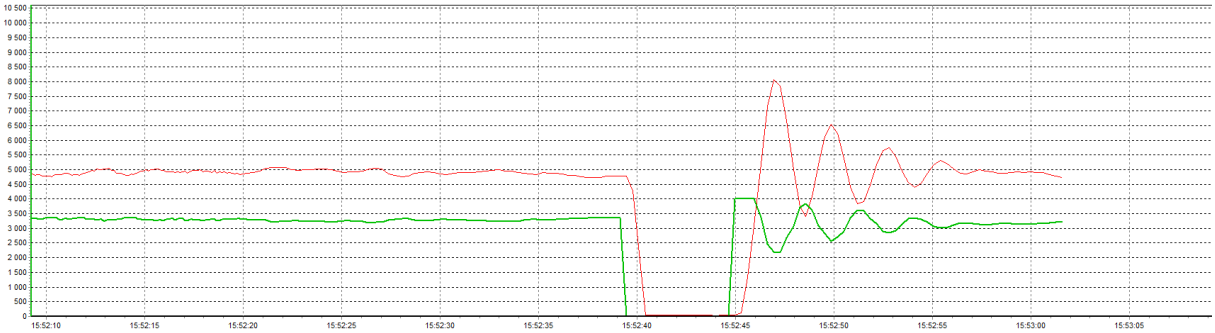


FIGURE 9 – Réponse du système avec les paramètres selon la méthode itérative

n'est pas adaptée à un tel régulateur. En partant de ce constat, et du fait que l'industrie travaille principalement avec des moteurs, une seconde maquette a été mise en place. Elle est décrite dans le point suivant.

4. Expérimentation 2

Pour cette deuxième expérimentation, le but reste inchangé. Il s'agit de donner une consigne de position et grâce au régulateur implémenté sur la carte Deflino, atteindre celle-ci. Le microprocesseur étant nettement plus rapide que l'API précédemment utilisé, devrait donner des résultats plus concluants. La figure 10, décrit l'installation. Le processus à régler est le moteur DC. La carte Delfino intègre le régulateur ainsi que différentes interfaces (Bouton poussoir, display). Celle-ci commande le pont H en PWM.La sortie du pont H commande le moteur. La mesure se fait à l'aide d'un codeur incrémental et est acquise par un ADC sur le microprocesseur.
Le langage de programmation pour cette carte est le "C". La base du programme à été récupérée afin de gagner du temps sur l'étude de la problématique. Le code en question se trouve dans le fichier "user.c" et "user.h" dont la fonction principale se nomme "UserTe". L'appel de cette fonction se fait grâce à un compteur qui l'exécute toute les 100μs.
Afin de pouvoir tester les phases les unes après les autres, le régulateur TOR a été fait en premier. Les calculs de paramètres ont été faits grâce au code Matlab 6. Puis le régulateur PID en déclarant des constantes comme paramètres précédemment calculé. Ces étapes sont entrecoupées de différents tests et mesurent pour valider l'implémentation et les calculs. Enfin, le régulateur autoajustable, est écrit dans son entier.

Note pour les utilisateurs
Le bouton 1 donne un saut de consigne. L'appui sur le bouton 2 lance la phase de calcul du régulateur autotuner.
Note pour les oscillogrammes :
le signal jaune correspond à la consigne, en bleu le signal mesuré et en vert le courant.

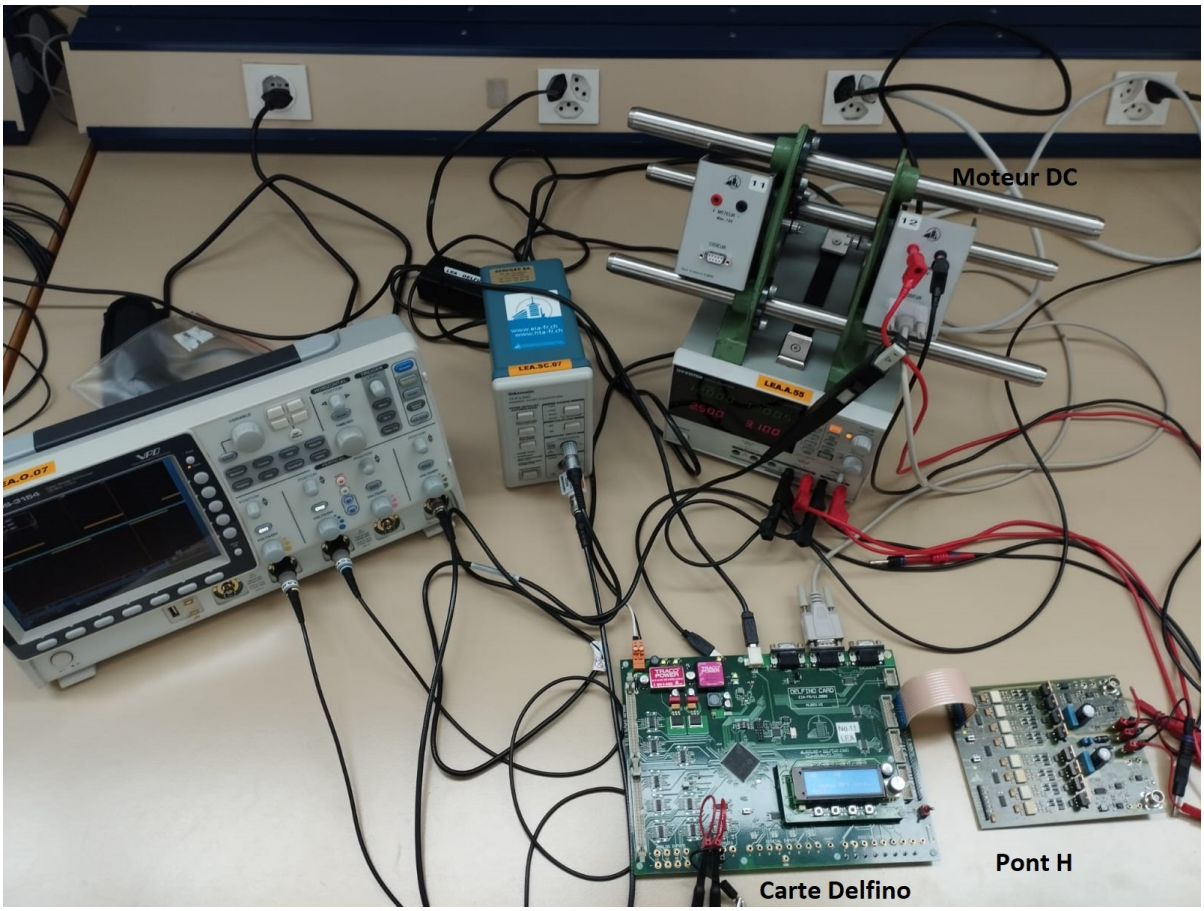


FIGURE 10 – Setup de test de la deuxième expérience

4.1. Régulateur TOR

Il s'agit de programmer un régulateur tout ou rien. C'est à dire, donner une consigne de position, une fois celle-ci atteinte inverser la consigne. La conséquence de cette commande est mesurée et devrait ressembler à une onde sinusoïdale. À noter que le signal peut être bruité. Le code 1 décrit la fonction void RegRelais(void).

```
1  void RegRelais(void)
2  {
3  ////////////////////Régulateur à Relais/////////////////////////////////
4    if(FirstRelais==true)               //initier le mouvement
5    {
6    FirstRelais=false;
7    mesure_t[0]=-0.4;
8    }
9
10   ecart_t[0] = mesure_t[0];
11
12   if(ecart_t[0]>0.3)
13   {
14   ucm=-consigneReg;
15   }
16   else if(ecart_t[0]<-0.3)
17   {
18   ucm=consigneReg;
19
20   }
21
22 }
```

Code 1 – Implémentation Régulateur TOR

Commentaires sur le code 1 :
Étant donné qu'aucune consigne n'est donnée, l'écart est de 0. Les lignes 4 à 8 imposent une mesure faussée pour initier le mouvement. Comme le booléen de la condition est sur false pour la suite de l'exécution, elle est faite une seule fois. La suite de la fonction inverse la commande du relais lorsque celle-ci est dépassée et inversement lorsqu'elle est négative. Afin que les oscillations soient visibles, on donne l'intervalle de -0.3 à 0.3.

Grâce à cet oscillogramme, il est possible de calculer les différents paramètres. Il suffit de repérer d'amplitude du signal de commande, l'amplitude du signal mesurer et la période d'oscillation. Avec les équations 4, 5 et 6 et la table 3 les paramètres, Kp, Ki, Kd et b sont facilement calculés. Le script Matlab renvoie les gains suivant : Kp=0.4478, Ki=0.0342, Kd=0.0037 (b est déjà multiplié avec kp dans le script). Le régulateur à relais peut être interrompu, le switch s'oriente vers le régulateur PID. À savoir que la sortie du DAC de la carte Delfino a été amplifiée de 5x. Il est judicieux de prendre les amplitudes selon la mesure de la carte et non la valeur de l'oscilloscope.

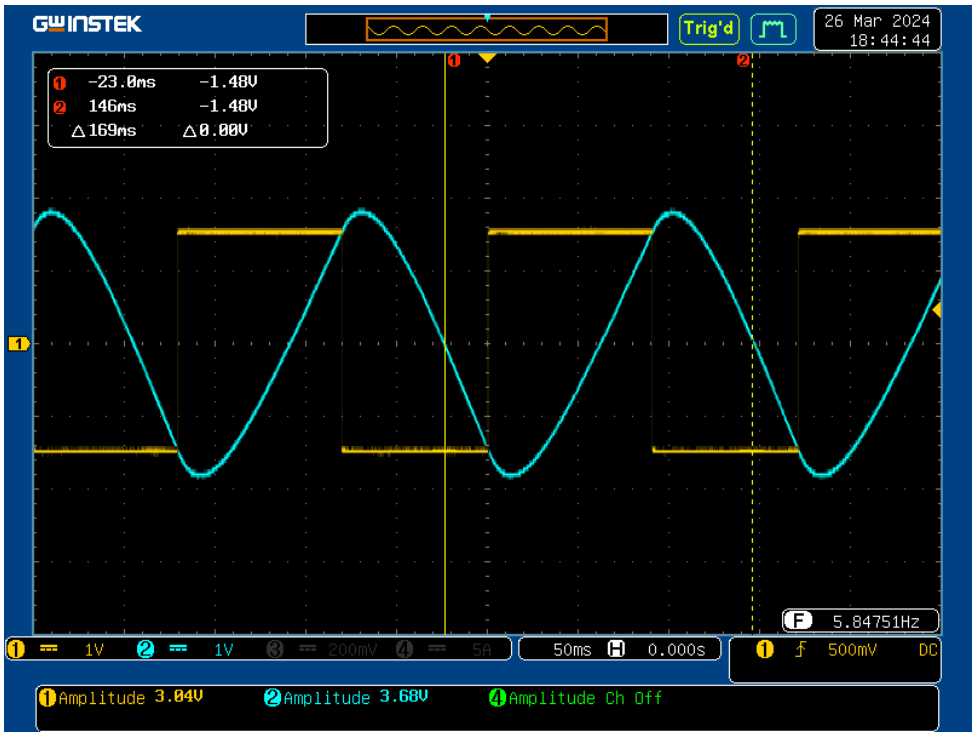


FIGURE 11 – Réponse du système pour la commande TOR

4.2. Régulateur PID

Comme le décrit la figure 1 ainsi que l'organigramme du régulateur PID numérique de la figure 21, un écart est calculé par entre la consigne et la valeur mesurée. Grâce à cet écart et l'écart précédent, on peut intégrer et dériver (numériquement). La commande régulée est l'addition de Yrp,Yri et Yrd selon le code 2.

```
float Kp=0.4478;
float Ki=0.0342;
float Kd=0.0037;

 ////////////////////////////Régulateur PID/////////////////////////////////////
ecart_t[0]=cons_t - mesure_t[0];          // calcul de l'erreur

Yrp= ecart_t[0];                         // Régulateur proportionnel

Yri[0]= Yri[1] + ecart_t[0]*Te;          // Régulateur intégrateur


// Anti Wind-up
if (Yri[0]>0.5)  {
  Yri[0] = 0.5;
}
else if (Yri[0]<-0.5) {
  Yri[0] = -0.5;
}

Yrd= (ecart_t[0]-ecart_t[1])/Te;         // Régulateur différenciateur

ucm= (Kp * Yrp + Ki * Yri[0] + Kd * Yrd); // sortie du régulateur

Yri[1]=Yri[0];                           // mise à jour des variables dans le temps
ecart_t[1]=ecart_t[0];
```

Code 2 – Implémentation Régulateur PID

```
//=====================================================================
//      MESURE DE LA POSITION ET DE L'ECART
//=====================================================================


comptCod = getQEP1Pos();              // Lecture du compteur de pos.
if (comptCod > ResolCod/2){      // Recentrage de la position mesurée
  comptCod -= ResolCod;            //   entre -1000 et +1000 [imp],
}                                  //   c'est-à-dire + ou - un 1/2 tour,
                  //   en partant de la plage par défaut
                  //   0 à 2000, qui présente donc une
                  //   discontinuité gênante en zéro.
mesure_t[1] = mesure_t[0];           // Mémorisation de la pos. préc.
mesure_t[0] = (float)comptCod/(float)ResolCod; // Pos. actuelle

if(FirstTime==true)                  //Initailisation du tableau de valeur Value à 0
{
  FirstTime=false;
  int i;
  for(i=0;i<nbrEchantillon;i++)
  {
    Value[i]=0;
  }
}
```

Code 3 – Mesure de la position et de l'écart

Commentaires sur le code 2 :

Les variables Kp,Ki et Kd sont déclarées en type float et initialisées d'après les valeurs du script Matlab, ceci afin de les tester. La première opération effectuée est de récupérer la valeur mesurée et la stocker dans la variable mesure_t. Cette commande peut être faite grâce à cette portion de code (3), qui lit le codeur incrémental. Yrp, Yri et Yrd sont les résultats des gains pour chaque opérateur. Une limite anti-wind-up sert de garde-fou pour l'intégrateur afin qu'il ne dépasse pas la consigne. La sortie du régulateur "ucm" est pilotée par l'addition de Yrp,Yri et Yrd. Enfin les variables sont misent à jour pour permettre d'intégrer et dériver au prochain appel de cette fonction.

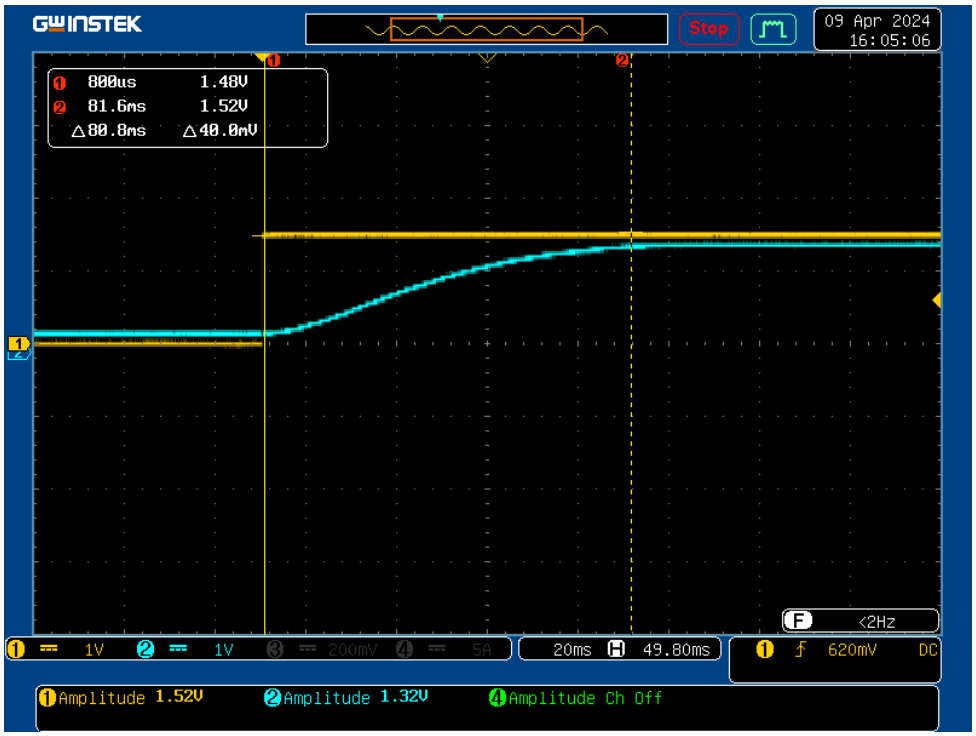


FIGURE 12 – Réponse du système pour la commande PID

On remarque que le moteur se comporte bien selon les commandes données. Aucun dépassement de consigne n'est à déplorer. La régulation se fait en 80ms. Pour une telle application, le temps de réglage est nettement suffisant. Un Kp plus grand permettrait une réponse plus rapide mais il faudrait ajuster les autres par la même occasion.
Grâce à cet essai, on remarque que la méthode d'Åström et Hägglund fonctionne bien. Il est question ensuite de s'attarder sur l'autoajustage.

4.3. Régulateur PID autoajusté

Pour la suite de ce projet, il faut mixer les différentes parties de code citées précédemment (code 1 et 2) et écrire un algorithme qui calcule la période d'oscillation, l'amplitude du relais et l'amplitude de la mesure. Le diagramme de flux 13 représente le déroulement de la fonction UserTe. Cette fonction est appelée toutes les $100\mu s$ par la fonction "main" grâce à un compteur. (ceux-ci ne sont pas décrits dans le présent rapport). Dans le fichier "user.c" se trouve aussi la fonction UserIdle qui gère l'affichage sur l'écran LCD et la gestion des boutons-poussoirs présents sur la carte. Le bouton bp1 sert à donner un saut de consigne. Le bp lance la phase de calcul des paramètres en initialisant des variables à zéro. Afin que le calcul s'effectue, il est nécessaire de mettre Kp, Ki et Kd à 0 pour ainsi satisfaire la condition ($Kp == 0 \&\& Ki == 0 \&\& Kd == 0$).

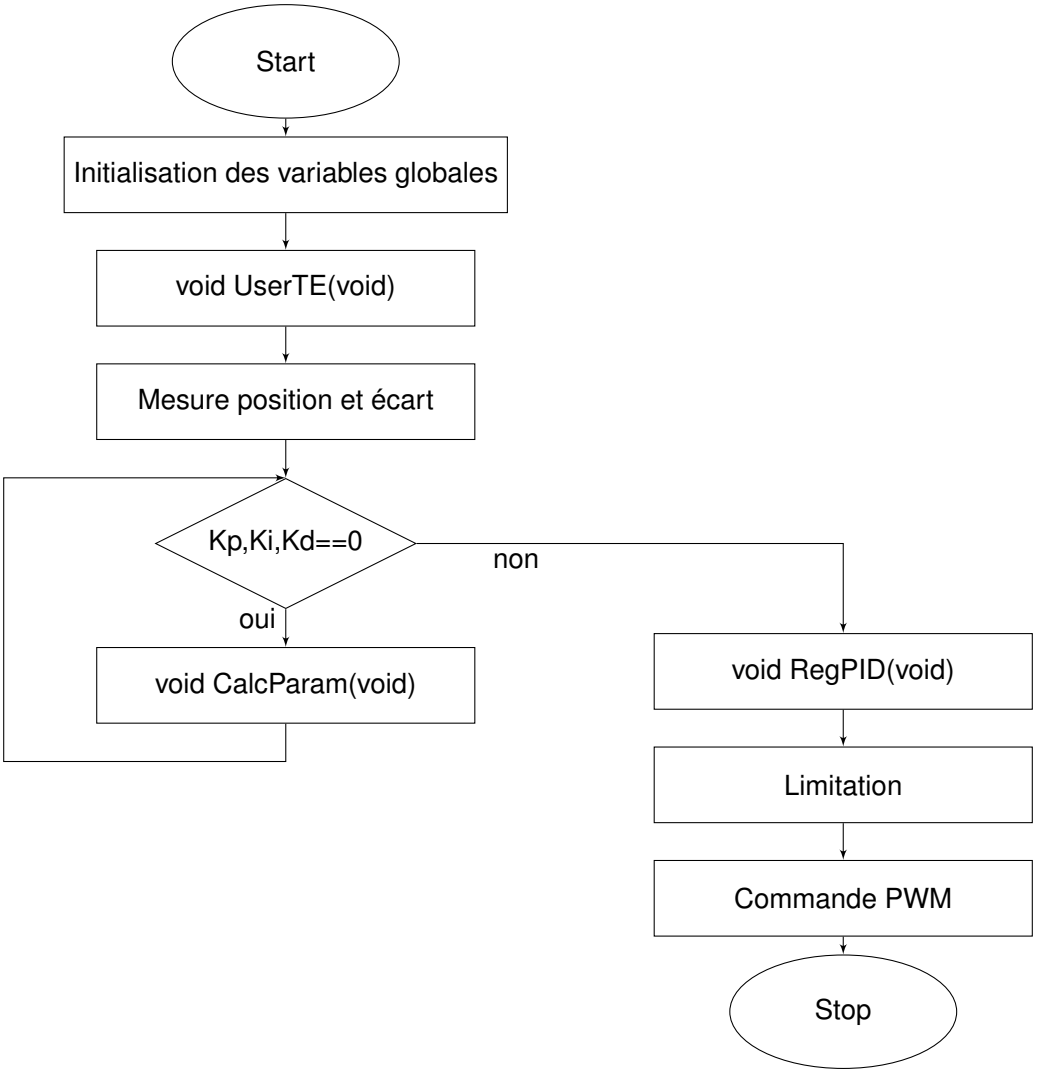


FIGURE 13 – Organigramme de la fonction UserTe

Commentaires sur le code **??** :

Premièrement, on exécute le régulateur à relais. On parcourt le tableau "Value[nbrEchantillon]" où nbrEchantillon =1000. On remplit à la suite chaque cellule avec la valeur de mesure_t[0], et on sort de la boucle une fois faite (cela accélère le processus). On teste si le tableau est entièrement rempli (le signal correspond à environ 1,5 sinusoïde). On parcourt le tableau à la recherche d'une valeur supérieure aux deux valeurs d'avant et d'après (correspondant à l'amplitude de la mesure). Si l'on ne teste que la valeur avant et après, les erreurs de maxima sont fréquentes. On stocke aussi la valeur de i. Une fois le premier maximum ainsi que son indice trouvé, on poursuit avec le deuxième maximum. La valeur de celui-ci importe peu, son indice par contre est utile pour déterminer la période. Sachant qu'entre i et i+1 il se passe un appel de UserTe qui correspond à $100\mu s$, on pourra calculer la période d'oscillation. À l'aide de la condition à la ligne 25 à 34 on test si les maximum1 et maximum2 ne sont plus égaux à -infini (initialement déclaré) et que l'indice du maximum2 n'est pas le même que le maximum1. Ceci prouve qu'une valeur a été assignée à chacune d'elle. Enfin, avant de calculer les paramètres, on s'assure encore une fois qu'il ait bien trouvé des valeurs pour les deux amplitudes. En ligne 48-64, on différencie deux cas de MS (marge de sensibilité) qui est définie comme constante en début de programme.

On remarque beaucoup de conditions, en effet, il faut s'assurer que le programme ne commence pas à calculer des valeurs dont certaines variables ne sont pas encore définies, que'elles sont égales à 0 ou -infini.

4.4. Résulats du régulateur autoajusté

Pour les réponses de la figure 15 et 16 le saut de consigne est de 0.4 dans le code (représente 2,4 sur l'oscilloscope) et pour la figure 17, le saut de consigne est de 0.2. Dans tous les cas, on observe la première phase qui consiste au régulateur à relais. Il effectue environ 1,5 sinus. Une fois les paramètres calculés, on voit le saut de consigne. Dès cet instant, le régulateur PID prend la main et régule le système. Dans les trois cas, la vitesse de régulation est bonne. Pour le meilleur des cas, c'est à dire moteur DC à vide avec MS=1,4, il régule en 60ms. La variante avec changement de dynamique, consiste à un accouplement du deuxième moteur DC présent sur la maquette. Celui-ci contraint le premier moteur en augmentant la charge à l'arbre et des forces de frottement supplémentaire.

Un problème est néanmoins à déplorer dans le code. Le temps que dure la phase du régulateur à relais est trop long (environ 250ms). Avec une période d'appel de la fonction de 100μs, il faudrait un tableau de 2500 valeurs, en float. La mémoire du microprocesseur de la carte n'étant pas infinie, il est judicieux de doubler la période d'appel de UserTe. Cette manœuvre ne péjore pas la précision des mesures. Les oscillogrammes représentés sont issus de la nouvelle période.

À ce stade, le code est fonctionnel. On ne prétend pas à un réglage parfait, mais tout de même satisfaisant. La variante de cette deuxième expérimentation avec la carte Delfino offre beaucoup de flexibilité. Ce langage de programmation ouvre la voie à de multiples possibles d'amélioration et d'optimisation. Ce niveau de souplesse dans le processus expérimental est essentiel pour répondre aux défis et aux exigences.

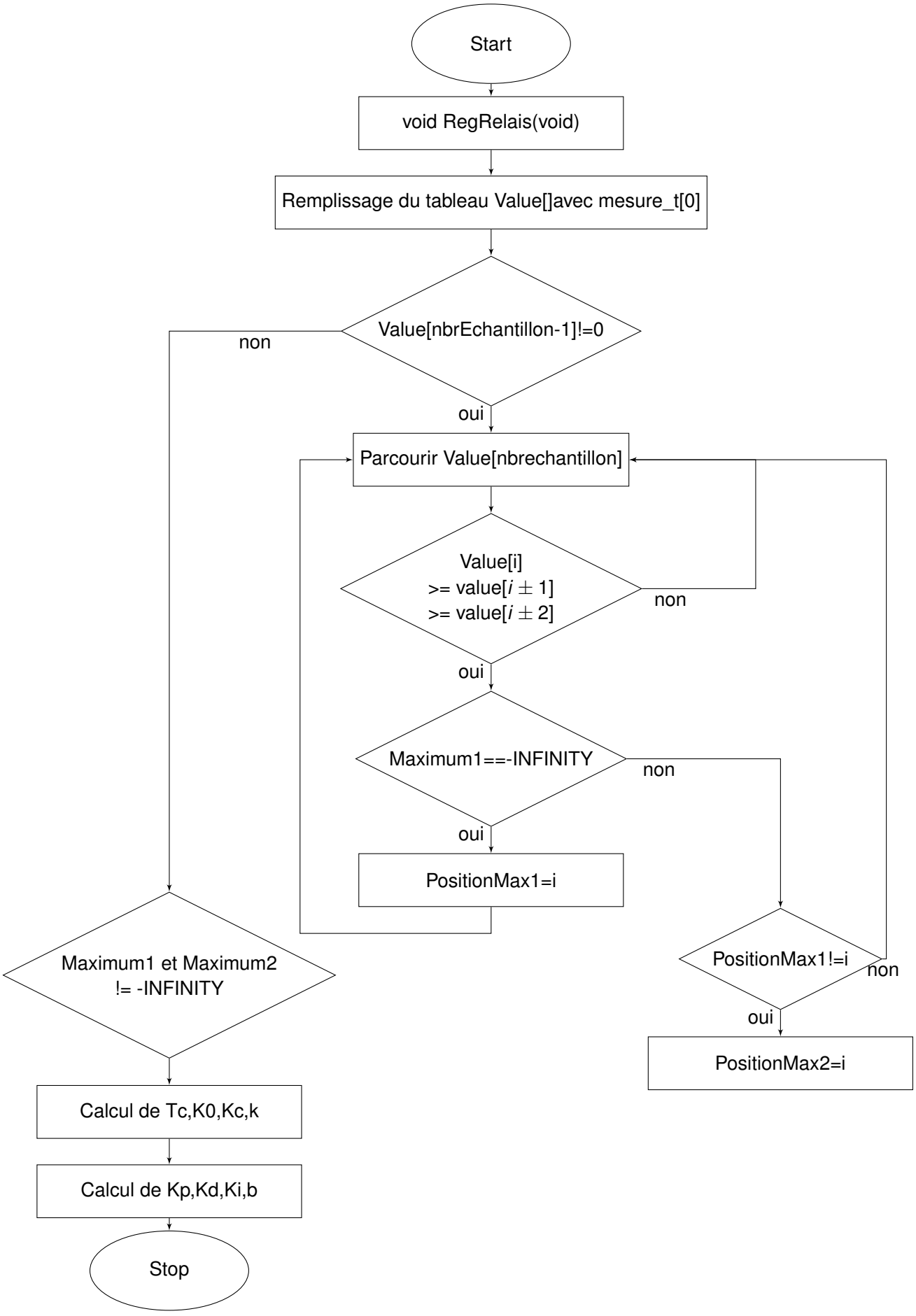


FIGURE 14 – Organigramme de la fonction CalcParam

```
1  void CalcParam(void)
2  {
3    ////////////////////Calculateur de paramètres/////////////////////////////////
4
5    RegRelais();
6    int index;
7    for(index=0; index<=nbrEchantillon-1;index++)            // Remplissage du tableau
       Value avec les valeurs mesurées
8    {
9      if(Value[index]==0)
10     {
11       Value[index]=mesure_t[0];
12       break;
13     }
14   }
15
16   if(Value[nbrEchantillon-1]!=0)                           //test si la dernière
       cellule du tableau est remplie (!=0)
17   {
18       int i;
19       for(i=100;i<nbrEchantillon-1;i++)              // Trouver les maximums locaux
       en passant les 100 premières cellules à cause des transitoires et ADC qui donnent
        des valeurs fausses
20       {
21         if(Value[i] >= Value[i-1]  && Value[i] >= Value[i-2]  && Value[i]>= Value[i
       +1] && Value[i] >= Value[i+2])
22         // trouver dans le tableau un maximum avec plusieurs condition à cause des
       transitoire et de l'ADC
23         {
24           if(Maximum1==-INFINITY)
25           {
26             Maximum1 =Value[i];
27             PositionMax1=i;
28           }
29           else if(PositionMax1 != i)
30           {
31             Maximum2 =Value[i];
32             PositionMax2=i;
33           }
34         }
35       }
36
37   }
38   //// calcul des paramètres selon AH
39
40   if(Maximum1 != -INFINITY && Maximum2 != -INFINITY)
41   {
42     float Tc = (PositionMax2 - PositionMax1) * Te; // Période d'oscillation
43     float K0 = consigneReg / Maximum2;
44     float Kc = (4 * Maximum2) / (M_PI * consigneReg);
45     float k = 1 / (K0 * Kc);
46
47     if(Ms==2) // Ms=2
48     {
49       b = 0.25* expf(0.56 * k - 0.12 * k * k);
50       Kp = 0.72 * expf(-1.6 * k + 1.2 * k * k) * Kc;
51       Kp= b * Kp;
52       Ki = (0.59 * expf(-1.3 * k + 0.38 * k * k) * Tc);
53       Kd = 0.15 * expf(-1.4 * k + 0.56 * k * k) * Tc;
54     }
55     else //Ms =1.4
56     {
57       b = 0.58* expf(-1.3 * k + 3.5 * k * k);
58       Kp = 0.33 * expf(-0.31 * k - 1.0 * k * k) * Kc;
59       Kp= b * Kp;
60       Ki = (0.76 * expf(-1.6 * k - 0.36 * k * k) * Tc);
61       Kd = 0.17 * expf(-0.46 * k - 2.1 * k * k) * Tc;
62
63     }
64     cons_t =0.2;
65   }
66
67 }
68
```

Code 4 – Fonction de calcul des paramètres selon AH

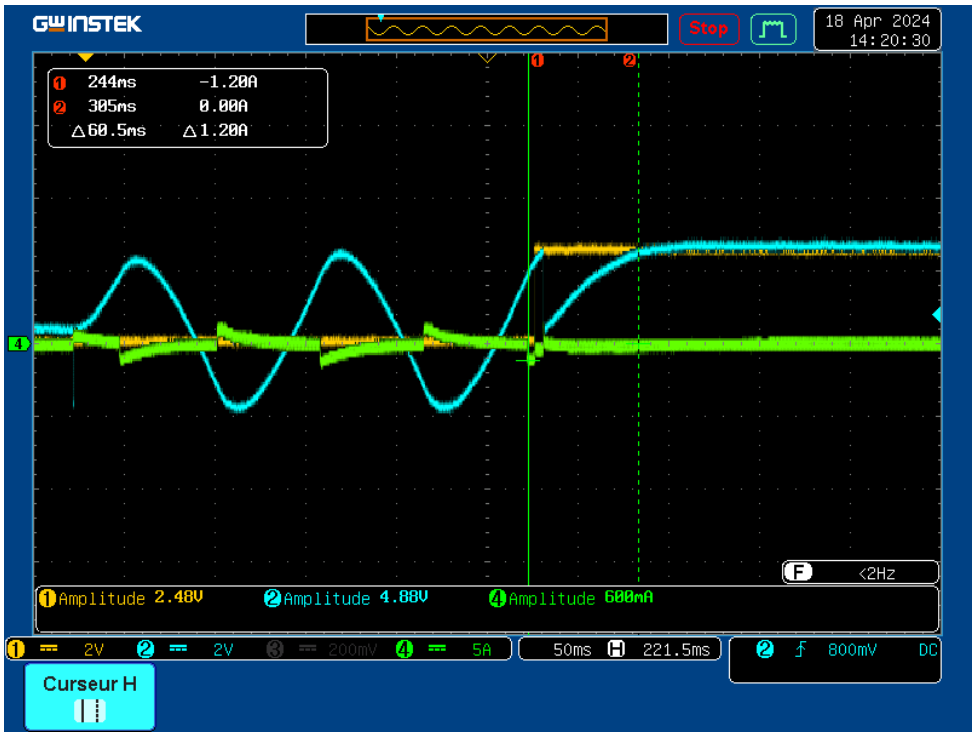


FIGURE 15 – Réponse du système pour la commande autoajustée avec MS=1.4

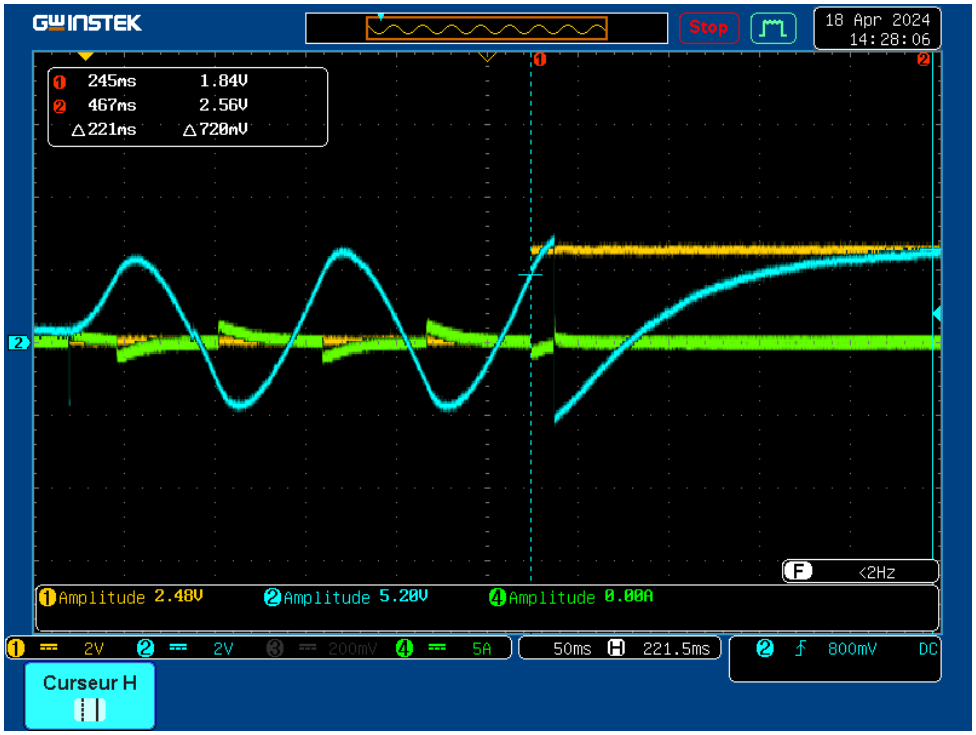


FIGURE 16 – Réponse du système pour la commande autoajustée avec MS=2

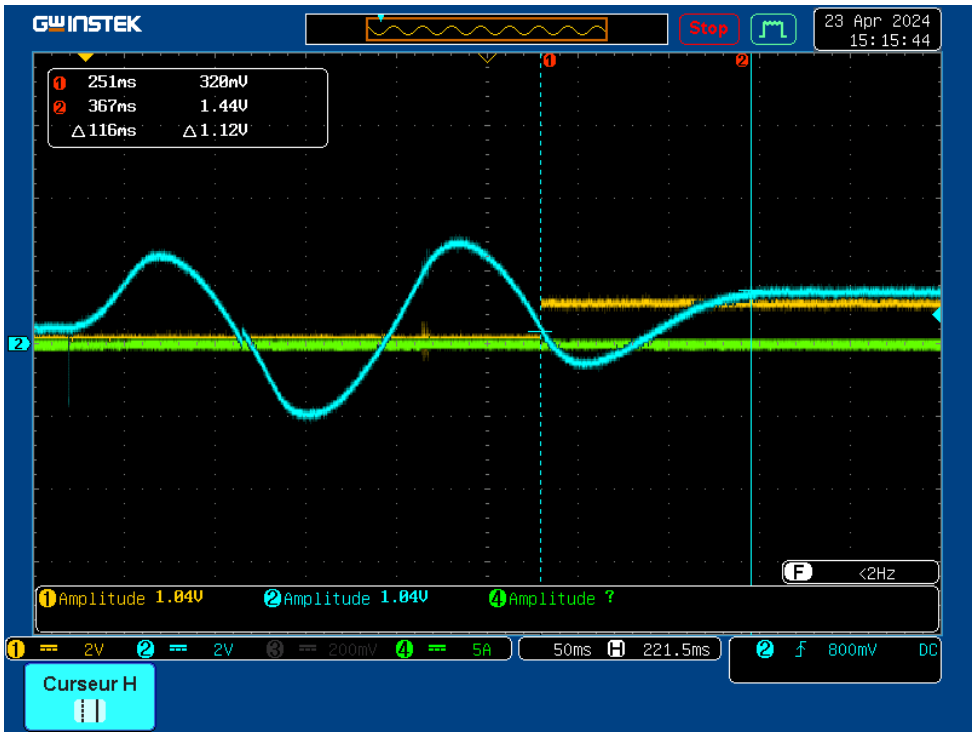


FIGURE 17 – Réponse du système pour la commande autoajustée, MS=1.4 avec changement de la dynamique

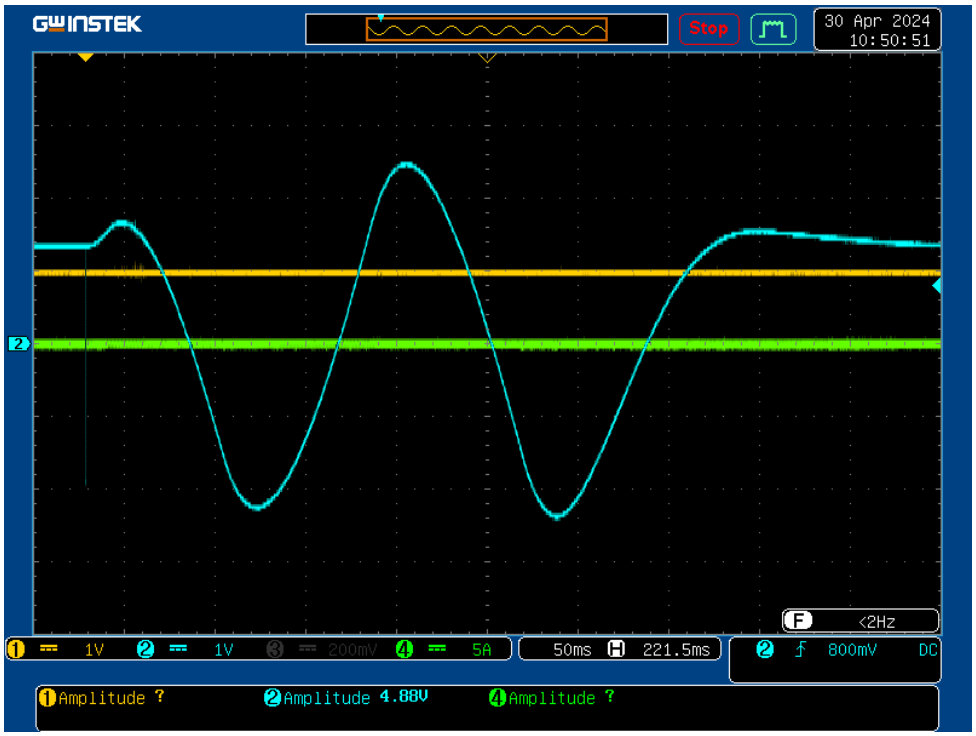


FIGURE 18 – Réponse du système pour la commande autoajustée, avec deux ondulations négatives et une consigne encore active

5. synthèse

Dans le cadre de ce projet de semestre, il a été question de réaliser un régulateur auto-ajusté. La méthode à appliquer a été celle de Åström et Hägglund qui consiste à créer une oscillation du système et mesurer 3 grandeurs clés. L'amplitude du régulateur, l'amplitude mesurée du système et enfin la période d'oscillation. À partir de ces valeurs, et grâce à la fonction $f(\kappa) = a_0 \cdot exp(a_1\kappa + a_2\kappa^2)$ avec les coefficients définit au tableau 3. Afin d'être efficace durant les phases d'expérimentation, les régulateurs ont été implémentés individuellement (Régulateur à relais et régulateur PID). Sur la maquette n°1, le système étant beaucoup trop oscillant, cette méthode ne fonctionne pas. Il est alors possible pour un tel cas d'utiliser la manière itérative en observant la réponse pour chaque gain changé. On remarque un gain intégrateur supérieur à 100 fois celui utilisé avec la méthode itérative. Les paramètres obtenus ne sont pas concluants. De plus, implémenter un régulateur autoajustable avec le langage de programmation de Saia aurait été fastidieux. Le fait de changer de maquette offre la possibilité de choisir un une commande avec la carte Delfino. L'implémentation de l'autorégulateur s'est déroulée de la même manière que cité précédemment. La différence est que les paramètres obtenus avec le régulateur TOR sont concluants. À partir de ces réponses, l'algorithme de l'autoajustage peut être fait. Le diagramme de flux 13 présente la réflexion de la fonction UserTe qui est appelée dans le cas final toutes les 200μs. Afin d'enlever le fruit du hasard, une autre dynamique a été appliquée au moteur en sortie. Une charge supplémentaire a été couplée à l'arbre du moteur DC. Le microprocesseur donne un saut de consigne immédiatement après le calcul des paramètres. (Celui-ci peut être observé avec le signal jaune sur les oscillogrammes précédents) La réponse avec le deuxième moteur couplé, est aussi bonne que sans. Cela implique que la régulation se fait de manière correcte.

Points sur lesquels des améliorations peuvent être faites dans le code.

1. La plupart des variables sont déclarées "globales"
2. Déclarer si possible des constantes pour les valeurs qui ne changent pas
3. Le tableau de valeur pour calculer les amplitudes et la période est très gourmand en ressources (dans ce cas 1000 valeurs en float)
4. Les boucles "if" imbriquées les une dans les autres pourraient être simplifiées ou optimisées (switch(case))
5. Les calculs en float sont long et certains peuvent être évités et reprenant la valeur dans une variable calculée précédemment
6. La structure générale peut être améliorée en créant des fonctions plus spécifiques

Les points cités ci-dessus ne sont qu'une liste non exhaustive basée sur le code existant. Il est bien entendu possible d'optimiser de manière optimale et de reconsidérer la manière de calcul des grandeurs critiques (amplitude max et période). Le programme est néanmoins fonctionnel et peut être adapté pour un tout autre système.
Après quelques tests prolongés, il s'avère que des problèmes surviennent quant à l'utilisation du régulateur dans son ensemble. Si on veut calculer les paramètres et que la consigne de position est sur une autre valeur que 0, l'oscillation présente deux ondes négatives et une seule positive (figure 18). Ceci empêche un calcul correct de la mesure et ne le fait pas remarquer à l'utilisateur. Un autre défaut se manifeste lors de l'appui sur bp1. En effet, après le saut de consigne le moteur tourne en continu alors qu'il devrait réguler à 0.4 (valeur par défaut). Mais lorsque la consigne est appliquée par l'interface de Code Composer Studio, le problème ne se manifeste jamais. Pourtant un système anti-rebond est implémenteé sur la carte.

Certains oscillogrammes présentent un écart entre la consigne et la position. L'écart sur le logiciel est très faible, mais il est amplifié par cinq lors de sa sortie du DAC.

6. Conclusion

En conclusion, ce projet de semestre relève quelques points importants. Tous les systèmes ne peuvent pas être auto-régulés de manière efficace. Malgré la performance de la méthode Åström et Hägglund, on remarque qu'elle n'est pas la solution miracle. Certains systèmes ne fonctionnent pas avec celle-ci. Il est important de noter que cette méthode donne un bon point de départ pour un réglage plus fin des paramètres. L'avantage d'un tel régulateur est d'entrainer des oscillations contrôlées sans risquer de déstabiliser la boucle de réglage. Cette technique d'auto-ajustage s'opère de manière complètement autonome. Il est possible de changer la dynamique du système et de recalculer les paramètres ensuite. Malgré que l'implémentation n'ait été testée que sur une maquette, le programme a été codé de manière "universelle" et pourrait donc commander d'autres systèmes. Concernant les gains obtenus, une vingtaine d'essais ont été faits. Les paramètres ont toujours été très proches, mais avec parfois des gains absurdes. (kp négatif, ki 10 fois supérieur aux précédents ...) Ceci appuie le fait que ce n'est pas LA solution, mais une aide à la régulation.

6.1. Conclusion personnelle

Ce travail aura été très bénéfique pour moi. La partie automatique m'a fait plonger dans les différentes méthodes de régulations pour la plupart déjà connues. La partie de code, elle, m'a donné quelques sueurs froides. J'ai quelques difficultés à penser comme un microprocesseur ainsi que de coder en "C". Je suis conscient que beaucoup d'optimisations peuvent être réalisées, malgré cela le code est fonctionnel. Pour résumer, il faut réfléchir au rythme de la carte et prendre point par point chaque variable pour un appel de fonction. Ensuite comprendre comment celle-ci est modifiée, quelle condition sera alors vraie ou fausse... En soi, j'imagine le programme au ralenti. La partie méthodologie n'est pas à laisser de côté. La structure permet de ne pas s'égarer ou perdre du temps sur certains sujets. Enfin, j'ai suivi mon planning avec quelques jours d'avance pour certaines tâches et l'inverse pour d'autres. Finalement l'écriture du rapport s'est faite tout au long du projet.

Je trouve que la programmation avec le PG5 est très limitée. Uniquement des fonctionnalités de base sont présentes. L'aspect visuel du programme peut être un atout. Comme j'ai pu le remarquer avec certains professeurs, la fonction d'auto-ajustage aurait été très fastidieuse à coder dans l'environnement de Saia.

Enfin, concernant les méthodes de régulation, je pense que la méthode de Ziegler et Nichols ainsi que celle de Åström et Hägglund donnent en principe de bons résultats, mais surtout un bon point de départ ! La méthode itérative permet d'avoir des régulateurs opérationnels très rapidement. La méthode de placement des pôles décrit la dynamique du système. Elle nécessite une bonne connaissance mathématique (Équation différentielle, transformée de Laplace, fonction de transfert, plan complexe...)

Dans l'exploration de ce projet, je réalise avec humilité qu'il n'existe pas de solution idéale ni de méthode parfaite. Cet objectif incessant de maîtriser notre environnement, nous confronte à une multitude de voies, chacune offrant ses propres défis et leçons. Cette diversité souligne l'importance de rester ouvert à différentes perspectives et approches.

Je tiens à remercier M. Justin Moncef Lalou qui m'a suivi durant ce projet. Il m'a aiguillé et donné de précieux conseils.

Fribourg, le 14 mai 2024

Maillard Dimitri

Références

[1] A. MUDRY, « Ajustage des paramètre d'un régulateur PID, » EIVD, Note d'application, 2002.

[2] N. N. J.G. ZIEGLER, *Optimum settings for automatic controllers*, 1942. adresse : `http://davidr.no/iiav3017/papers/Ziegler_Nichols_%201942.pdf`.

[3] K. ÅSTRÖM et T. HÄGGLUND, « *PID Controllers : Theory, Design and Tuning », Instrument Society of America, 2nd edition*. 1995.

[4] K. ÅSTRÖM, T. HÄGGLUND, H. C.C. et W. HO, *Automatic Tuning and Adaptation for PID Controllers - A Survey, Control Engineering Practice*, 1993. adresse : `https://www.sciencedirect.com/science/article/abs/pii/096706619391394C`.

A. Logiciels utilisés

Nom	Version	Éditeur	Utilisation
TeXstudio	4.7.3	TeXstudio	Éditeur de documents LaTeX
PG5	V2.3.195	Saia	Editeur de circuit Logique
Matlab	R2021b	Mathworks	Traitement des données
CodeComposerStudio	V11.1.0	TexasInstrument	Environnement de dév.
GitHub	V3.3.13	GitHub	système de contrôle de version

B. Matériels utilisés

	n°
Maquette	5
Alimentation	LEA-A-2
Automate	LEA-PCD2.M5 10

FIGURE 19 – Matériel expérimentation n°1

	n°
Banc moteur	11-12
Alimentation	LEA-A-55
Sonde de courant	LEA.SC.07
Oscilloscope	LEA.O.07
Carte Delfino	11
Carte pont H	20

FIGURE 20 – Matériel expérimentation n°2

C. Annexe

Cette annexe regroupe les liens vers les ressources essentielles associées à ce projet. En particulier, nous fournissons ici le lien vers le dépôt GitHub, où le code source complet du projet est disponible. Ce dépôt constitue une ressource précieuse pour ceux qui souhaitent explorer davantage ce travail, contribuer à son développement ou simplement en apprendre plus sur sa mise en œuvre. Nous vous encourageons à visiter ce dépôt pour accéder au code source, aux instructions de déploiement et à d'autres documents pertinents pour une meilleure compréhension du projet.

— GitHub : `https://github.com/Dimi1452/Projet-de-semestre-6`

C.1. Code pour le placement des pôles

```
%%%%%% Methode de placment des pôles %%%%%%
clc;close all;clear all;clear workspace;
REG = "PID" ; %% Selection du régulateur (H'r(s))
s1 = -20;     %% Si PI,PD,PID choisi comme régulateur
s2 = -100;  %% Si PID choisi comme régulateur

% Attention à avoir le numérateur et dénominateur de même taille !!
% H'r
if REG=="P" % Hpr(s)=1
  numHpr=[1];
  denHpr=[1];
elseif REG=="PI" % Hpr(s)=(s-s1)/s
  numHpr=[1, -s1];
  denHpr=[1, 0];
elseif REG=="PD" % Hpr(s)=1-s/s1
  numHpr=[-1/s1, 1];
  denHpr=[0, 1];
elseif REG=="PID" % Hpr(s)=(s-s1)(s-s2)/(-s(s1+s2))
  numHpr=[1, -s1-s2, s1*s2];
  denHpr=(-s1-s2)*[0, 1, 0];
end

% H'r customisé. Commenter si non utilisé
% REG="?"
%numHpr=[1, 8e5];
%denHpr=[1, 0];

% Hcm
numHcm=[1.2];
denHcm=[1];

% Hs
numHs=[0 0 1];
denHs=[1 1 0.26];

% Hm
numHm=[0.2];
denHm=[1];

%%%%%% Lieux des pôles %%%%%%
% calcul de HL
[numHL, denHL] = series(numHcm, denHcm, numHs, denHs);
[numHL, denHL] = series(numHL, denHL, numHm, denHm);
[numHL, denHL] = series(numHL, denHL, numHpr, denHpr)
% LP de HL
figure; sgrid('new');
rlocus(numHL, denHL);
title("LP régulateur " + REG)

% Hit target
[kp, poles]=rlocfind(numHL, denHL);

% Choix de Kp
```

```
54  % kp=2
55
56  %%%%%% Réponse indicielle %%%%%%
57  % calcul de la fonction de transfert en bouclefermee
58  [numH0, denH0]=series(kp*numHcm, denHcm, numHs, denHs);
59  [numH0, denH0]=series(numH0, denH0, numHpr, denHpr);
60
61  % calcul de Hbc
62  [numHbc, denHbc]=feedback(numH0, denH0, numHm, denHm);
63
64  %%%%%% plot de la response indicielle %%%%%%
65  figure;
66  step(numHbc, denHbc);
67  title("réponse indicielle régulateur " + REG)
```

Code 5 – Code paramètre régulateur méthode de placement des pôles

C.2. Organigramme régulateur PID numérique

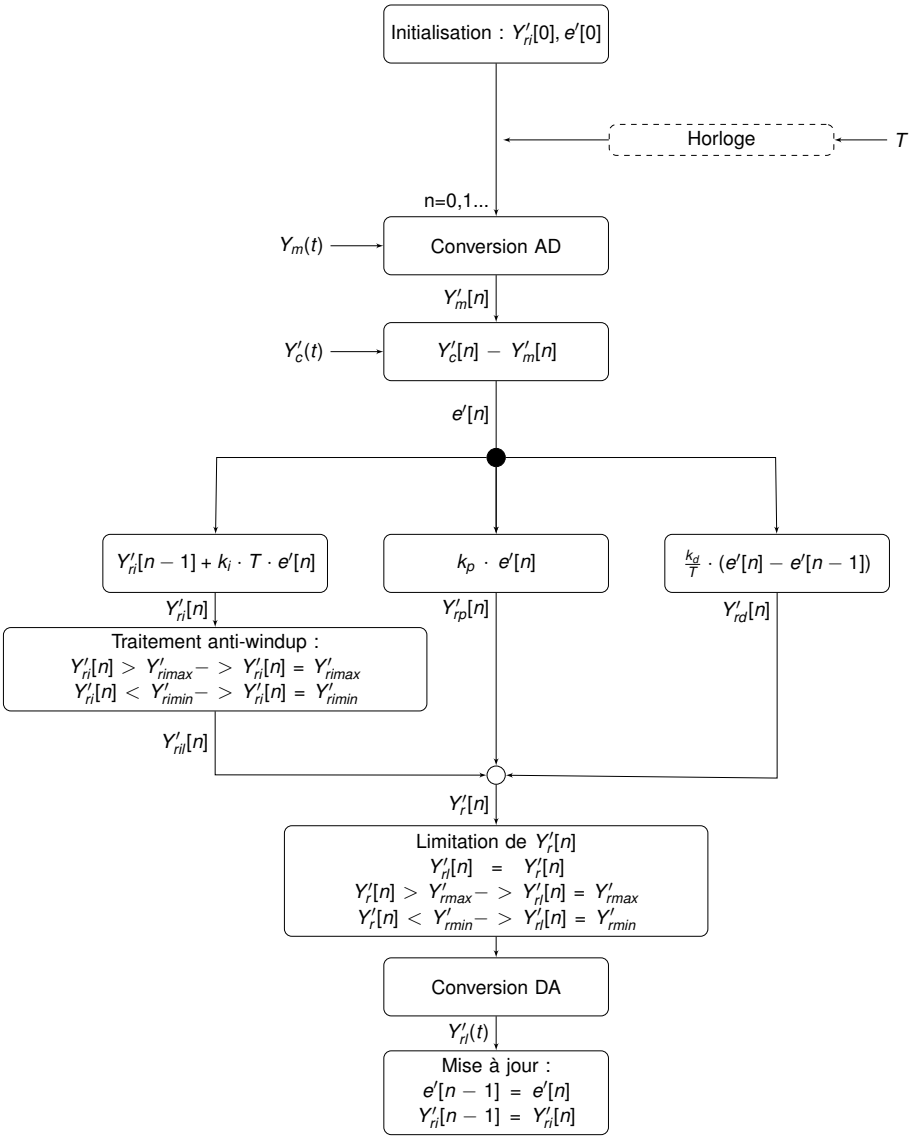


FIGURE 21 – Organigramme du régulateur PID numérique

C.3. Code Matlab paramètres méthode AH

```
%% Calcul des paramètres PID par l'autoajustage selon Astrm et Hagglund
clc;close all;clear all;
Ar= (4000-0)/2;% Aplitude relais max
Am= (10000-0)/2;% Amplitude mesure max
Tc=3.2; %Période d'oscillation
Ms=1.4; %coefficient de sensibilité 1.4 ou 2

K0=Am/Ar;% gain statique
Kc=4*Ar/(pi*Am); %
k=1/(K0*Kc);

  if Ms==1.4
    kpa0=0.33;kpa1=-0.31;kpa2=-1;
    kia0=0.76;kia1=-1.6;kia2=-0.36;
    kda0=0.17;kda1=-0.46;kda2=-2.1;
    ba0=0.58;ba1=-1.3;ba2=3.5;
    Kp= kpa0*exp(kpa1*k+kpa2*k^2)*Kc;
    Ti= kia0/(exp(kia1*k+kia2*k^2)*Tc);
    Td= kda0*exp(kda1*k+kda2*k^2)*Tc;
    b=ba0*exp(ba1*k+ba2*k^2);

  else
    kpa0=0.72;kpa1=-1.6;kpa2=1.2;
    kia0=0.59;kia1=-1.3;kia2=-0.38;
    kda0=0.15;kda1=-1.4;kda2=0.56;
    ba0=0.25;ba1=0.56;ba2=-0.12;
    Kp= kpa0*exp(kpa1*k+kpa2*k^2)*Kc;
    Ti= kia0/(exp(kia1*k+kia2*k^2)*Tc);
    Td= kda0*(exp(kda1*k+kda2*k^2)*Tc);
    b=ba0*exp(ba1*k+ba2*k^2);

  end

  Kp=Kp*b;
  Ki=1/Ti;
  Kd=1/Td;
  display(Kp);
  display(Ki);
  display(Kd);
  display(b);
```

Code 6 – Code Matlab: calcul des paramètres du régulateur selon la méthode AH