



**IMT Atlantique**  
Bretagne-Pays de la Loire  
École Mines-Télécom

# Soutenance - Projet Protorob

Respo. TAF : F. Claveau

Auteurs : Grégory Boursin, Loïc Fournier,  
Cédric Léger, Emilien Wolff

Groupe 1 PROTOROB  
version 1.0  
IMT Atlantique  
22/01/26

# SOMMAIRE

1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion

1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion

Prototypage des systèmes robotisés → fabriquer une robot thermomètre

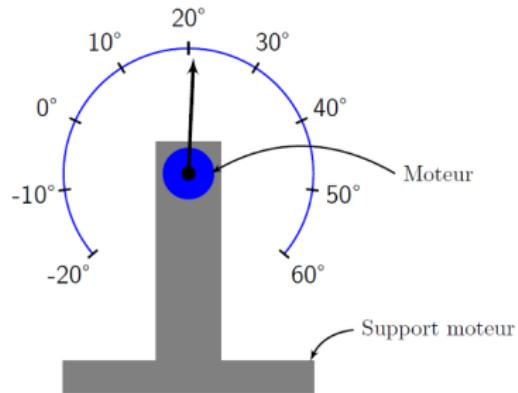


Figure 1 – Schéma du cadran proposé

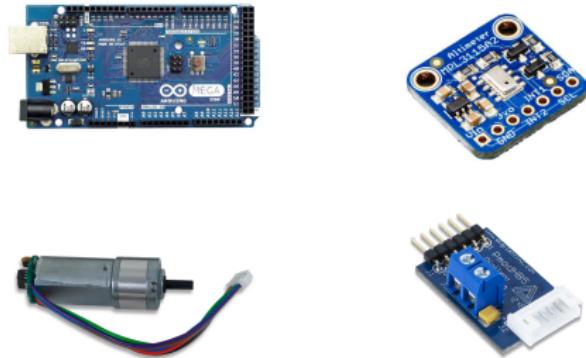


Table 1 – Composants électroniques fournis

## Objectif

Conceptualiser un robot thermomètre qui remplit la fonction suivante :

**Afficher précisément la température sur un cadran circulaire.**

# Présentation du cahier des charges

6

Fonction	Critère(s)	Niveau(x)	Flexibilité	Validation
<b>FP1 – Affichage précis de la température avec l'aiguille</b>	Précision	-20 à 60°C	±1°C	1
	Rapidité	2s	NON	
	Stabilité	Aucune oscillation en régime permanent	NON	
<b>FP2 – Transportabilité</b>	Masse	< 500g	Facile à transporter	1
	Dimensions du boîtier	Longueur < 10 cm ; Largeur < 10 cm ; Hauteur < 10 cm	NON	
<b>FC1 – Modélisation et impression 3D</b>	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	

# Présentation du cahier des charges

7

Fonction	Critère(s)	Niveau(x)	Flexibilité	Priorité
<b>FS1 – Affichage de la température avec la ringled</b>	Affichage	Leds allumées entre -20°C et la température mesurée	±5°C	2
	Signature lumineuse	Dégradé de couleurs bleu(=−20°C) → vert(=20°C) → rouge(=60°C)		
	Consommation électrique	Faible intensité lumineuse		

1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion

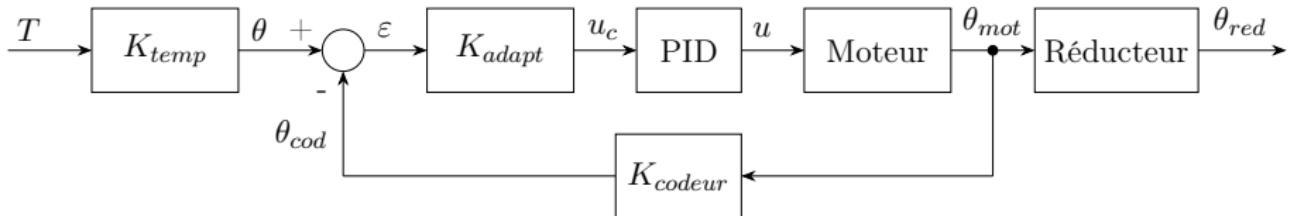


Figure 2 – Premier schéma bloc du système

Il sera donc question de s'intéresser tout particulièrement aux transferts

- ▶  $K_{temp}$  fonction de conversion  $T \rightarrow \theta$
- ▶ PID transfert du correcteur

Idées de conception (brainstorming de début de projet)

- Un **carter fermé** et aéré qui inclue l'ensemble des composants du projet
- Une **aiguille**, assez fine pour lire précisément une température
- **Graduations** et leds

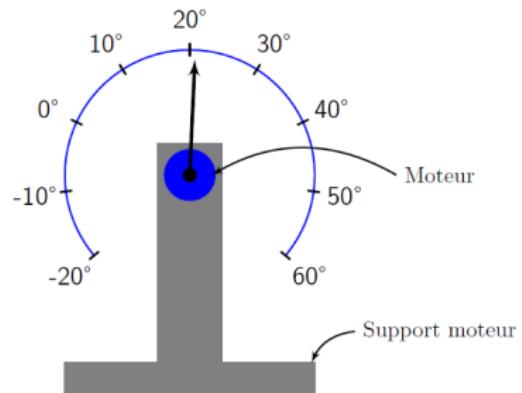


Figure 3 – Schéma du cadran donné dans les contraires projet



Figure 4 – Capture Fusion 360

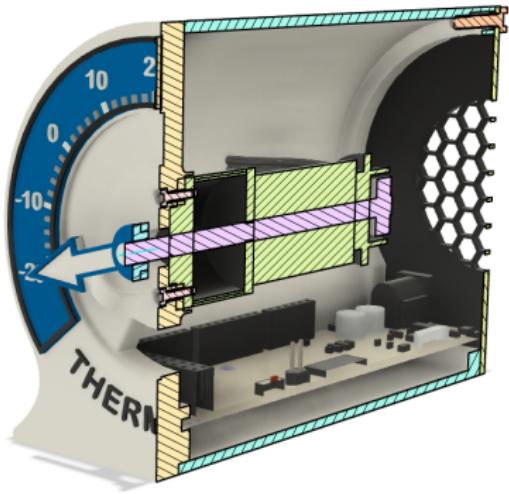
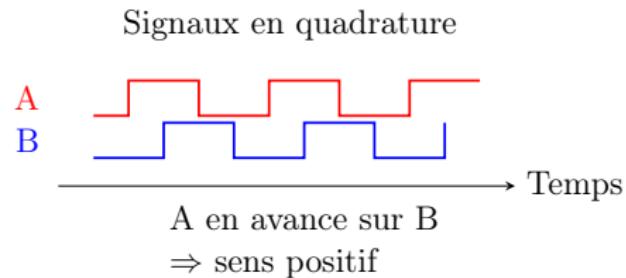
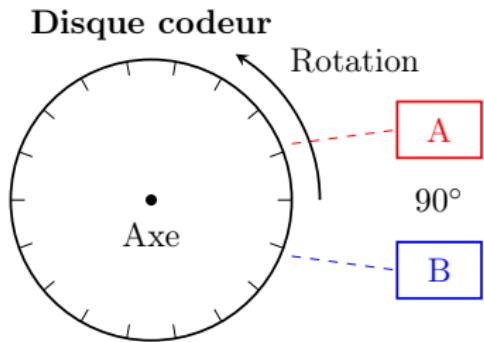


Figure 5 – Vue de coupe

Moteur DC Motor/Réducteur Custom 6V Motor équipé d'un codeur incrémental



1. Connaître la position  $\theta$  de l'aiguille
2. Récupérer, en dérivant, la vitesse de rotation  $\omega$

Commande à envoyer au pont en H Pmod HB5

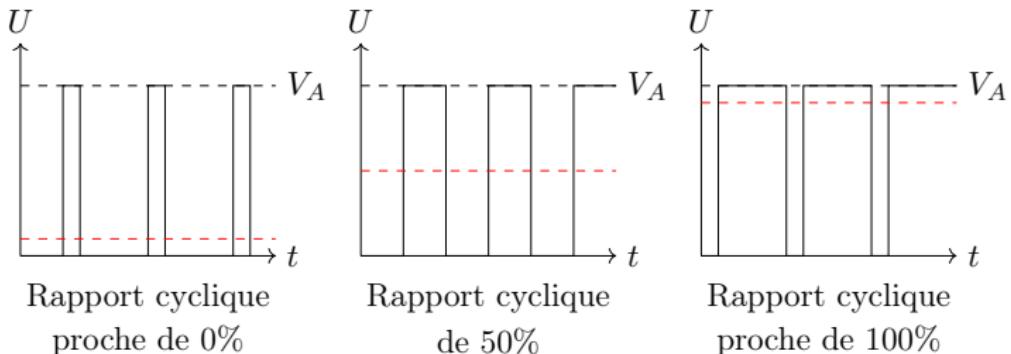


Figure 6 – Fonctionnement d'un signal PWM

En faisant varier le rapport cyclique du PWM → on ajuste l'énergie envoyée au moteur

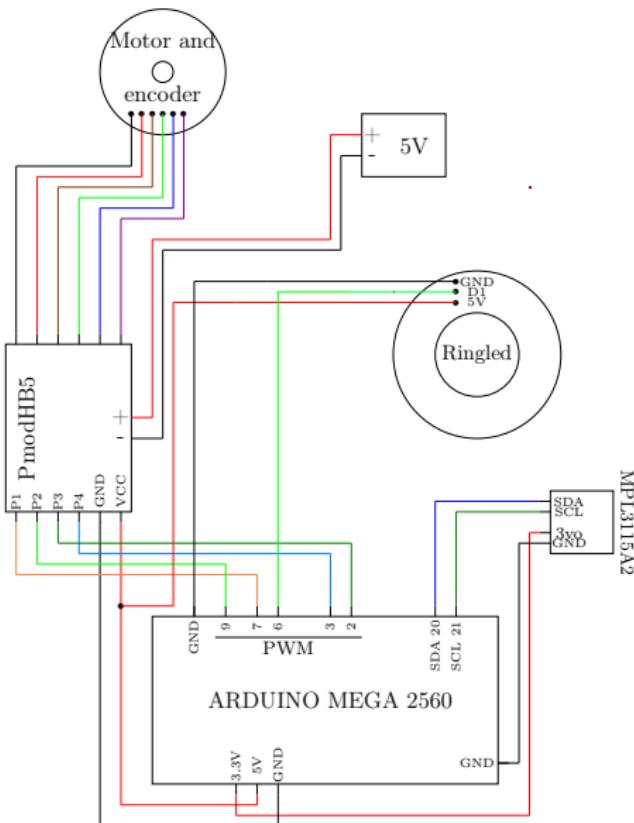


Figure 7 – Schéma du circuit électrique

## La calibration

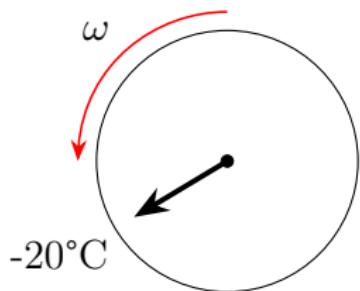
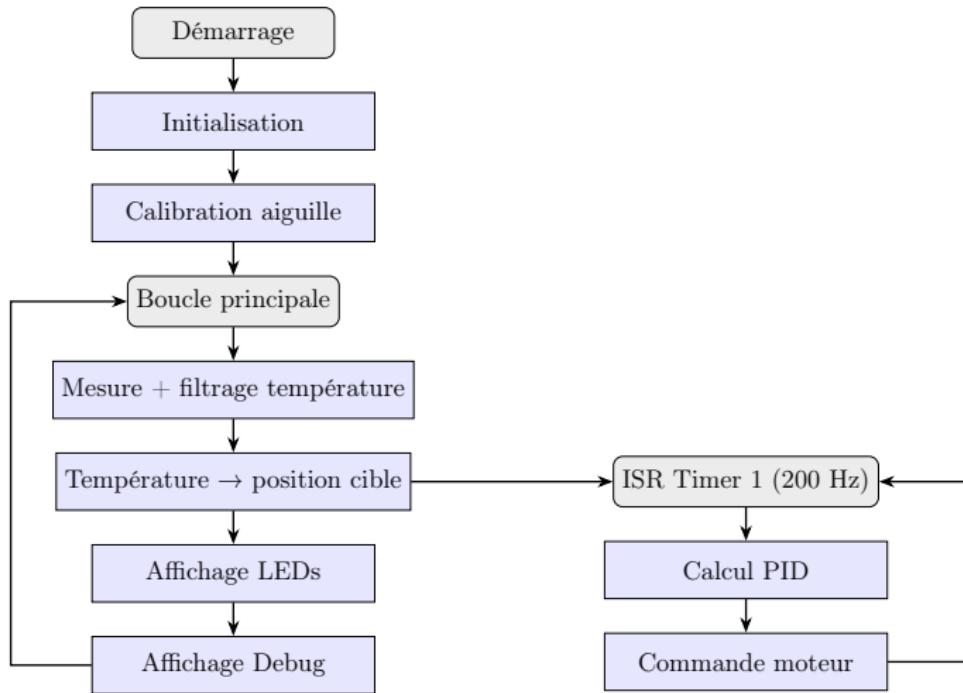


Figure 8 – Butée

**Idée :** On choisit de placer l'aiguille à  $-20^{\circ}\text{C}$  pour la calibration. On place donc une butée mécanique en  $-20^{\circ}\text{C}$ .

⇒ On détecte une baisse de vitesse de rotation  $\omega$



1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion

## Realisation des pièces en PLA

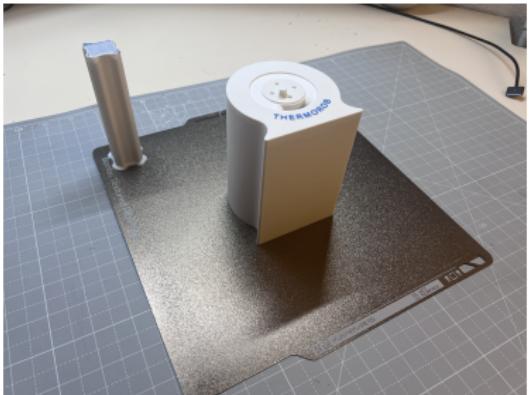
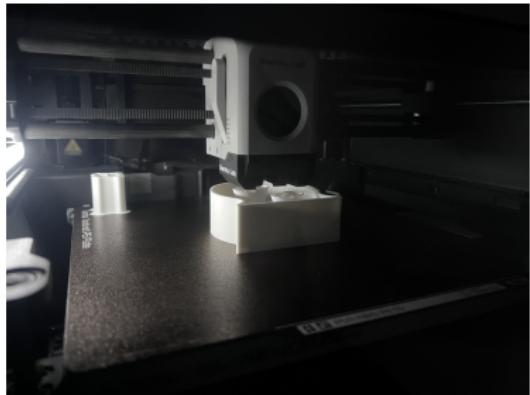


Figure 9 – Impression du boîtier sur Bambulab P1S

**Points de vigilance :** remplissage correct, épaisseurs de parois, impression à la verticale pour permettre le multi-couleurs

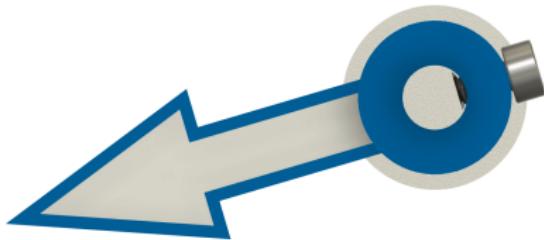


Figure 10 – Aiguille modélisée

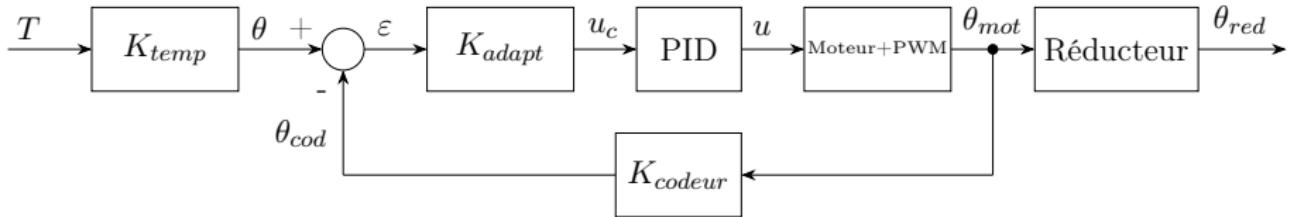
Coût en matière plastique → 2,75€.

## Choses à revoir :

- Emplacement et design de la butée
- Frottement au niveau de l'axe
- Jeux fonctionnels
- Centrage du cadran nécessaire



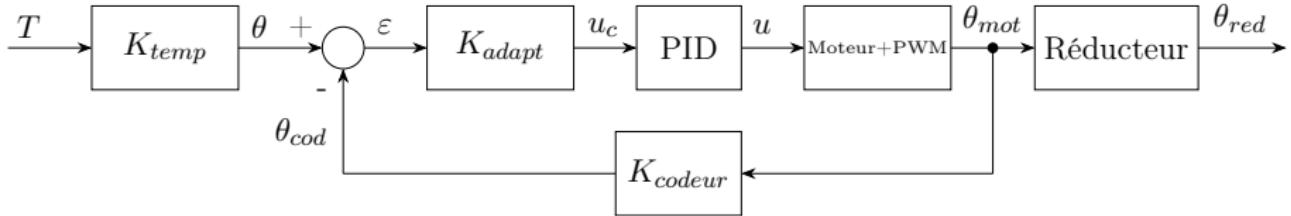
Figure 11 – V1



Choix d'un Correcteur PI :

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

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



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 s'écrit sous la forme :

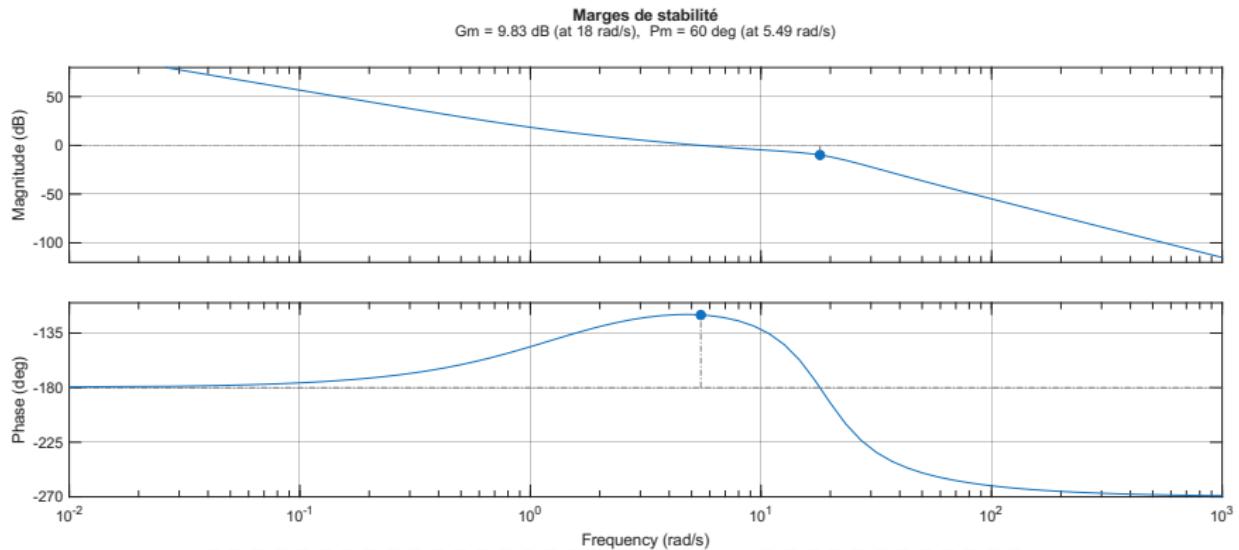
$$T(p) = \frac{C(p)G(p)}{1 + C(p)G(p)}$$

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

Le correcteur est obtenu par la commande :

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

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



# Implémentation d'un correcteur

24

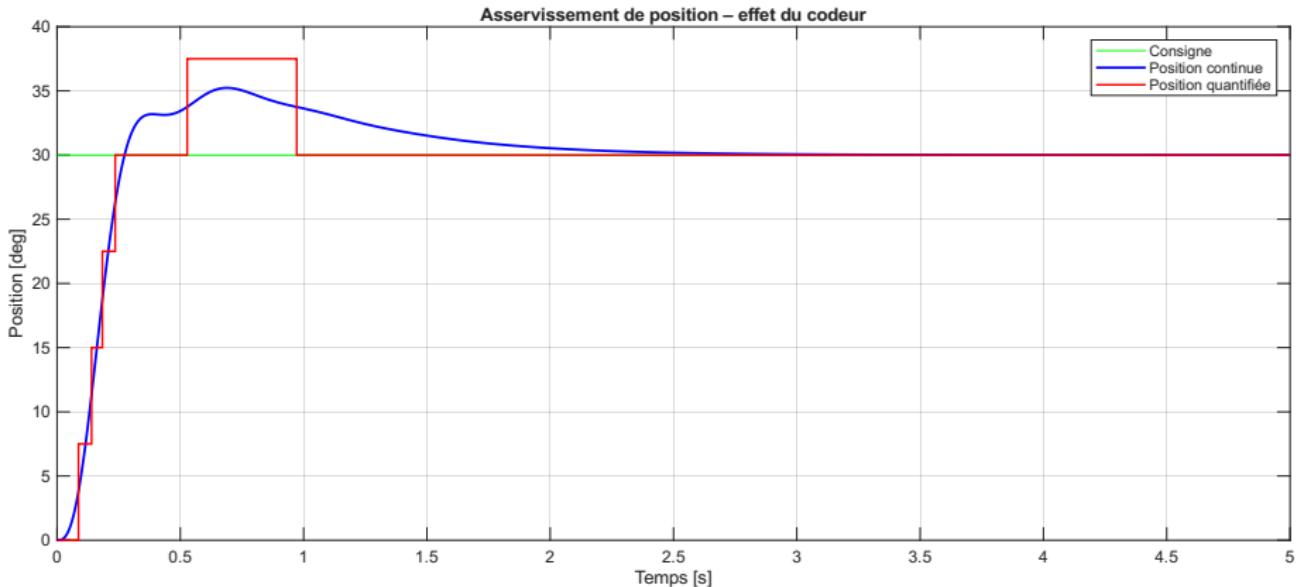


Figure 12 – Réponse échelon  $30^\circ$

1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion

Après tests unitaire et V1 → montage final du boîtier, avec soudures, montage des inserts et assemblage

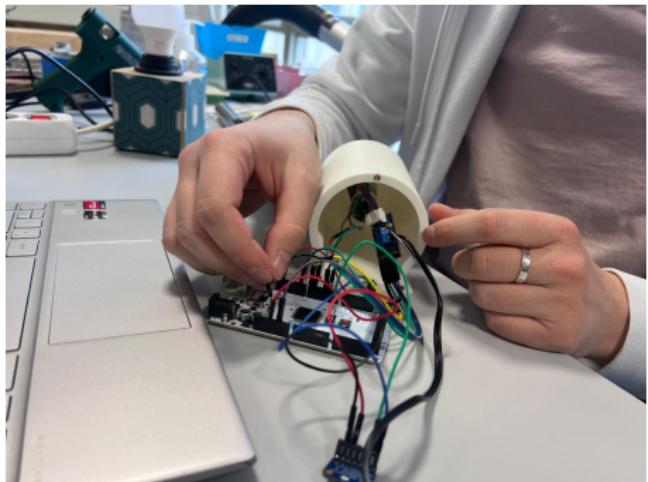


Figure 13 – Assemblage des différentes pièces

Boîtier fonctionnel, esthétique et compact

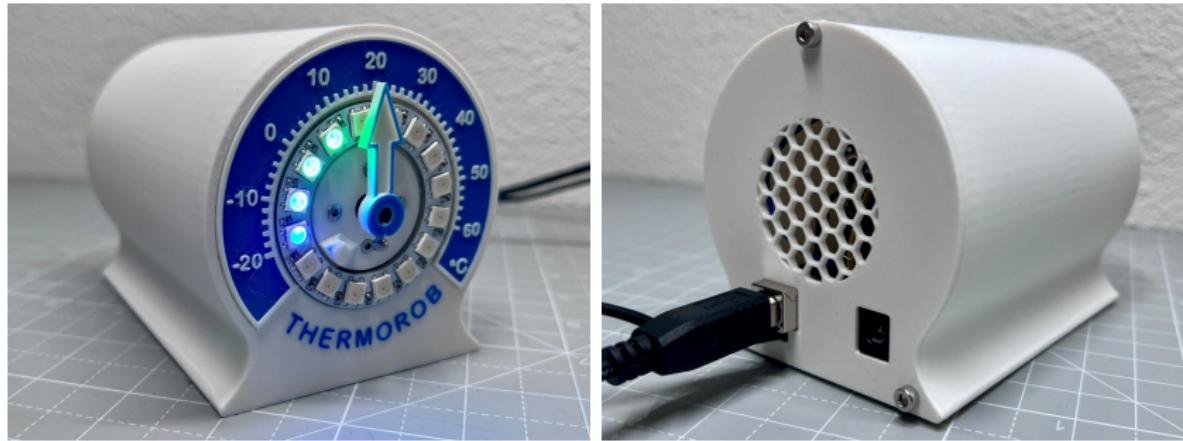


Figure 14 – Thermorob V2

**Protocole :** on monte à 60 degrés avec un sèche cheveux puis on note les couples  $T_{mes}/T_{capteur}$

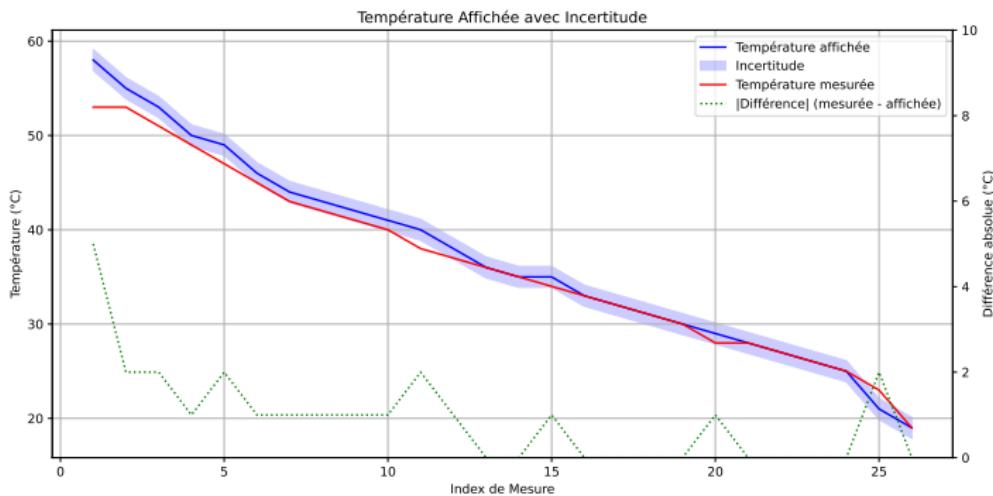


Figure 15 – Tracés expérimentaux

# Thermorob en fonctionnement

29

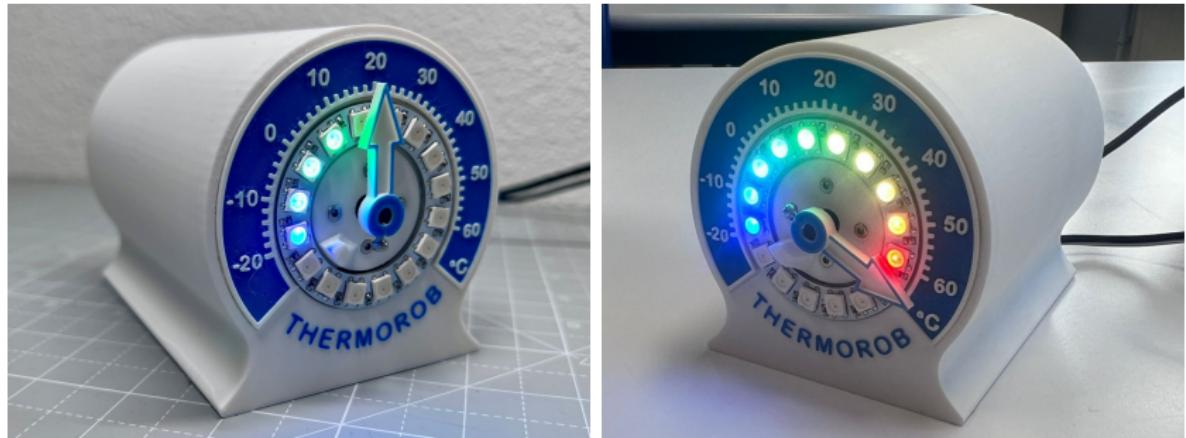


Figure 16 – Réalisation de tests (Température ambiante et chauffé)

# Validation du cahier des charges

30

Fonction	Critère(s)	Niveau(x)	Flexibilité	Validation
FP1 – Affichage précis de la température avec l'aiguille	Précision	-20 à 60°C	±1°C	Validé
	Rapidité	2s	NON	1s
	Stabilité	Aucune oscillation en régime permanent	NON	Aucune oscillation
FP2 – Transportabilité	Masse	< 500g	facile à transporter	230g
	Dimensions du boîtier	Longueur < 10 cm ; Largeur < 10 cm ; Hauteur < 10 cm	NON	L = 11 cm ; l = 7.5 cm ; H = 8.3 cm
FC1 – Modélisation et impression 3D	Utilisation du matériel imposé	Utilisation de tous les composants électroniques essentiels	Ajout d'une ringled Breadboard facultative	Validé
	Limiter l'utilisation de plastique	Nombres d'impression du prototype < 3	NON	Validé

Fonction	Critère(s)	Niveau(x)	Flexibilité	Validation
<b>FS1 – Affichage de la température avec la ringled</b>	Affichage	Leds allumées entre -20°C et la température mesurée	±5°C	Validé
	Signature lumineuse	Dégradé de couleurs bleu(=−20°C) → vert(=20°C) → rouge(=60°C)		
	Consommation électrique	Faible intensité lumineuse		

1. Contexte et objectifs
2. Conception
3. Réalisation
4. Intégration finale et tests
5. Bilan et conclusion



(1) Nouvelle alimentation du moteur



(2) Ajout d'un bouton d'alimentation

- ▶ (3) Améliorer la ventilation du capteur de température en ajoutant des grilles sur les côtés du boîtier



- ✓ Conception et réalisation du robot thermomètre
- ✓ Affichage précis de la température
- ✓ Implémentation d'un correcteur PID
- ✓ Ajout d'une fonctionnalité esthétique : bandeau leds

## Projet utile et dynamisant !

- Montée en compétence en conception : CAO méca, timer, interruption, électronique
- Application sur un sujet concret et appropriation du sujet puisque libre
- Défi sur un temps court, autonomie d'organisation
- Plaisir de travailler avec des gens motivés et compétents



Figure 18 – Thermorob

Et maintenant ... démo !

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

Pour fermer proprement le couvercle arrière, usage d'inserts filetés

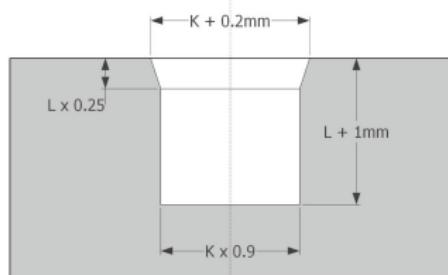


Figure 19 – Préconisation de modélisation

Utilisation du PLA (polylactic acid) :

- ▶ Facilité d'impression : faible retrait thermique, bonne adhérence au plateau
- ▶ Écologique et biodégradable : fabriqué à partir d'amidon de maïs ou de canne à sucre
- ▶ Disponible au fablab

# ANNEXE 3 - Code Arduino

39

```
1 #include <Wire.h>
2 #include <Adafruit_MPL3115A2.h>
3 #include <Encoder.h>
4 #include <RunningMedian.h>
5 #include <FastLED.h>
6
7 // ===== PINS =====
8 #define PIN_PWM 9
9 #define PIN_DIR 7
10#define ENC_A 2
11#define ENC_B 3
12
13#define LED_PIN 6
14#define NUM_LEDS_TOTAL 16 // LEDs physiques
15#define NUM_LEDS_TEMP 10 // affichage temperature
16#define LED_TYPE WS2812B
17#define COLOR_ORDER GRB
18#define LED_OFFSET 3 // decalage horaire
19
20 CRGB leds[NUM_LEDS_TOTAL];
21
22 // ===== SYSTEME MECANIQUE =====
23#define ENCODER_PPR 12 // nb de fentes dans l'encodeur par tour
24#define QUAD_FACTOR 3 // fronts montants et descendants
25#define GEAR_RATIO 19.0
26#define NEEDLE_DEG 210.0 // amplitude en ° entre la position max et min
27
28#define TICKS_PER_MOTOR_REV (ENCODER_PPR * QUAD_FACTOR)
29#define TICKS_PER_OUTPUT_REV (TICKS_PER_MOTOR_REV * GEAR_RATIO)
30#define TICKS_MAX (long)(TICKS_PER_OUTPUT_REV * (NEEDLE_DEG / 360.0))
31#define OFFSET -6 //TICKS
32
33 // ===== TEMP =====
34#define TEMP_LIMIT_MIN -20.0
35#define TEMP_MAX 60.0
```

```
37 // ====== OBJETS ======
38 Adafruit_MPL3115A2 tempSensor;
39 Encoder encoder(ENC_A, ENC_B);
40 RunningMedian samples(10);
41
42 // ====== PID ======
43 volatile float Kp = 0.03; //2.0;
44 volatile float Ki = 0.004; //0.006;
45 volatile float Kd = 0.01; //10.0; //impact du Kd négligeable
46
47 volatile long targetPosition = 0;
48 volatile long currentPosition = 0;
49
50 volatile float error = 0;
51 volatile float prevError = 0;
52 volatile float integral = 0;
53 volatile float output = 0;
54
55 volatile float error_isr = 0;
56 volatile float output_isr = 0;
57
58 #define PWM_MAX 255
59 #define PWM_MIN 60           // vitesse minimale pour garantir la rotation
60 #define INTEGRAL_LIMIT 2000
61
62 // ====== ROUTINE D'INTERRUPTION (200 Hz) ======
63 ISR(TIMER1_COMPA_vect) {
64
65     currentPosition = -encoder.read();
66     error = targetPosition - currentPosition;
67
68     integral += error;
69     integral = constrain(integral, -INTEGRAL_LIMIT, INTEGRAL_LIMIT); // partie integrale bornee
70
71     float derivative = error - prevError;                      // calcul de variation de l'erreur
72     prevError = error;
```

```
74     output = Kp * error + Ki * integral + Kd * derivative;    // calcul du PID
75     output = constrain(output, -PWM_MAX, PWM_MAX);           // sortie bornee
76
77     error_isr = error;
78     output_isr = output;
79
80     if (abs(error) < 2) {
81         digitalWrite(PIN_PWM, 0);      // si l'erreur de position est suffisemment faible, on arrete le moteur
82     } else {
83         int pwm = abs(output);
84         pwm = constrain(pwm, PWM_MIN, PWM_MAX);
85
86         digitalWrite(PIN_DIR, (error > 0) ? LOW : HIGH); // si l'erreur est positive, on tourne dans un sens
87         digitalWrite(PIN_PWM, pwm);                      // sinon dans l'autre
88     }
89 }
90
91 // ===== LED CALIBRATION =====
92 void blinkCalibrationLeds(bool state) {          // fait clignoter les leds en orange durant la calibration
93     CRGB color = state ? CRGB::Orange : CRGB::Black;
94     for (int i = 0; i < NUM_LEDS_TOTAL; i++) {
95         leds[i] = color;
96     }
97     FastLED.show();
98 }
99
100 // ===== CALIBRATION =====
101 void calibrateNeedle() {
102
103     Serial.println("== CALIBRATION EN COURS ==");
104
105     long lastPosition = encoder.read();
106     long current;
107     int stableCount = 0;
108     bool ledState = false;
109     unsigned long lastBlink = 0;
```

```
111 digitalWrite(PIN_DIR, HIGH);           // on fait tourner l'aiguille vers la gauche pour aller sur la butee
112
113 analogWrite(PIN_PWM, 60);             // on lance le moteur à la vitesse min normale
114 delay(130);
115 analogWrite(PIN_PWM, 30);             // on diminue pour se mettre au plus lent et ne pas casser la butee
116
117 while (stableCount < 3) {
118
119     if (millis() - lastBlink > 250) { // on change l'état des leds toutes les 250ms
120         lastBlink = millis();
121         ledState = !ledState;
122         blinkCalibrationLeds(ledState);
123     }
124
125     delay(20);
126     current = encoder.read();
127     long delta = abs(current - lastPosition);
128     lastPosition = current;
129
130     if (delta <= 1) stableCount++;    // l'aiguille est bloquée 3 fois de suite pour sortir de la boucle while
131     else stableCount = 0;
132 }
133
134 analogWrite(PIN_PWM, 0);              // arrêt du moteur
135 encoder.write(0 + OFFSET);           // initialisation du codeur
136
137 FastLED.clear();                   // on éteint les leds
138 FastLED.show();                    // application du clear
139
140 integral = 0;
141 prevError = 0;
142 targetPosition = 0;
143
144 Serial.println("== CALIBRATION OK ==");
145 }
```

```
147 // ===== LED TEMPERATURE =====
148 void updateTemperatureRing(float temperature) {
149
150     temperature = constrain(temperature, TEMP_LIMIT_MIN, TEMP_MAX);
151
152     int activeLeds = map(
153         (int)temperature,
154         (int)TEMP_LIMIT_MIN,
155         (int)TEMP_MAX,
156         1,
157         NUM_LEDS_TEMP
158     );
159     activeLeds = constrain(activeLeds, 1, NUM_LEDS_TEMP);
160
161     FastLED.clear();
162
163     // --- ALLUMAGE DU DÉGRADÉ ---
164     for (int i = 0; i < activeLeds; i++) {
165
166         int ledIndex = (LED_OFFSET + i) % NUM_LEDS_TOTAL;
167
168         uint8_t hue = map(i, 0, NUM_LEDS_TEMP - 1, 160, 0); // bleu → rouge
169         leds[ledIndex] = CHSV(hue, 255, 255);
170     }
171
172     // --- EXTINCTION DES 6 LEDS RESTANTES (ANTI-HORAIRE) ---
173     for (int i = NUM_LEDS_TEMP; i < NUM_LEDS_TOTAL; i++) {
174         int ledIndex = (LED_OFFSET + i) % NUM_LEDS_TOTAL;
175         leds[ledIndex] = CRGB::Black;
176     }
177
178     FastLED.show();
179 }
```

```
181 // ====== TIMER SETUP ======
182 void setupTimer1() {                                // setup timer 1 : 16 bits
183     cli();                                         // désactive l'interruption globale
184     TCCR1A = 0;                                     // registres de controle
185     TCCR1B = 0;
186     OCR1A = 1249;                                  // comparaison des sorties sur timer 1
187     TCCR1B |= (1 << WGM12);                      // activation du mode CTC
188     TCCR1B |= (1 << CS11) | (1 << CS10);        // reglage du prescaler
189     TIMSK1 |= (1 << OCIE1A);                     // flag d'interruption
190     sei();                                         // mise en route des interruptions
191 }
192
193 // ====== SETUP ======
194 void setup() {
195
196     Serial.begin(115200);
197     Wire.begin();
198
199     pinMode(PIN_PWM, OUTPUT);                      // pins de controle du pont en H
200     pinMode(PIN_DIR, OUTPUT);
201
202     FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS_TOTAL);
203     FastLED.setBrightness(10);                    // gestion de la luminosite des leds
204     FastLED.clear();
205     FastLED.show();
206
207     if (!tempSensor.begin()) {                  // initialisation de la com avec le capteur de temp
208         Serial.println("ERREUR CAPTEUR !");
209         while (1);
210     }
211
212     calibrateNeedle();                         // recherche de la butee
213     setupTimer1();                            // demarrage du timer
214 }
```

```
216 // ===== LOOP =====
217 void loop() {
218
219     float temp = tempSensor.getTemperature(); // mesure de la temperature
220     samples.add(temp); // stockage dans la variable du PID
221
222     float temperature = samples.getMedian(); // lissage pour eviter les glitches
223     temperature = constrain(temperature, TEMP_LIMIT_MIN, TEMP_LIMIT_MAX);
224
225     targetPosition = map(
226         (int)temperature,
227         (int)TEMP_LIMIT_MIN, // intervalle de temperatures
228         (int)TEMP_LIMIT_MAX,
229         0, // intervalle des angles
230         TICKS_MAX
231     );
232     targetPosition = constrain(targetPosition, 0, TICKS_MAX); // limitation dans un intervalle realiste
233                                         // par securite materielle
234     updateTemperatureRing(temperature); // allumage des leds correspondantes
235
236     // ----- PRINT TEMP TOUTES LES 2 SECONDES -----
237     static unsigned long lastPrint = 0;
238
239     if (millis() - lastPrint >= 2000) { // affichage de debug sur console Arduino
240         lastPrint = millis();
241         Serial.print("Température mesurée : ");
242         Serial.print(temperature);
243         Serial.println(" °C");
244     }
245     // -----
246
247     delay(50);
248 }
```