

IMT Atlantique

PROTOROB - Prototypage de systèmes

robotisés

4 Rue Alfred Kastler, 44300 Nantes

Téléphone : +33 (0)2 51 85 81 00

Télécopie : +33 (0)2 29 00 10 12

URL : www.imt-atlantique.fr



Rapport de projet - GROUPE 1

UE E - Prototypage de systèmes robotisés

PROJET PROTOROB : CONCEPTION ET FABRICATION D'UN ROBOT THERMOMÈTRE

Destinataires : Fabien Claveau, Vincent Lebastard, Alessandro Colotti

Grégory BOURSIN
FISE A2 ASCy

Loïc FOURNIER
FISE A2 ASCy

Cédric LEGER
FISE A3 ASCy

Emilien WOLFF
FISE A2 ASCy

Respo. TAF : Fabien CLAVEAU



IMT Atlantique

Bretagne-Pays de la Loire

École Mines-Télécom

Sommaire

1. Présentation du projet et cahier des charges	3
1.1. Contexte	3
1.2. Introduction au projet	3
1.2.1. Plan d'attaque	3
1.2.2. Travail à réaliser	4
1.2.3. L'équipe	4
1.3. Cahier des charges du projet	4
2. Conception	6
2.1. Choix de la chaîne électromécanique	6
2.2. Partie mécanique	6
2.2.1. Dimensionnement général	6
2.2.2. Conception assistée par ordinateur	7
2.3. Partie électronique	7
2.3.1. Carte électronique et capteur	7
2.3.2. Motorisation et codeur incrémental	7
2.3.3. PinOut et circuit final	9
2.4. Programmation	10
2.4.1. Calibration de l'aiguille	10
2.4.2. Loi de commande	10
2.4.3. Résolution de calcul et d'affichage	12
2.4.4. Mise en place d'un timer et d'une interruption	12
2.4.5. Organigramme de fonctionnement du code	13
3. Réalisation, développements et tests unitaires	14
3.1. Impression des pièces	14
3.2. Implémentation du code	15
3.2.1. Librairies utilisées	15
3.2.2. Réglage du PID sous Matlab	15
3.3. Premier prototype	17
3.3.1. Tests fonctionnels hors mécanique	17
3.3.2. Erreurs de conception	17
3.4. Intégration définitive	18
4. Conclusion	19
4.1. Résultats	19
4.2. Conclusion	19
Annexes	20
Annexe 1 – Code Arduino du ProtoRob	21
Annexe 2 – Code Matlab du projet	26
Références	28

Liste des figures

1.1. Différents types de thermomètres disponibles dans le commerce à affichage analogique et numérique	3
2.1. Premier schéma bloc du système	6
2.2. Schéma du cadran	6
2.3. Capture du modèle 3D du robot thermomètre fait sous <i>Fusion 360</i> (Autodesk)	7
2.4. Documentation technique du motoréducteur	8
2.5. Motoréducteur utilisé	8
2.6. Codeur incrémental : disque optique et signaux A (rouge) et B (bleu) en quadrature	8
2.7. Fonctionnement d'un signal PWM	9
2.8. Schéma du circuit électronique	10
2.9. Butée	10
2.10. Schéma bloc de l'assemblage PWM/Moteur	11
2.11. Principe de fonctionnement du Timer 1 en mode CTC	12
2.12. Registre de comptage TCNT1	12
2.13. Drapeau de comparaison OCF1A du Timer 1	12
2.14. Configuration du registre TCCR1B	13
2.15. Organigramme compact et lisible du thermomètre à aiguille avec PID	13
3.1. Impression du boîtier sur Bambulab P1S	14
3.2. Réponse à un échelon 30°	17
3.3. Première version du boîtier, avec butée et sans guide	17
3.4. Fixation du cadran et mise en place des composants électroniques dans le boîtier	18
3.5. Le <i>Thermorob</i> (vue de face et vue arrière)	18
3.6. Le <i>Thermorob</i> en fonctionnement et chauffé	18
4.1. Tracé des couples de température et incertitude associée	19

Liste des tableaux

1.1. Tableau des fonctionnalités, avec leur(s) critère(s) de valeur, leur niveau(x), leur flexibilité et leur priorité. FP : fonction principale, FS : fonction secondaire, FC : fonction de contraintes.	5
3.1. Bibliothèques Arduino utilisées et leur rôle	15
3.2. Arguments principaux de la fonction <code>pidtune</code> de MATLAB	16

Chapitre 1

Présentation du projet et cahier des charges

1.1. Contexte

Mesurer une température peut s'effectuer de différentes manières, et la façon de communiquer la température à l'utilisateur du dispositif peut également varier. Voici quelques exemples que l'on peut trouver dans le commerce :

- **Thermomètre à liquide** : la température est déduite de la dilatation du liquide qui monte ou descend dans un tube gradué.
- **Thermomètre bimétallique** : la température provoque la déformation d'une lame bimétallique qui entraîne une aiguille.
- **Thermistance (NTC/PTC)** : la température est mesurée via la variation de résistance électrique du matériau.
- **Sonde RTD (PT100/PT1000)** : la température est déterminée à partir de la variation quasi linéaire de la résistance du platine.



FIGURE 1.1 – Différents types de thermomètres disponibles dans le commerce à affichage analogique et numérique

Notre mesure de température s'effectuera par un capteur de température MPL3115A2, et son affichage sera analogique, composé d'un cadran et d'une aiguille. L'objectif est de motoriser et commander en position cette aiguille de manière précise en fonction de la température grâce à un système de commande basé sur un microcontrôleur Arduino Mega.

1.2. Introduction au projet

1.2.1. Plan d'attaque

Étant donné que l'objet du projet est d'aborder le prototypage rapide, il sera nécessaire d'aborder plusieurs points. On cherchera alors successivement à

1. Imaginer un montage électronique permettant de répondre à la problématique
2. Concevoir des pièces en 3D sur un logiciel de CAO
3. Assembler le robot et programmer le microcontrôleur
4. Finalement, tester le capteur en conditions réelles.

1.2.2. Travail à réaliser

Sur le plan mécanique, il s'agit de concevoir et d'imprimer en 3D l'aiguille et le support du moteur. Sur le plan électronique, l'objectif est d'asservir le moteur en position afin de placer l'aiguille sur la graduation correspondant à la valeur fournie par le capteur de température. Ceci nécessite une calibration de l'aiguille au démarrage, car on utilise un codeur incrémental.

Du côté logiciel, il s'agit de programmer le microcontrôleur pour mettre en œuvre un régulateur PID en temps discret, qui permet de positionner l'aiguille avec précision. Le code inclut la gestion des interruptions et des timers pour garantir un asservissement fiable et éviter les phénomènes d'emballement.

1.2.3. L'équipe

L'équipe est composé de quatre étudiants, tous en TAF ASCy.

- ▷ **Emilien Wolff** : Chef de projet et responsable CAO
- ▷ **Loïc Fournier** : Co-responsable du code arduino
- ▷ **Gregory Boursin** : Co-responsable du code arduino
- ▷ **Cedric Léger** : Responsable câblage et rendu

1.3. Cahier des charges du projet

OBJECTIF

Conceptualiser un robot thermomètre qui remplit la fonction suivante :

Afficher précisément la température sur un cadran circulaire.

Nous avons dès lors posé dans la [TABLE 1.1](#) le cahier des charges associé à cet objectif.

Fonction	Critère(s)	Niveau(x)	Flexibilité	Priorité
FP1 – Affichage précis de la température avec l'aiguille	Précision	-20 à 60°C	±1°C	1
	Rapidité	2s	NON	
	Stabilité	Aucune oscillation en régime permanent	NON	
FP2 – Transportabilité	Masse	< 500g	facile à transporter	1
	Dimensions du boîtier	Longueur < 10 cm ; Largeur < 10 cm ; Hauteur < 10 cm	NON	
FC1 – Modélisation et impression 3D	Utilisation du matériel imposé	Utilisation de tous les composants électroniques essentiels	Ajout d'une ringled Breadboard facultative	1
	Limiter l'utilisation de plastique	Nombres d'impression du prototype < 3	NON	
FS1 – Affichage de la température avec la ringled	Affichage	Leds allumées entre -20°C et la température mesurée	±5°C	2
	Signature lumineuse	Degradié de couleurs bleu(= -20°C) → vert(= 20°C) → rouge(= 60°C)		
	Consommation électrique	Faible intensité lumineuse		

TABLE 1.1 – Tableau des fonctionnalités, avec leur(s) critère(s) de valeur, leur niveau(x), leur flexibilité et leur priorité. FP : fonction principale, FS : fonction secondaire, FC : fonction de contraintes.

Chapitre 2

Conception

2.1. Choix de la chaîne électromécanique

Nous pouvons dans un premier temps proposer ce schéma bloc qui décrit le fonctionnement du système. Il prend en consigne la température T et intègre une boucle de rétroaction, qui asservit le système en position angulaire.

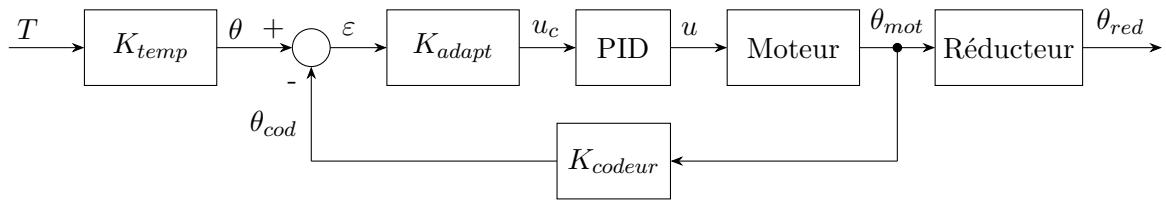


FIGURE 2.1 – Premier schéma bloc du système

Il sera nécessaire de déterminer les transferts K_{temp} et PID qui correspondent respectivement à la conversion de la température, mesurée en angle sur le cadran, et aux paramètres du correcteur PID qui permet de corriger l'erreur ε en sortie du comparateur.

2.2. Partie mécanique

Même si les parties sont présentées de façon distinctes, il est évident que mécanique et électronique ont été conçues pour s'intégrer parfaitement ensemble et tenir compte des contraintes de chacune. De plus, il en est de même pour la CAO des pièces qui ont été développées en tenant compte de la capacité de fabrication (machines et matériel disponible).

2.2.1. Dimensionnement général

Pour dimensionner correctement notre robot thermomètre, il a fallu trouver un compromis entre les contraintes imposées par le cahier des charges sur la taille du boîtier et la facilité d'utilisation.

La FIGURE 2.2 correspond à la proposition de conception sur laquelle ont été fixées des contraintes de taille à savoir ($40 \times 110\text{mm}$).

Nous tenions à conserver l'aspect transportable et compact, tout en y ajoutant des fonctions utiles, non imposées lors du lancement du projet à savoir :

- Un carter fermé et aéré qui inclut l'ensemble des composants du projet
- Une aiguille, assez fine pour lire précisément une température
- Des graduations tous les 2°C , avec l'ajout d'un bandeau de leds circulaire qui s'allument en fonction de la position de l'aiguille et dont la couleur dépend de la température.

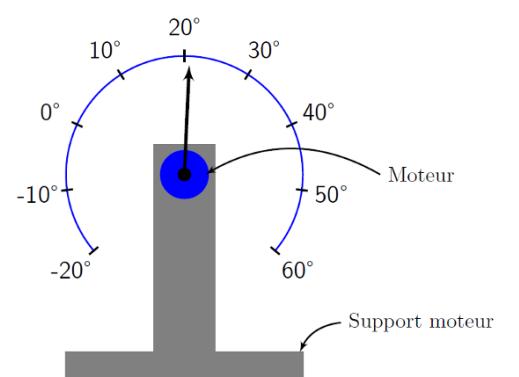


FIGURE 2.2 – Schéma du cadran

2.2.2. Conception assistée par ordinateur

Après quelques heures de modélisation sous *Fusion 360*, nous obtenons finalement une conception complète et propre. Elle regroupe d'ailleurs les liaisons mécaniques entre les pièces permettant notamment d'imaginer les mouvements possibles, et surtout de définir virtuellement des butées et de relever les angles correspondants.



FIGURE 2.3 – Capture du modèle 3D du robot thermomètre fait sous *Fusion 360* (Autodesk)

La CAO était pour nous un incontournable. Elle permet :

- un placement précis et une simulation mécanique des liaisons pour garantir un fonctionnement en dynamique
- une visualisation qui simplifie la conception sans avoir besoin de maquetter et qui permet de valider ou non un design
- un paramétrage qui permet d'avoir un modèle qui se reconstruit automatiquement en cas de changement de cote, produisant ainsi immédiatement les pièces prêtes à la fabrication
- et surtout ... l'exportation des pièces en format **.stl**, lisible par un logiciel trameur d'une imprimante 3D

Nous reviendrons dans le [Chapitre 3](#) plus en détail sur le montage et les choix d'assemblage sur les pièces imprimées en 3D.

2.3. Partie électronique

2.3.1. Carte électronique et capteur

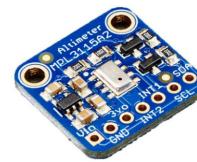
Un certain nombre de composants nous ont été fourni dans un kit lors du démarrage du projet.



L'Arduino Mega 2560 est une carte de développement basée sur le microcontrôleur ATmega2560. Elle est conçue pour des projets complexes nécessitant beaucoup d'entrées/sorties et une grande capacité mémoire.

Elle dispose de 54 broches numériques, 16 entrées analogiques, 4 ports UART, et est **cadencée sur une fréquence d'horloge à 16 MHz**.

Le MPL3115A2 est un capteur numérique barométrique capable de mesurer la pression atmosphérique, l'altitude et la température. Il communique via le protocole I2C, ce qui facilite son intégration avec des cartes comme Arduino ou Raspberry Pi.



2.3.2. Motorisation et codeur incrémental

Le moteur utilisé dans ce projet est le DC Motor/Réducteur Custom 6V Motor, spécialement conçu pour les kits de robots Digilent. Ce moteur, référence IG220053X00085R, est équipé d'un encodeur pour la détection de la rotation et de la vitesse.

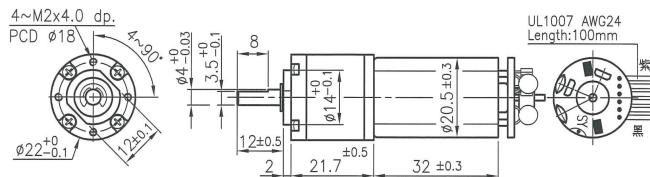


FIGURE 2.4 – Documentation technique du motoréducteur



FIGURE 2.5 – Motoréducteur utilisé

L'encodeur permet de déterminer la position. En effet, il possède un disque avec des zones de contact uniformément espacées qui sont connectées à la broche commune C et à deux autres broches de contact séparées A et B, comme illustré sur la [FIGURE 2.6](#).

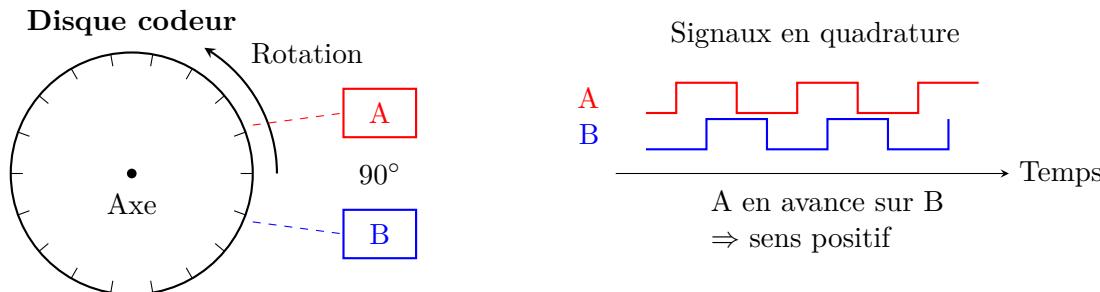


FIGURE 2.6 – Codeur incrémental : disque optique et signaux A (rouge) et B (bleu) en quadrature

Lorsque le disque commence à tourner pas à pas, les broches A et B entrent successivement en contact avec la broche commune C. Deux signaux de sortie en onde carrée sont alors générés.

Chacune des deux sorties peut être utilisée pour déterminer la position de rotation, à condition de compter les impulsions du signal. Pour déterminer le sens de rotation, les deux signaux doivent être analysés simultanément. En effet, les deux signaux sont déphasés de 90 degrés l'un par rapport à l'autre :

- Si l'encodeur tourne dans le sens des aiguilles d'une montre, la sortie A est en avance sur la sortie B.
- À l'inverse, si l'encodeur tourne dans le sens inverse des aiguilles d'une montre, la sortie B est en avance sur la sortie A.

Ainsi, en exploitant ce principe, l'encodeur permet d'identifier à la fois la variation de position (en comptant le nombre d'impulsion) et le sens de rotation (en déterminant le déphasage). Dès lors on pourra :

1. Connaître la position θ de l'aiguille, à condition de partir d'une référence
2. Récupérer, en dérivant, la vitesse de rotation ω , qui sera exploitée dans la [Section 2.4.1](#)

2.3.2.1. Pont en H

Le pont en H est nécessaire pour adapter la puissance entre la carte Arduino et le moteur, que la carte ne peut ni alimenter ni commander directement. Il permet aussi d'inverser le sens de rotation, de régler la vitesse par PWM et de protéger l'électronique contre les courants et tensions générés par le moteur.



Le moteur est alimenté par le biais d'un contrôleur moteur, de type pont en H PmodHB5. Le pont est alimenté par un bloc secteur de 5,02V

Le contrôleur moteur PmodHB5 est un module compact permettant de piloter un moteur à partir d'un microcontrôleur ou d'un FPGA via une interface Pmod^a. Il assure la commande du sens de rotation et de la vitesse, tout en faisant le lien entre la logique de commande et la puissance nécessaire au moteur.

^{a.} peripheral module interface, interface développée par Digilent

Le pinout de cette carte est donnée par le tableau suivant. On utilise toutes les broches de la carte.

Header J1 (pin 1 on the top)		
Pin	Signal	Description
1	DIR	Direction pin
2	EN	Enable pin
3	SA	Sensor A feedback pin
4	SB	Sensor B feedback pin
5	GND	Power Supply Ground
6	VCC	Positive Power Supply (3.3/5V)

Header J2 (pin 1 on the bottom)		
Pin	Signal	Description
1	SB	Sensor B feedback pin
2	SA	Sensor A feedback pin
3	GND	Power Supply Ground
4	VCC	Positive Power Supply (3.3/5V)
5	M+	Motor positive pin
6	M-	Motor negative pin

SA et SB correspondent à l'information venant du moteur qui est utilisée par la librairie Encoder. DIR et EN sont pilotés directement par des GPIO configurés en sortie.

2.3.2.2. Le signal PWM

La broche EN du driver doit recevoir les impulsions de commande dans un format PWM. Un signal Pulse Width Modulation (PWM), c'est l'équivalent numérique du signal analogique, en jouant sur la durée d'une impulsion plutôt que sa hauteur. Ainsi, les installations adéquates récupèrent la moyenne du signal, le temps d'une impulsion étant bien moins sensible que sa hauteur, cette méthode est adoptée pour le système de communication du driver au moteur.

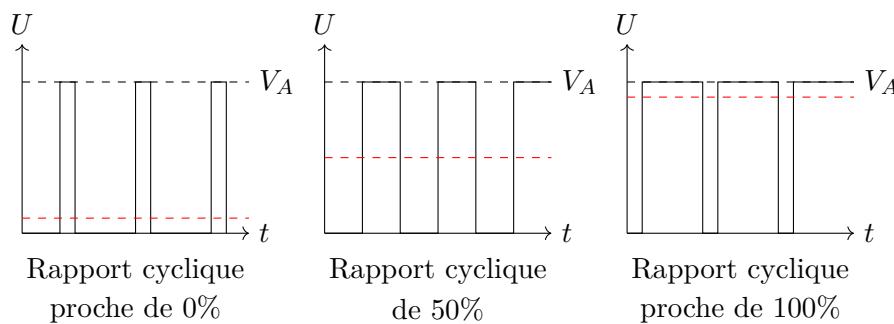


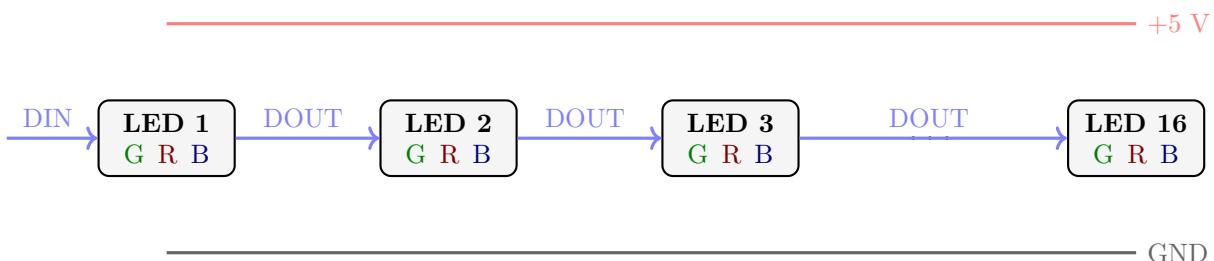
FIGURE 2.7 – Fonctionnement d'un signal PWM

En faisant varier le rapport cyclique du PWM, on ajuste l'énergie envoyée au moteur, ce qui permet un mouvement fluide, précis et stable de l'aiguille, dans la boucle d'asservissement.

2.3.3. PinOut et circuit final

Une fois les interfaces de chaque composant identifiées, on donne ci-dessous le schéma d'interconnexion final. Plusieurs critères se sont imposés selon nous en début de projet :

- Limiter l'encombrement : nous avons choisi de ne pas utiliser la breadboard donnée dans le kit, qui nous a cependant été utile pour les tests. Les câbles mâle/femelle étaient suffisants pour notre montage.
- Le choix d'implémenter un composant supplémentaire : une ringled de 16 leds qui s'allume en fonction de la position de l'aiguille.



Pour l'adapter, il a été nécessaire de reporter l'unique 5V présent sur la carte Arduino, pour alimenter la ringled et le driver.

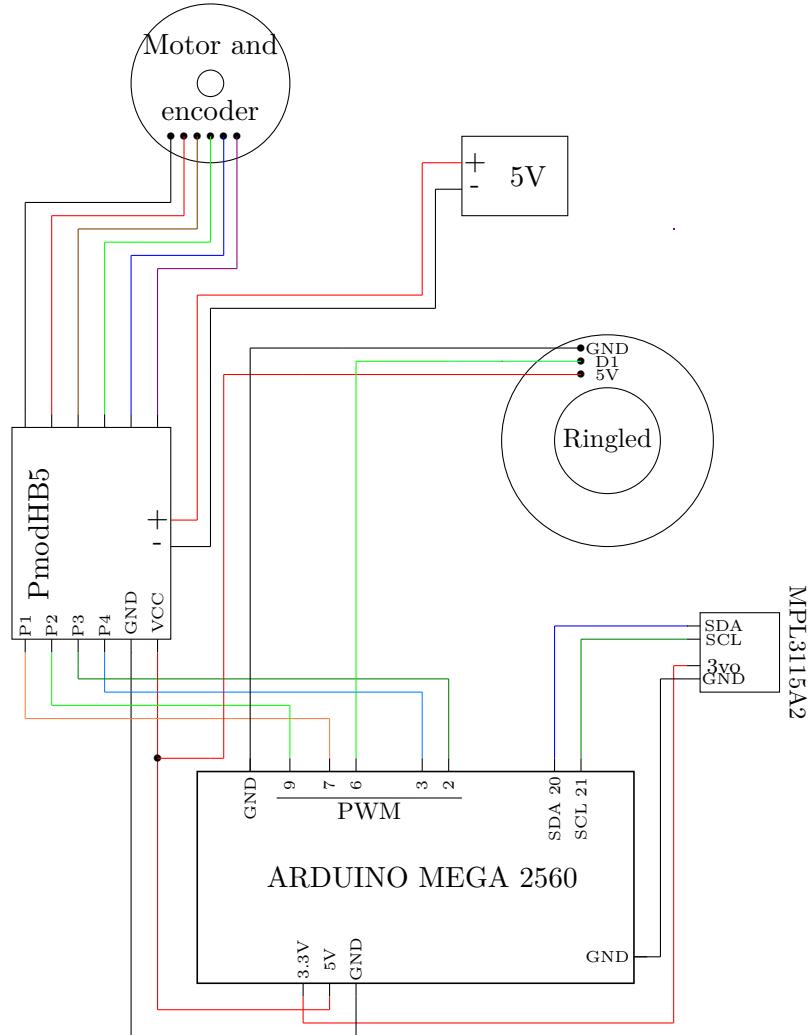


FIGURE 2.8 – Schéma du circuit électrique

2.4. Programmation

2.4.1. Calibration de l'aiguille

Pour pouvoir indiquer précisément la température avec l'aiguille, il faut une position de référence, le codeur n'étant qu'incrémental.

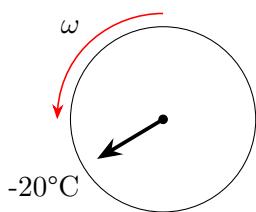


FIGURE 2.9 – Butée

Idée : On choisit de placer l'aiguille à -20°C pour la calibration. On place donc une butée mécanique en -20°C.

Réalisation : -20°C étant la plus petite valeur de température mesurable, l'aiguille doit venir en butée dans le sens inverse des aiguilles d'une montre. La détection de la butée mécanique se fait à l'aide du codeur, on a la position du moteur et donc sa vitesse. Lorsqu'une baisse de vitesse est détectée, le système considère qu'il est à -20°C et affiche ensuite la température réelle.

Contrainte : Il faut que l'aiguille arrive en butée lentement et la détecte rapidement au risque d'abîmer l'aiguille et la butée.

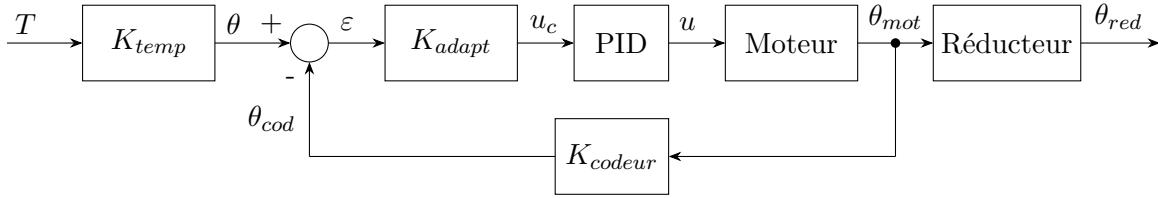
Ajustement : Au début de la calibration, on fait démarrer le moteur rapidement (durant 0.1 seconde) puis on le fait ralentir pour respecter la contrainte précédente. Cet ajustement permet d'éviter le cas où le moteur démarre trop lentement et pense donc qu'il est déjà en butée alors que ce n'est pas le cas.

Signature lumineuse : Lors de la calibration, les 16 leds clignotent en orange (deux fois par seconde).

2.4.2. Loi de commande

On utilise dans ce projet un système de contrôle en boucle fermée dit SISO (single input, single output). Il consiste à mesurer le signal de sortie θ_{mot} au moyen d'un encodeur et à comparer ce signal à θ , image de la

température mesurée, générant donc une erreur ε .



Pour faire tendre cette erreur ε vers 0, un régulateur PID est mis en place pour que la tension moteur soit modifiée pour faire tourner le rotor en modifiant θ . On donne alors les transferts suivants :

- **Capteur de température** : Il est modélisé comme un transfert qui convertit la température en angle. Ainsi, à partir de notre conception, on relève un angle de $26,7^\circ$ pour une variation de 10°C . Ainsi

$$K_{temp} = \frac{\theta(p)}{T(p)} = \frac{26,7}{10} [\text{deg}/^\circ\text{C}] \quad (2.1)$$

- **Codeur incrémental** : Dans notre modèle, on fait abstraction de la technique d'encodeur, son gain est donc unitaire, car le rapport entre l'angle de l'encodeur et celui de l'axe moteur est identique

$$K_{codeur} = 1 \quad (2.2)$$

- **Moteur** : Le transfert du moteur peut être modélisé comme deux transferts en série. L'un correspondant à la liaison entre le signal PWM et sa conversion en tension, et le deuxième qui correspond à un transfert de MCC classique.

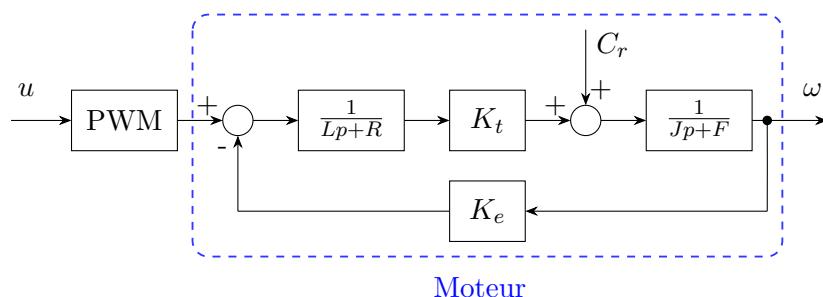


FIGURE 2.10 – Schéma bloc de l'assemblage PWM/Moteur

Pour décrire le comportement du PWM, on mesure la vitesse du moteur pour la valeur PWM la plus élevée possible, 255. En multipliant ce rapport par la constante électromotrice du moteur k_a et un facteur de conversion en rad/s, on a la fonction de transfert du bloc PWM

$$G_{PWM} = k_a \times \frac{v}{255} \times \frac{2\pi}{60} \quad (2.3)$$

A partir du modèle du second ordre fourni par la FIGURE 2.10, on peut donner la fonction de transfert

$$G_{MCC}(p) = \frac{\omega(p)}{\nu(p)} = \frac{K_t}{(Lp + R)(Jp + F) + K_t K_e} \quad (2.4)$$

- **Contrôleur PID** : On rappelle l'équation temporelle du correcteur PID qui s'écrit

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (2.5)$$

Puis, en passant dans le domaine de Laplace, on obtient

$$PID(p) = \frac{u(p)}{u_c(p)} = \frac{K_i}{p} + K_d p + K_p \quad (2.6)$$

On détaillera dans le Chapitre 3 l'implantation et la détermination des coefficients de ce correcteur.

2.4.3. Résolution de calcul et d'affichage

En tenant compte du nombre d'incrément par tour dans le codeur incrémental et le rapport de réduction du réducteur, on peut déterminer théoriquement la précision de notre thermomètre :

$$n_{capt} = \frac{\text{deg}}{\text{impulsions} \times \text{rapport reduc.}} = \frac{360}{48 \times 19} \approx 0,41^\circ \quad (2.7)$$

Le moteur pourra donc avoir un pas de $0,41^\circ$, soit, d'après la modélisation Fusion 360, un pas de température de $0,15^\circ C$.

2.4.4. Mise en place d'un timer et d'une interruption

Une interruption périodique a été mise en place afin d'assurer le contrôle en temps réel de la position de l'aiguille du thermomètre. Pour cela, le **Timer 1 (16 bits)** du microcontrôleur **ATmega2560** a été configuré en **mode CTC** (*Clear Timer on Compare Match*), permettant de déclencher une interruption à fréquence fixe et précisément maîtrisée.

Le Timer 1 fonctionne avec une fréquence d'horloge du microcontrôleur de $f = 16$ MHz. Un **prescaler de 64** a été sélectionné, ce qui conduit à une fréquence de comptage interne de 250 kHz. Le registre de comparaison **OCR1A** a été fixé à la valeur 1249, imposant une interruption toutes les

$$T = \frac{OCR1A + 1}{250\,000} = 5 \text{ ms},$$

soit une fréquence de **200 Hz**, adaptée à l'asservissement sans saccades de l'aiguille.

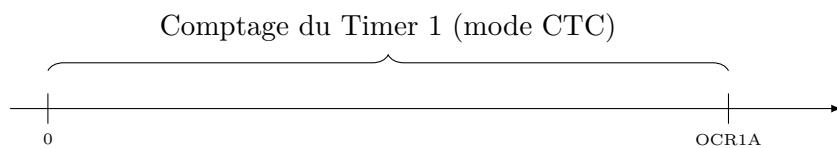


FIGURE 2.11 – Principe de fonctionnement du Timer 1 en mode CTC

À chaque incrémentation du registre **TCNT1**, le compteur est comparé à la valeur du registre **OCR1A**. Lorsque l'égalité est atteinte, le compteur est automatiquement remis à zéro et le flag de comparaison est levé, déclenchant l'exécution de la routine d'interruption **ISR(TIMER1_COMPA_vect)**.

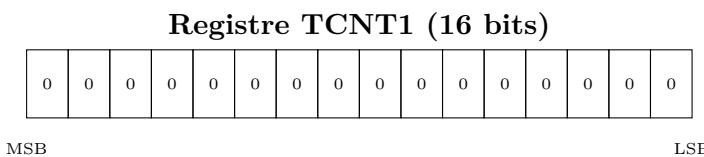


FIGURE 2.12 – Registre de comptage TCNT1

Le déclenchement de l'interruption repose sur le drapeau **OCF1A**, bit du registre **TIFR1**, positionné automatiquement lors d'une comparaison valide entre **TCNT1** et **OCR1A**.



FIGURE 2.13 – Drapeau de comparaison OCF1A du Timer 1

La configuration du mode CTC est obtenue en positionnant le bit **WGM12** à 1, les autres bits **WGM13**, **WGM11** et **WGM10** étant maintenus à 0. Le prescaler est défini par les bits **CS11** et **CS10** du registre **TCCR1B**.

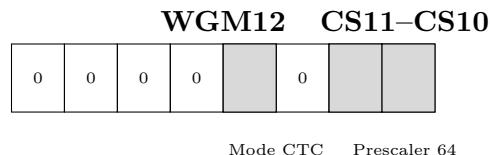


FIGURE 2.14 – Configuration du registre TCCR1B

La routine d'interruption est dédiée à la **boucle d'asservissement de la position**. À chaque appel, la position actuelle de l'aiguille est mesurée via l'encodeur incrémental, puis comparée à la position cible issue de la température mesurée. L'erreur de position est traitée par un correcteur **PID**, dont la sortie est convertie en une commande **PWM** appliquée au moteur via un pont en H.

Lorsque l'erreur devient inférieure à un seuil prédéfini, la commande moteur est annulée afin d'éviter les oscillations autour de la position cible et garantir la stabilité visuelle de l'aiguille. Certaines variables calculées dans l'interruption sont copiées dans des variables intermédiaires afin de permettre leur affichage sécurisé dans la boucle principale.

2.4.5. Organigramme de fonctionnement du code

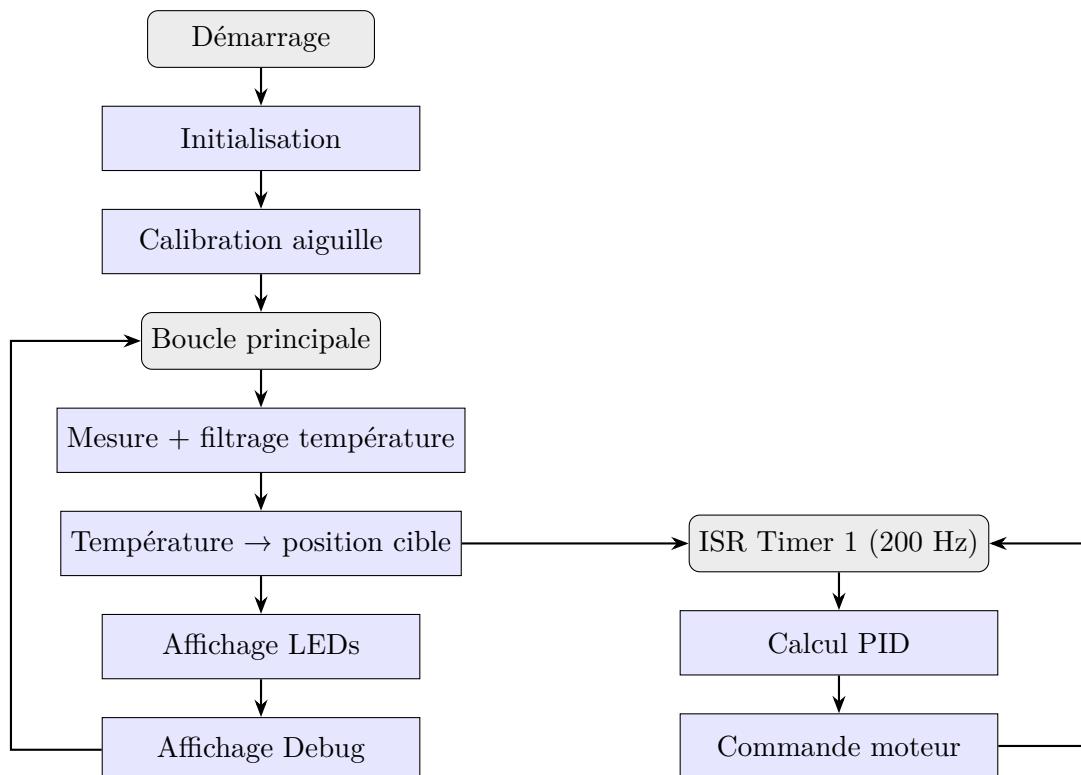


FIGURE 2.15 – Organigramme compact et lisible du thermomètre à aiguille avec PID

La partie temps réel est traitée dans l'ISR, donc la boucle principale n'a pas de timing critique, elle s'occupe uniquement de la mesure de température, qui, elle, a des constantes de temps longues.

Chapitre 3

Réalisation, développements et tests unitaires

3.1. Impression des pièces

L'impression 3D est, depuis quelques années, le moyen de prototypage le plus utilisé : peu coûteux, facile à prendre en main et totalement sur mesure.

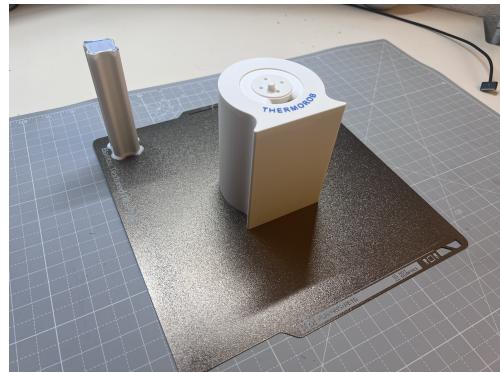
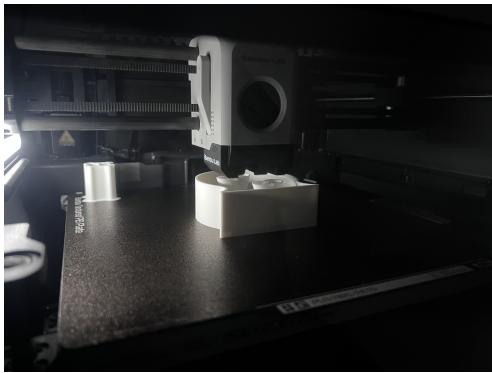


FIGURE 3.1 – Impression du boîtier sur Bambulab P1S

Plusieurs points nous semblent primordiaux pour garantir un assemblage durable, solide et esthétique



- Le PLA a été choisi pour sa facilité d'impression et sa résistance mécanique correcte par rapport à notre besoin. Nous avons pris soin de choisir des paramètres d'impression privilégiant la robustesse.
- Nous utilisons de la visserie M3 mécanique pour fermer le couvercle arrière du boîtier. Au lieu d'imprimer des filetages dans la pièce, qui ne seraient pas suffisamment précis et solide, nous décidons d'y intégrer des inserts filetés chauffés au fer à souder et noyés (visibles ci-contre).
- Nous avons prévu, sur le couvercle arrière, des passages de câble sur le modèle ainsi que des trous pour faire passer et utiliser les connecteurs de l'Arduino, une fois la carte calée dans le boîtier.

Du fait du principe retenu de mise à zéro du dispositif de mesure, il nous fallait un montage de l'aiguille en liaison encastrement avec l'axe moteur solide et durable, pour permettre la détection d'un blocage, avec butée. Pour cela, nous avons opté pour un montage sur base d'insert fileté dont la vis interne vient serrer l'axe moteur au niveau du méplat.



Le boîtier est imprimable en trois parties, le boîtier principal, le cadran et le couvercle arrière. Ce dernier est doté d'une grille en nid d'abeille qui permet une aération correcte des composants à l'intérieur. En particulier, il permet au capteur de température d'être en contact direct avec l'air ambiant extérieur.

Coût en matière plastique : Grâce au logiciel trancheur, on peut estimer, à partir du prix d'une bobine de PLA et de la longueur de fil utilisé, le coût total du boîtier (toutes pièces comprises) → 2,75€.

3.2. Implémentation du code

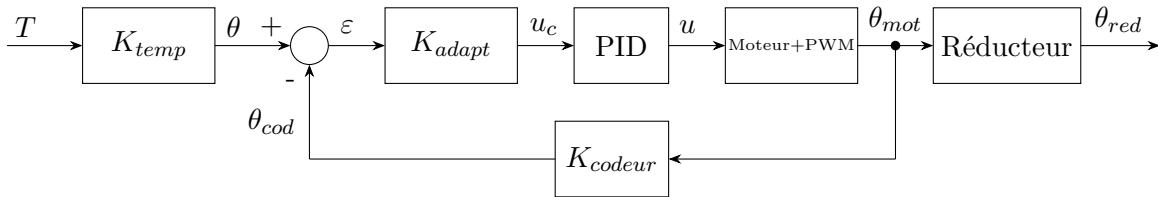
3.2.1. Librairies utilisées

Bibliothèque	Rôle dans le projet
Wire.h	Communication I ² C entre le microcontrôleur et les périphériques.
Adafruit_MPL3115A2.h	Accès au capteur MPL3115A2 pour mesurer la température ambiante.
Encoder.h	Lecture d'un encodeur incrémental en quadrature afin de mesurer précisément la position angulaire de l'aiguille.
RunningMedian.h	Filtrage médian glissant pour lisser les mesures de température et éliminer les valeurs aberrantes.
FastLED.h	Pilotage des LEDs adressables WS2812B pour l'affichage visuel de la température.

TABLE 3.1 – Bibliothèques Arduino utilisées et leur rôle

3.2.2. Réglage du PID sous Matlab

Afin de régler numériquement le PID, on implémente la fonction de transfert du système sous matlab.



La fonction de transfert du moteur asservi en position est :

$$G_{moteur} = \frac{1}{p} \times \frac{K_t}{(Lp + R)(Jp + F) + K_t K_e}$$

La fonction de transfert de la boucle de ce système peut s'écrire sous la forme :

$$T(p) = \frac{C(p)G(p)}{1 + C(p)G(p)}$$

avec $C(p)$ le PID et $G(p) = K_{adapt} \times G_{moteur} \times K_{codeur}$

3.2.2.1. Choix du correcteur

Pour un moteur à courant continu, la présence naturelle de frottements mécaniques et de la constante de force contre-électromotrice assure déjà un amortissement suffisant du système. Ainsi, l'utilisation d'un correcteur PI est généralement suffisante pour :

- assurer une erreur statique nulle,
- obtenir une bonne rapidité,

Le correcteur choisi est donc de la forme :

$$C(p) = K_p + \frac{K_i}{p}$$

3.2.2.2. Réglage automatique par pidtune

Le réglage du correcteur est réalisé automatiquement à l'aide de la fonction `pidtune` de MATLAB. Tout le code utilisé est détaillé en [ANNEXE 2](#). Cette méthode repose sur une synthèse fréquentielle garantissant :

- des marges de stabilité suffisantes
- une bande passante cohérente avec la dynamique du système
- une robustesse vis-à-vis des incertitudes du modèle

Le correcteur est obtenu par la commande :

$$C(p) = \text{pidtune}(G(p), \text{PI})$$

Argument	Rôle dans la fonction pidtune
sys	Modèle du système à asservir (fonction de transfert ou représentation d'état LTI).
type	Type de correcteur à synthétiser : 'P', 'PI' ou 'PID'.
options	Objet pidtuneOptions permettant d'ajuster la robustesse, la bande passante ou les marges de stabilité.

TABLE 3.2 – Arguments principaux de la fonction pidtune de MATLAB

3.2.2.3. Résultat du réglage du correcteur

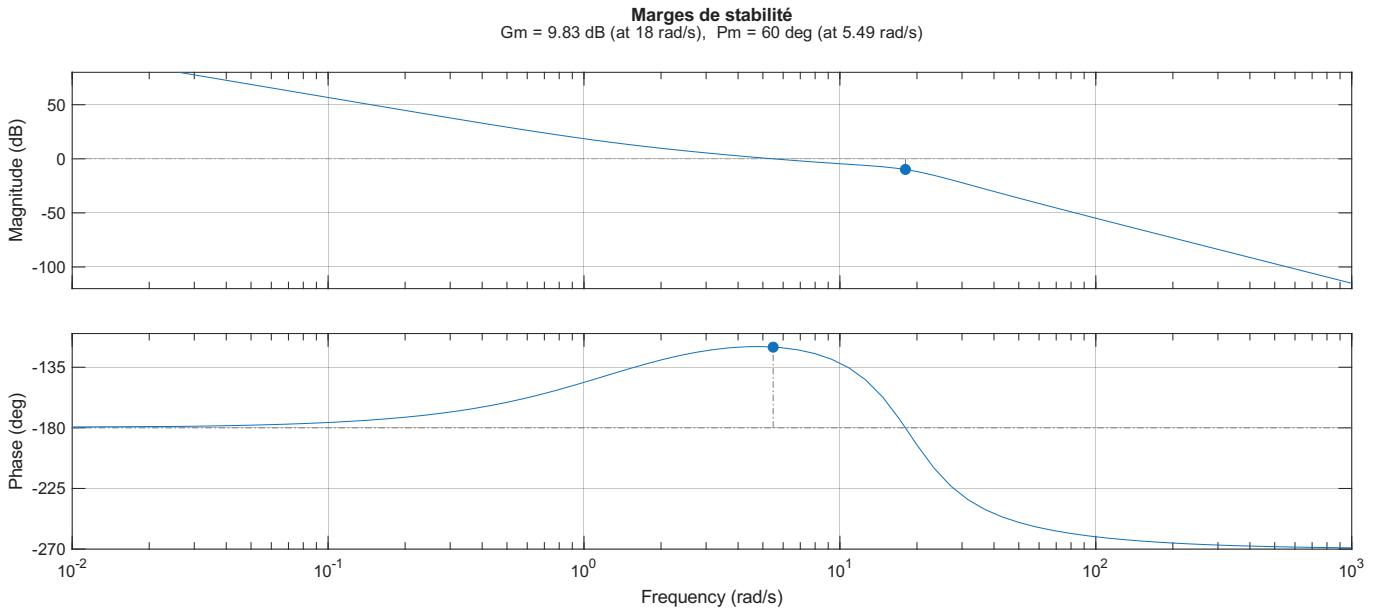
Le réglage automatique du correcteur a été effectué à l'aide de la fonction pidtune de MATLAB. Les gains obtenus sont les suivants :

$$K_p = 0,0344, \quad K_i = 0,00464, \quad K_d = 0$$

Le correcteur résultant est donc un correcteur de type **PI**, dont l'expression est :

$$C(p) = 0,0344 + \frac{0,00464}{p}$$

On trace le bode du système corrigé :



Le système est stable et robuste, avec une marge de gain de $MG = 9,83$ dB et une marge de phase de $\varphi = 60^\circ$. La marge de phase importante indique que le système est loin du risque d'instabilité et qu'il présentera une réponse transitoire bien amortie, avec peu ou pas d'oscillations. En pratique, cela signifie que le système peut supporter des variations de gain ou de paramètres sans devenir instable, offrant une stabilité confortable et une bonne fiabilité dans son fonctionnement.

On trace par la suite la réponse du système simulé à un échelon de 30° pour vérifier temps de réponse, dépassement et précision. On retrouve en vert l'échelon de consigne, en bleu la position continue de l'angle et en rouge la position angulaire discrétisée en raison du PWM.

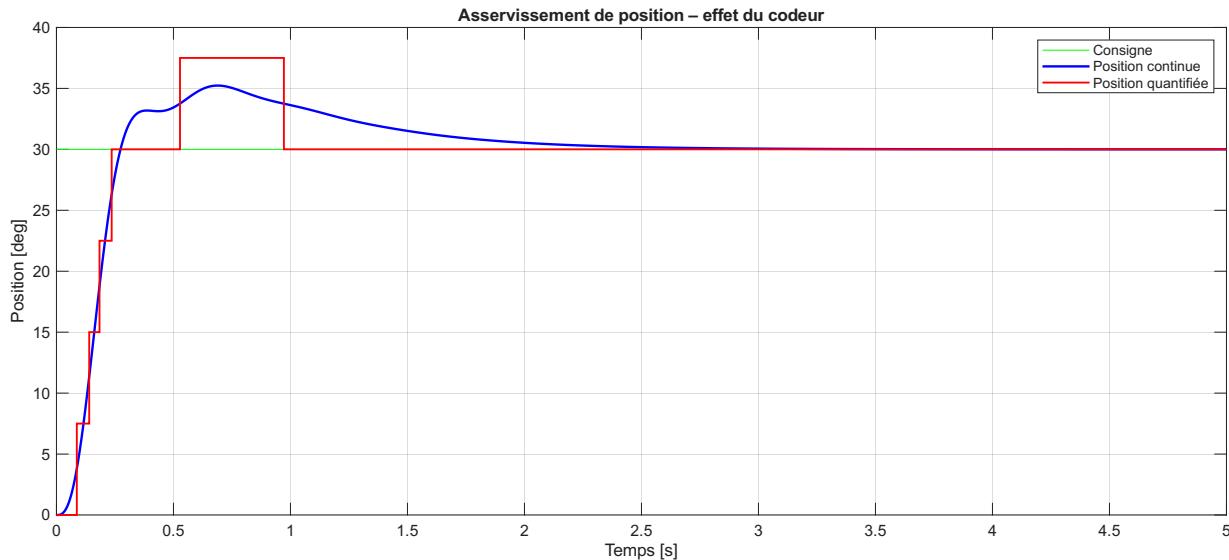


FIGURE 3.2 – Réponse à un échelon 30°

Sur cette courbe, on constate un léger dépassement lors de l'échelon. En pratique, lors de la réalisation du capteur et des tests nous avons observé un léger vas et viens unique de l'aiguille autour de la position finale. C'est cohérent avec notre modèle. D'autre part, le système est précis dans la modélisation Matlab en régime permanent et atteint la consigne avec une rapidité d'environ 2 secondes, ce qui valide l'exigence imposée par le cahier des charges.

3.3. Premier prototype

3.3.1. Tests fonctionnels hors mécanique

Avant d'attaquer le montage électronique, il a été nécessaire de tester indépendamment les composants :

- **Le capteur de température** : un affichage simple de la température mesurée par le capteur dans la console Arduino toutes les secondes
- **Le moteur** : test de rotation du moteur contrôlé par PWM sous alimentation 5V
- **La ringled** : test des leds, qui a permis de déterminer quelles leds étaient utiles pour l'affichage de la température (10 leds utiles)

3.3.2. Erreurs de conception

Nous avons imprimé une première version de notre boîtier. Plusieurs erreurs sont apparues et ont dû être corrigées dans la V2.



FIGURE 3.3 – Première version du boîtier, avec butée et sans guide

- La première version comportait une butée proche de l'axe. D'une part il s'avère que ce choix n'a, d'un point de vue bras de levier, pas du tout été judicieux, et en plus était obstacle à la tête de vis qui maintient l'aiguille, celle-ci étant la seule pièce de quincaillerie qui n'a pas été modélisée dans la 3D. C'était une erreur ...
- L'axe moteur n'est pas la seule pièce tournante apparente du moteur. Il est relié à un plateau visible et affleurant le moyeu oblong. Dans la conception il était prévu qu'il s'appuie sur le bâti. Résultat : axe bloqué après serrage du moteur dans le bâti et le logiciel a considéré que la butée était directement atteinte. L'aiguille est donc partie à pleine vitesse en rotation horaire vers la position voulue, et a cassé la butée.
- Les jeux initialement prévus (0,3 mm) s'avèrent insuffisants. Par exemple, la ringled n'est pas rentrée dans la feuillure prévue.

3.4. Intégration définitive

Suite à l'impression de la V2 du boîtier (modification du design et de l'emplacement de la butée mécanique, ajout d'un guide pour le positionnement du cadran et quelques corrections de dimensionnement), nous avons effectué le montage du prototype final. Le montage est décomposé en 4 étapes :

- Fixation du **cadran**, avec du ruban adhésif double-face, et de la **ringled** avec de la colle chaude.
- Installation du **moteur** et de l'**aiguille**
- Branchement et mise en place des **composants électroniques** dans le boîtier
- Fermeture du **boîtier** avec le **couvercle arrière**

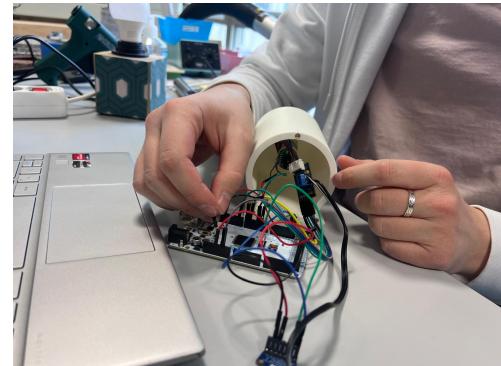


FIGURE 3.4 – Fixation du cadran et mise en place des composants électroniques dans le boîtier

Visuels finaux du prototype de robot thermomètre "Thermorob". Caractéristiques :

- Boîtier compact et léger (230g)
- Affichage analogique de la température avec une aiguille et un dégradé de couleurs (bleu → vert → rouge)
- Deux câbles d'alimentation (moteur et carte Arduino)



FIGURE 3.5 – Le Thermorob (vue de face et vue arrière)



FIGURE 3.6 – Le Thermorob en fonctionnement et chauffé

Conclusion

4.1. Résultats

Pour prouver le bon fonctionnement de notre boîtier, on décide de le soumettre à un test permettant de juger de la précision de la mesure. On part de la température ambiante et on monte en température avec un sèche-cheveux. Toutes les 5 secondes, on relève le couple ($T_{capteur}/T_{affichage}$).

On intègre au graphique ci dessous l'erreur de mesure pour chaque couple de valeur.

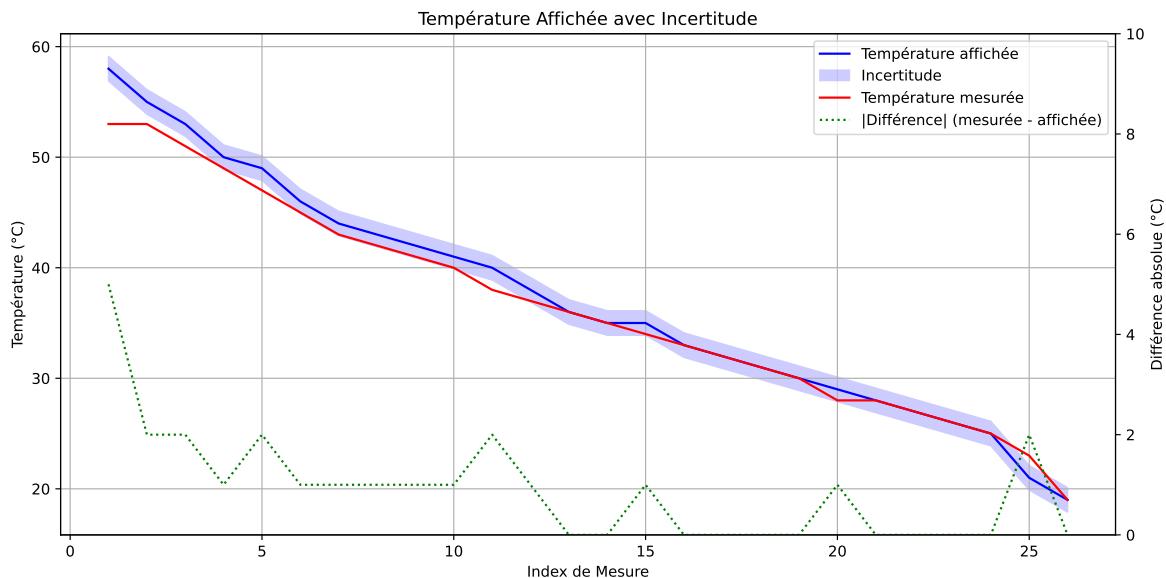


FIGURE 4.1 – Tracé des couples de température et incertitude associée

Etant donné que la mesure sur le cadran se fait avec une précision de lecture maximale à $1^{\circ}C$ près, l'écart de température est soit de 0, soit de $1^{\circ}C$, ce qui est acceptable pour ce capteur. D'autant plus que, sur notre petit cadran de 60mm de diamètre, un degré de température correspond à $1,3^{\circ}$ d'angle. En stabilisant la température, l'écart tend rapidement (2s) vers 0 pour atteindre la température exacte mesurée par le capteur.

4.2. Conclusion

Ce projet Protorob s'est avéré être un très bon moyen pour tous les membres du groupe de se tester sur un projet sur une courte durée et aborder, renforcer, et appliquer leurs compétences en conception mécanique, électronique, contrôle commande et programmation (timer, interruption et protocole I^2C). Le robot fabriqué est d'une qualité de finition très satisfaisante et s'avère être très efficace.

D'un point de vue organisationnel, ce travail a été l'occasion de travailler en équipe et de partager nos connaissances à travers ce projet. Le travail au sein de l'équipe fut un réel plaisir au fur et à mesure des séances et les progrès de chacun ont très vite été constatés.

Annexes

Annexe 1 – Code Arduino du ProtoRob

```

1
2 #include <Wire.h>
3 #include <Adafruit_MPL3115A2.h>
4 #include <Encoder.h>
5 #include <RunningMedian.h>
6 #include <FastLED.h>
7
8 // ===== PINS =====
9 #define PIN_PWM 9
10 #define PIN_DIR 7
11 #define ENC_A 2
12 #define ENC_B 3
13
14 #define LED_PIN 6
15 #define NUM_LEDS_TOTAL 16 // LEDs physiques
16 #define NUM_LEDS_TEMP 10 // affichage temperature
17 #define LED_TYPE WS2812B
18 #define COLOR_ORDER GRB
19 #define LED_OFFSET 3 // decalage horaire
20
21 CRGB leds[NUM_LEDS_TOTAL];
22
23 // ===== SYSTEME MECANIQUE =====
24 #define ENCODER_PPR 12 // nb de fentes dans l'encodeur par tour
25 #define QUAD_FACTOR 3 // fronts montants et descendants
26 #define GEAR_RATIO 19.0
27 #define NEEDLE_DEG 210.0 // amplitude en deg entre la position max et min
28
29 #define TICKS_PER_MOTOR_REV (ENCODER_PPR * QUAD_FACTOR)
30 #define TICKS_PER_OUTPUT_REV (TICKS_PER_MOTOR_REV * GEAR_RATIO)
31 #define TICKS_MAX (long)(TICKS_PER_OUTPUT_REV * (NEEDLE_DEG / 360.0))
32 #define OFFSET -6 //TICKS
33
34 // ===== TEMP =====
35 #define TEMP_LIMIT_MIN -20.0
36 #define TEMP_MAX 60.0
37
38 // ===== OBJETS =====
39 Adafruit_MPL3115A2 tempSensor;
40 Encoder encoder(ENC_A, ENC_B);
41 RunningMedian samples(10);
42
43 // ===== PID =====
44 volatile float Kp = 0.03; //2.0;
45 volatile float Ki = 0.004; //0.006;
46 volatile float Kd = 0.01; //10.0; //impact du Kd negligable
47
48 volatile long targetPosition = 0;
49 volatile long currentPosition = 0;
50
51 volatile float error = 0;
52 volatile float prevError = 0;
53 volatile float integral = 0;
54 volatile float output = 0;
55
56 volatile float error_isr = 0;
57 volatile float output_isr = 0;
58
59 #define PWM_MAX 255
60 #define PWM_MIN 60 // vitesse minimale pour garantir la rotation

```

```

61 #define INTEGRAL_LIMIT 2000
62
63 // ===== ROUTINE D'INTERRUPTION (200 Hz) =====
64 ISR(TIMER1_COMPA_vect) {
65
66     currentPosition = -encoder.read();
67     error = targetPosition - currentPosition;
68
69     integral += error;
70     integral = constrain(integral, -INTEGRAL_LIMIT, INTEGRAL_LIMIT); // partie
71     // integrale bornee
72
73     float derivative = error - prevError; // calcul de
74     // variation de l'erreur
75     prevError = error;
76
77     output = Kp * error + Ki * integral + Kd * derivative; // calcul du PID
78     output = constrain(output, -PWM_MAX, PWM_MAX); // sortie bornee
79
80     error_isr = error;
81     output_isr = output;
82
83     if (abs(error) < 2) {
84         analogWrite(PIN_PWM, 0); // si l'erreur de position est suffisemment
85         // faible, on arrete le moteur
86     } else {
87         int pwm = abs(output);
88         pwm = constrain(pwm, PWM_MIN, PWM_MAX);
89
90         digitalWrite(PIN_DIR, (error > 0) ? LOW : HIGH); // si l'erreur est
91         // positive, on tourne dans un sens
92         analogWrite(PIN_PWM, pwm); // sinon dans l'autre
93     }
94 }
95
96 // ===== LED CALIBRATION =====
97 void blinkCalibrationLeds(bool state) { // fait clignoter les leds
98     CRGB color = state ? CRGB::Orange : CRGB::Black;
99     for (int i = 0; i < NUM_LEDS_TOTAL; i++) {
100         leds[i] = color;
101     }
102     FastLED.show();
103 }
104
105 // ===== CALIBRATION =====
106 void calibrateNeedle() {
107
108     Serial.println("== CALIBRATION EN COURS ==");
109
110     long lastPosition = encoder.read();
111     long current;
112     int stableCount = 0;
113     bool ledState = false;
114     unsigned long lastBlink = 0;
115
116     digitalWrite(PIN_DIR, HIGH); // on fait tourner l'aiguille vers la gauche
117     // pour aller sur la butee
118
119     analogWrite(PIN_PWM, 60); // initialement : moteur -> vitesse min
120     // normale
121     delay(130);
122     analogWrite(PIN_PWM, 30); // on diminue pour se mettre au plus

```

```

    lent et ne pas casser la butee

117
118 while (stableCount < 3) {
119
120     if (millis() - lastBlink > 250) { // on change l'etat des leds toutes les
121         250ms
122         lastBlink = millis();
123         ledState = !ledState;
124         blinkCalibrationLeds(ledState);
125     }
126
127     delay(20);
128     current = encoder.read();
129     long delta = abs(current - lastPosition);
130     lastPosition = current;
131
132     if (delta <= 1) stableCount++; // l'aiguille est bloquee 3 fois de
133     suite pour sortir de la boucle while
134     else stableCount = 0;
135 }
136
137
138     digitalWrite(PIN_PWM, 0); // arret du moteur
139     encoder.write(0 + OFFSET); // initialisation du codeur
140
141     FastLED.clear(); // on eteint les leds
142     FastLED.show(); // application du clear
143
144     integral = 0;
145     prevError = 0;
146     targetPosition = 0;
147
148     Serial.println("== CALIBRATION OK ==");
149 }
150
151 // ===== LED TEMPERATURE =====
152 void updateTemperatureRing(float temperature) {
153
154     temperature = constrain(temperature, TEMP_LIMIT_MIN, TEMP_MAX);
155
156     int activeLeds = map(
157         (int)temperature,
158         (int)TEMP_LIMIT_MIN,
159         (int)TEMP_MAX,
160         1,
161         NUM_LEDS_TEMP
162     );
163     activeLeds = constrain(activeLeds, 1, NUM_LEDS_TEMP);
164
165     FastLED.clear();
166
167     // --- ALLUMAGE DU DEGRADE ---
168     for (int i = 0; i < activeLeds; i++) {
169
170         int ledIndex = (LED_OFFSET + i) % NUM_LEDS_TOTAL;
171
172         uint8_t hue = map(i, 0, NUM_LEDS_TEMP - 1, 160, 0); // bleu -> rouge
173         leds[ledIndex] = CHSV(hue, 255, 255);
174     }
175
176     // --- EXTINCTION DES 6 LEDS RESTANTES (ANTI-HORAIRES) ---
177     for (int i = NUM_LEDS_TEMP; i < NUM_LEDS_TOTAL; i++) {
178         int ledIndex = (LED_OFFSET + i) % NUM_LEDS_TOTAL;
179         leds[ledIndex] = CRGB::Black;

```

```

177 }
178
179 FastLED.show();
180 }
181
182 // ====== TIMER SETUP ======
183 void setupTimer1() {                                     // setup timer 1 : 16 bits
184   cli();                                              // desactive l'interruption globale
185   TCCR1A = 0;                                         // registres de controle
186   TCCR1B = 0;
187   OCR1A = 1249;                                       // comparaison des sorties sur timer 1
188   TCCR1B |= (1 << WGM12);                           // activation du mode CTC
189   TCCR1B |= (1 << CS11) | (1 << CS10);             // reglage du prescaler
190   TIMSK1 |= (1 << OCIE1A);                          // flag d'interruption
191   sei();                                              // mise en route des interruptions
192 }
193
194 // ====== SETUP ======
195 void setup() {
196
197   Serial.begin(115200);
198   Wire.begin();
199
200   pinMode(PIN_PWM, OUTPUT);                            // pins de controle du pont en H
201   pinMode(PIN_DIR, OUTPUT);
202
203   FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS_TOTAL);
204   FastLED.setBrightness(10);                           // gestion de la luminosite des leds
205   FastLED.clear();
206   FastLED.show();
207
208   if (!tempSensor.begin()) {                         // initialisation de la com avec le
209     capteur de temp
210     Serial.println("ERREUR CAPTEUR !");
211     while (1);
212   }
213
214   calibrateNeedle();                               // recherche de la butee
215   setupTimer1();                                   // demarrage du timer
216 }
217
218 // ====== LOOP ======
219 void loop() {
220
221   float temp = tempSensor.getTemperature(); // mesure de la temperature
222   samples.add(temp);                         // stockage dans la variable du PID
223
224   float temperature = samples.getMedian(); // lissage pour eviter les glitches
225   temperature = constrain(temperature, TEMP_LIMIT_MIN, TEMP_MAX);
226
227   targetPosition = map(          // definition des correspondances entre deux
228     int)temperature,           // intervalles
229     int)TEMP_LIMIT_MIN,         // intervalle de temperatures
230     int)TEMP_MAX,              // intervalle des angles
231     0,                         // TICKS_MAX
232   );
233   targetPosition = constrain(targetPosition, 0, TICKS_MAX); // limitation dans
234   un intervalle realiste
235
236   updateTemperatureRing(temperature);           // par securite
237                                                 // materielle
238                                                 // allumage des leds

```

```
correspondantes

236
237 // ----- PRINT TEMP TOUTES LES 2 SECONDES -----
238 static unsigned long lastPrint = 0;
239
240 if (millis() - lastPrint >= 2000) {                                // affichage de debug
241     sur console Arduino
242     lastPrint = millis();
243     Serial.print("Temperature mesuree : ");
244     Serial.print(temperature);
245     Serial.println("deg C");
246 }
247 // -----
248 delay(50);
249 }
```

Listing 1 – Code Arduino du thermomètre à aiguille

Annexe 2 – Code Matlab du projet

```

1 %% =====
2 % INITIALISATION
3 %% =====
4 close all;
5 clear;
6 clc;
7
8 %% =====
9 % PARAMETRES DU MCC
10 %% =====
11 L = 0.5; % Inductance [H]
12 R = 3.4; % Resistance [Ohm]
13 J = 0.0007; % Inertie [kg.m^2]
14 F = 0.007; % Frottement visqueux [N.m.s]
15 K_t = 0.3129; % Constante de couple [N.m/A]
16 K_e = 0.3129; % Constante FEM [V/(rad/s)]
17
18 %% =====
19 % GAINS DU SCHEMA-BLOCS
20 %% =====
21 K_temp = 2.67; % K1 (deg/ C )
22 K_adapt = 1; % K2
23 K_red = 1; % Reducteur
24
25 %% =====
26 % CODEUR
27 %% =====
28 K_codeur = 180/pi; % conversion rad -> deg (CONTINU)
29 Delta_theta = 360/48; % resolution du codeur (deg)
30
31 %% =====
32 % PWM
33 %% =====
34 Umax = 24; % tension max [V]
35 Umin = -24;
36
37 %% =====
38 % VARIABLE DE LAPLACE
39 %% =====
40 s = tf('s');
41
42 %% =====
43 % MODELE DU MOTEUR (POSITION)
44 % _mot (s) / U(s)
45 %% =====
46 G_mot = K_t / ( s*((L*s + R)*(J*s + F) + K_t*K_e) );
47
48 %% =====
49 % PLANTE VUE PAR LE CORRECTEUR
50 % _mes (s) / U(s)
51 %% =====
52 G_eq = K_adapt * G_mot * K_codeur;
53
54 %% =====
55 % REGLAGE AUTOMATIQUE DU PI
56 %% =====
57 C = pidtune(G_eq, 'PI');
58
59 Kp = C.Kp
60 Ki = C.Ki

```

```

61 Kd = C.Kd      % = 0
62
63 %% =====
64 % BOUCLE FERMEE CONTINUE ( ID ALE )
65 %% =====
66 sys_cl = feedback(C * G_eq, 1);
67
68 %% =====
69 % SIMULATION TEMPORELLE REALISTE
70 %% =====
71 t = 0:0.001:5;           % temps
72 theta_ref = 30*ones(size(t));    % consigne [deg]
73
74 % Sortie ideale continue
75 theta_cont = lsim(sys_cl, theta_ref, t);
76
77 % Quantification codeur
78 theta_quant = Delta_theta * round(theta_cont / Delta_theta);
79
80 %% =====
81 % AFFICHAGE
82 %% =====
83 figure;
84 plot(t, theta_ref, 'g'); hold on;
85 plot(t, theta_cont, 'b', 'LineWidth', 1.5); hold on;
86 stairs(t, theta_quant, 'r', 'LineWidth', 1.2);
87 grid on;
88 xlabel('Temps [s]');
89 ylabel('Position [deg]');
90 title('Asservissement de position effet du codeur');
91 legend('Consigne', 'Position continue', 'Position quantifi e');
92
93 %% =====
94 % ANALYSE FREQUENTIELLE
95 %% =====
96 figure;
97 margin(C * G_eq)
98 grid on;
99 title('Marges de stabilit ');

```

Listing 2 – Code MatLab pour la simulation du système

Bibliographie

- [1] Description du capteur adafruit de température, pression... pour la maquette. Disponible sur : <https://www.adafruit.com/product/1893>
- [2] Documentation du moto-réducteur : https://digilent.com/reference/motor_gearbox/motor_gearbox
- [3] Description du pont en H (driver) pour piloter par PWM le moteur Digilent : <https://digilent.com/reference/pmod/pmodhb5/start>
- [4] Fonctionnement codeur incrémental et librairie arduino <https://www.arduinolibraries.info/libraries/encoder>

OUR WORLDWIDE PARTNERS UNIVERSITIES - DOUBLE DEGREE AGREEMENTS



3 CAMPUS, 1 SITE



IMT Atlantique Bretagne-Pays de la Loire – <http://www.imt-atlantique.fr/>

Campus de Brest

Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes

4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes

2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

Site de Toulouse

10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
T +33 (0)5 61 33 83 65



IMT Atlantique

Bretagne-Pays de la Loire
École Mines-Télécom